Examine GitHub Copilot prompts and keywords

6 minutes

GitHub Copilot's chat interface uses natural language processing and machine learning to interpret your prompts and provide relevant responses. Your prompts and the context that you provide are essential for generating accurate and useful responses.

Prompt quality

When you write prompts for GitHub Copilot, consider the following guidelines to improve the quality of the responses:

- The prompts that you submit should be clear, concise, and specific. For example, instead of asking "How do I use this function?", you could ask "Can you explain how the `calculateTotal` function works in the `shoppingCart.js` file?".
- When you need to write a longer prompt, the prompt should be written using several short sentences. Start with an overview that describes your goal and then provide specific details.
- Consider breaking complex prompts into smaller, more manageable parts. Breaking down complex prompts helps the AI to understand your intent and provide more accurate responses. Subsequent prompts can build on the previous ones, allowing you to refine your request and get more specific answers. Working with the peer programmer is similar to working with a person. It's better to frame your conversation first, and then describe the specific details rather than sending everything in a single request that may be misinterpreted.

Use prompt keywords

In addition to processing natural language text, GitHub Copilot Chat uses keywords to help you specify the context and intent of your prompts. Keywords are special words or phrases that have specific meanings in the context of GitHub Copilot Chat. By using keywords, you can help the AI understand what you're asking for and provide more relevant responses.

GitHub Copilot uses the following keywords to help you specify the context and intent of your prompts:

- Chat participants: Chat participants are like experts in a specific field that help GitHub Copilot generate better responses. Chat participants are specified using the `@` symbol.
- Slash commands: Slash commands help to describe the intent (the goal or objective) of your prompt. One of Copilot Chat's tasks when answering questions is to determine the intent, understanding what you want to do. Slash commands can help clarify your intent.
- Chat variables: Chat variables provide domain-specific context. You can reference a chat variable in your chat prompt by using the `#` symbol. By using a chat variable, you can be more specific about the context that you include in your chat prompt.

Chat participants

GitHub Copilot Chat provides better responses when it understands the context that you're interested in and can apply the best resources. You can help GitHub Copilot Chat generate better responses by adding a Chat participant to your prompts. Chat participants are like experts in a specific field that provide better responses. Chat participants are specified using the `@` symbol.

Currently, Copilot Chat supports the following built-in chat participants:

- `@workspace`: The `@workspace` participant can help answer questions about the code in your workspace or suggest ways to refactor or improve your code.
- `@vscode`: The `@vscode` participant knows about commands and features in the Visual Studio Code editor itself, and can help you use them.
- `@terminal`: The `@terminal` participant can help with the integrated terminal shell and its contents.
- `@github`: The `@github` participant can help get answers grounded in web search, code search, and your enterprise's knowledge bases.

You can prefix your prompt with a specific chat participant to help Copilot generate a more relevant response.

Slash commands

Slash commands help Copilot Chat understand your intent when you ask a question. Are you learning about a code base (`/explain`), do you want help with fixing an issue (`/fix`), or are you creating test cases (`/tests`)? By letting Copilot Chat know what you're trying to do, it can tune its reply to your task and provide helpful commands, settings, and code snippets.

Chat participants are often bundled with slash commands. The slash command is a concise way to explain your intent to the chat participant. The `/explain` slash command is often bundled with the `@workspace` chat participant. The combination of a chat participant and a slash command is a powerful way to clarify your intent.

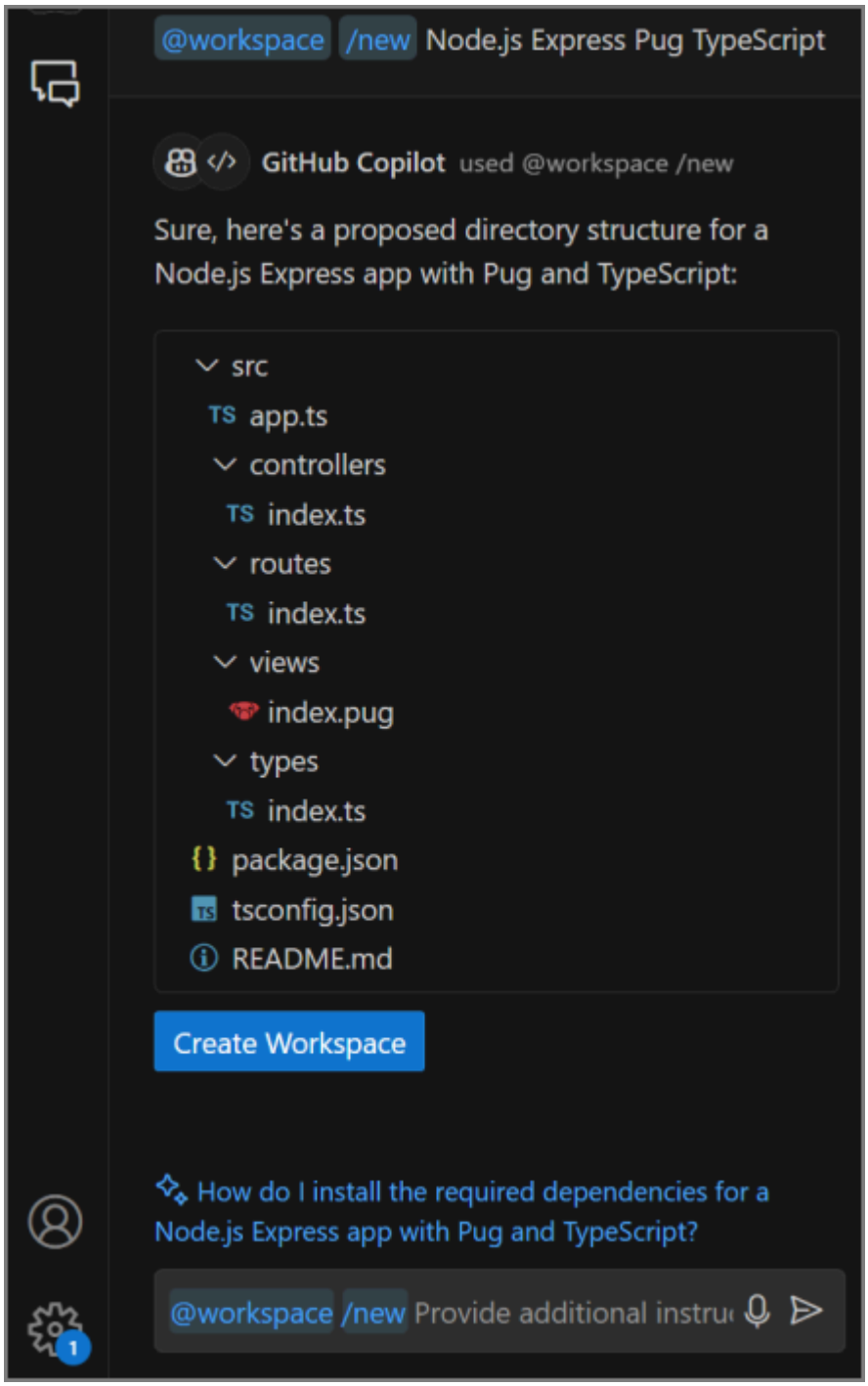
For example, consider the following sample prompt:

```plaintext
Create a new workspace that uses Node.js, the Express framework, the Pug template engine, and TypeScript.
```

GitHub Copilot may interpret this prompt as a request for new code project created using the specified technologies, but it's easier for GitHub Copilot to understand the following prompt that uses keywords:

```plaintext
@workspace /new Node.js Express Pug TypeScript
```

If you enter the second prompt, GitHub Copilot proposes the following directory structure. The file list can be used to preview the proposed files, and the **Create Workspace** button can be used to generate the files in a new folder.



Examples of built-in slash commands:

- `/help`: Get help about using GitHub Copilot.
- `/doc`: Generate code documentation.
- `/clear`: Start a new chat session.
- `/explain`: Explain how the selected code works.
- `/tests`: Generate unit tests for the selected code.
- `/fix`: Propose a fix for the selected code.
- `/new`: Scaffold code for a new workspace. Only the chat prompt is used as context.
- `/newNotebook`: Create a new Jupyter Notebook. Only the chat prompt is used as context.

Examples using chat participants combined with slash commands:

- `@workspace /explain`: Generate an explanation of the full workspace.
- `@workspace /fix` (or `/fix`): Propose a fix for the problems in the selected code.
- `@workspace /tests` (or `/tests`): Generate unit tests for the selected code.
- `@vscode /api` (or `/api`): Ask about Visual Studio Code extension development.
- `@workspace /new` (or `/new`): Scaffold code for a new workspace.
- `@workspace /newNotebook` (or `/newNotebook`): Create a new Jupyter Notebook.

Chat variables

Chat variables are used to specify context. You can reference a chat variable in your chat prompt by using the `#` symbol. For example, the `#selection` variable contains the text selection in the active editor.

By using a chat variable, you can be more specific about the context that you include in your chat prompt. For example, the prompt "which sorting algorithm is used in `#selection`" focuses the chat request on the selected code snippet.

Chat participants, such as @workspace or @vscode, can contribute chat variables that provide domain-specific context.

Examples of built-in chat variables are:

- `#editor`: the visible source code in the active editor.
- `#selection`: the current selection in the active editor. The editor content is implicitly included in the Chat view context.
- `#<file or folder name>`: type #, followed by a file or folder name, to add it as chat context.
- `#codebase`: add relevant workspace content as context to your prompt.
- `#terminalSelection`: the active terminal's selection.
- `#terminalLastCommand`: the active terminal's last run command.

What's the difference between `@workspace` and `#codebase`?

Conceptually, both `@workspace` and `#codebase` enable you to ask questions about your entire codebase. However, there are some differences in how you can use them:

About the `@workspace` keyword:

- The `@workspace` keyword is a chat participant that's dedicated to answering questions about your codebase.

- The `@workspace` keyword takes control of the user prompt and uses the codebase to provide an answer.
- The `@workspace` keyword can't invoke other tools.
- The `@workspace` keyword can only be used when you're using the ask mode.
- Example: `"@workspace how can I validate a date?"`

About the `#codebase` keyword:

- The `#codebase` keyword is a tool that performs a codebase search based on the user prompt and adds the relevant code as context to the chat prompt.
- When you use the `#codebase` keyword, the language model remains in control and can combine `#codebase` with other tools for editing scenarios.
- The `#codebase` keyword can be used in all chat modes (Ask, Edit, and Agent).
- Examples: `"add a tooltip to this button, consistent with other button #codebase"`, `"add unit tests and run them #codebase"`

GitHub recommends using `#codebase` in your chat prompts, as it provides more flexibility.

Summary

GitHub Copilot's chat interface uses natural language processing and machine learning to interpret your prompts and provide relevant responses. Your prompts and the context that you provide are essential for generating accurate and useful responses. You can improve the quality of the responses by using clear, concise, and specific prompts, breaking complex prompts into smaller parts, and using keywords such as chat participants, slash commands, and chat variables.

Next unit: Analyze and explain code using GitHub Copilot

Previous Next