A Guide to Agent Tools

Table of contents

1 A Guide to Agent Tools

The power of the `code_agent` lies in its tools. This guide provides a deep dive into each of the built-in tools, explaining what they do and how they are used by the agent.

1.1 Understanding Tools

A tool is a function that the agent can call to interact with its environment. This can be anything from reading and writing files to searching for information or running code. The agent decides which tool to use based on the user's prompt and the context of the conversation. All tools are classes that inherit from `langchain_core.tools.BaseTool`.

1.2 Built-in Tools

Here is a comprehensive list of the tools that come with `code_agent`.

1.2.1 `new-file`

- **Purpose**: Creates a new file with specified content.
- **Why it's useful**: This is the primary tool for creating new source files, documentation, or any other text-based file in your project.
- **Agent Usage**: > **User**: "Create a new Python file named `utils.py` in the `src` directory with a function that adds two numbers."

1.2.2 `edit-file`

- **Purpose**: Modifies an existing file. It supports three modes:
  - `replace`: Overwrites the entire file with new content.
  - `append`: Adds the new content to the end of the file.
  - `patch`: Intelligently inserts content, often used for updating specific sections.
- **Why it's useful**: Use this tool to add new functions to existing code, update documentation, or correct errors.
- **Agent Usage**: > **User**: "Add a new function to `src/utils.py` that subtracts two numbers."

1.2.3 `read-file`

- **Purpose**: Reads the content of a specified file.
- **Why it's useful**: Allows the agent to get context from existing files before making changes.
- **Agent Usage**: > **User**: "Read the content of `README.md` and tell me what it says."

1.2.4 `search-explain`

- **Purpose**: Searches for a specific code snippet or keyword in your project and uses the LLM to explain it.
- **Why it's useful**: Invaluable for understanding unfamiliar codebases. Instead of manually searching for definitions, you can ask the agent to find and explain a function or class.
- **Agent Usage**: > **User**: "Find the `build_agent` function and explain what it does."

1.2.5 `generate-test`

- **Purpose**: Generates a new `pytest` test file for a given Python source file.
- **Why it's useful**: Automates the tedious process of creating boilerplate test code.
- **Agent Usage**: > **User**: "Generate tests for `src/utils.py`."

1.2.6 `format-code`

- **Purpose**: Formats a code file using standard formatters (like Black or Ruff).
- **Why it's useful**: Helps maintain a consistent and readable code style across the project.
- **Agent Usage**: > **User**: "Format the file `src/utils.py`."

1.2.7 `notebook`

- **Purpose**: Executes Python code in a sandboxed, notebook-like environment.
- **Why it's useful**: Allows the agent to run code, test solutions, and perform data analysis tasks dynamically.
- **Agent Usage**: > **User**: "Calculate the first 10 Fibonacci numbers and print them."

1.2.8 `linker`

- **Purpose**: Analyzes dependencies and relationships between files in your project.
- **Why it's useful**: Helps visualize the structure of your codebase, which is essential for large-scale refactoring.
- **Agent Usage**: > **User**: "What are the dependencies for `code_agent/main.py`?"

1.2.9 `r-script`

- **Purpose**: Executes R code.
- **Why it's useful**: Enables the agent to perform statistical analysis and data visualization using R.
- **Agent Usage**: > **User**: "Create an R script that plots a histogram of 100 random numbers."

1.3 Extending the Agent with New Tools

One of the most powerful features of `code_agent` is its extensibility.

1.3.1 1. Create the Tool Class

Create a new file in the `code_agent/tools/` directory. Your tool should be a class that inherits from `BaseTool` and defines its `name`, `description`, and `_run` method.

```
# In code_agent/tools/my_custom_tool.py
from langchain_core.tools import BaseTool

class MyCustomTool(BaseTool):
    name: str = "my-custom-tool"
    description: str = "A simple tool that returns a custom greeting."

    def _run(self, name: str) -> str:
        """Returns a greeting for the given name."""
        return f"Hello, {name}!"
```

1.3.2 2. Register the New Tool

Add your new tool to the `create_default_tools` function in `code_agent/agents/base_agent.py`.

```
# In code_agent/agents/base_agent.py
from code_agent.tools.my_custom_tool import MyCustomTool

def create_default_tools(...) -> list[BaseTool]:
    ...
    standard_tools: list[BaseTool | None] = [
        ...,
        MyCustomTool(),  # Add your new tool here
    ]
    ...
```

That's it! The agent can now use your new tool when it determines it's appropriate based on the user's prompt.