Usage

Sofia Billger Bergström

This page provides a variety of examples to demonstrate how to use `code_agent` for common development tasks.

CLI Usage

The command-line interface is perfect for quick, one-off tasks.

Scaffold a New Web Application

Create a new project named `my_web_app` designed for a Flask application.

```
# Create the project directory and scaffold the structure
mkdir my_web_app
code-agent scaffold ./my_web_app --name my_web_app
```

This will create a full project structure inside the `my_web_app` directory.

Create and Populate a File

Now, let's create a basic Flask application file. Since passing multiline strings can be tricky in some terminals, it's often easier to do this in two steps.

```
# 1. Create an empty file
code-agent create ./my_web_app/src/my_web_app/app.py --content ""

# 2. Use the interactive agent or your editor to add the Flask content
```

```
# Flask app content
"""
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, from Code Agent!'

if __name__ == '__main__':
    app.run(debug=True)
"""
```

Convert a Script to a Notebook

If you have a Python script for data analysis, you can easily convert it to a Jupyter notebook to visualize the results.

```
# Your script, analysis.py
# %%
import pandas as pd
import matplotlib.pyplot as plt

# %%
data = {'col1': [1, 2], 'col2': [3, 4]}
df = pd.DataFrame(data)
print(df)

# %%
plt.plot(df['col1'], df['col2'])
plt.show()
```

Run the conversion command:

```
code-agent py2ipynb analysis.py analysis.ipynb
```

This will create `analysis.ipynb` with three distinct cells.

Interactive Agent Session

For more complex or iterative tasks, the interactive agent is the best tool.

Start the agent:

```
code-agent chat
```

You can now give instructions to the agent in plain English.

Example:

```
You: Create a new file named 'utils.py' in the 'src/my_web_app' directory.
```

The agent will use the `new-file` tool to create the file.

Python API Usage

The Python API gives you full control to integrate `code_agent` into your own applications and scripts. This example shows how to use the agent to add a function to an existing file.

```
from code_agent.agents.base_agent import build_agent, create_default_tools
from code_agent.main import create_llm, load_config
from code_agent.file_generator import write_file
from langchain_core.messages import HumanMessage
from pathlib import Path
import os

# --- 1. Setup ---
# Create a temporary project directory for this example
project_dir = Path("./api_project_example")
project_dir.mkdir(exist_ok=True)
os.chdir(project_dir)

# Load configuration and create LLM
config = load_config()
llm = create_llm(config)

# Create an initial file to be edited
initial_file = Path("my_module.py")
write_file(initial_file, "# This is a module for utility functions.\n")
```

```
# Create tools and build the agent
tools = create_default_tools(root_dir=".", llm=llm)
agent_app = build_agent(llm=llm, tools=tools)

# --- 2. Define the Task ---
prompt = (
    "Add a Python function to 'my_module.py' that takes two numbers and returns their sum. "
    "The function should be named 'add_numbers'."
)

print(f"--- Prompting Agent ---\n{prompt}\n-------------------------")

# --- 3. Invoke the Agent ---
# The agent will process the prompt, select the 'edit-file' tool, and apply the change.
response = agent_app.invoke({"messages": [HumanMessage(content=prompt)]})

# --- 4. Verify the Result ---
final_content = initial_file.read_text()
print(f"\n--- Final Content of {initial_file} ---")
print(final_content)
print("-------------------------------------")

# Clean up the temporary directory
os.chdir("..")
# You would typically remove the directory in a real script, but we leave it for inspection
# import shutil
# shutil.rmtree(project_dir)
```

This example demonstrates a complete, end-to-end workflow using the Python API to modify a file based on a natural language prompt.