GitHub Copilot can review your code and provide feedback. Where possible, Copilot's feedback includes suggested changes which you can apply with a couple of clicks.

**Using Copilot code review**

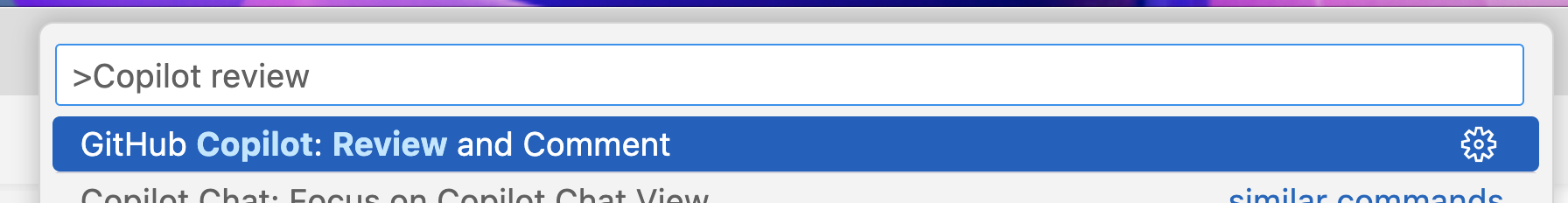
These instructions explain how to use Copilot code review in Visual Studio Code. To see instructions for other popular coding environments, use the tool switcher at the top of the page.

*Note : Copilot code review is available in Visual Studio Code with version 0.22 or later of the GitHub Copilot Chat extension.*

**Reviewing a selection of code**

You can request an initial review of a highlighted selection of code in Visual Studio Code.

1. In Visual Studio Code, select the code you want to review.
2. Open the VS Code Command Palette
   * For Mac:
     + Use: Shift+Command+P
   * For Windows or Linux:
     + Use Ctrl+Shift+P
3. In the command palette, search for and select **GitHub Copilot: Review and Comment**.

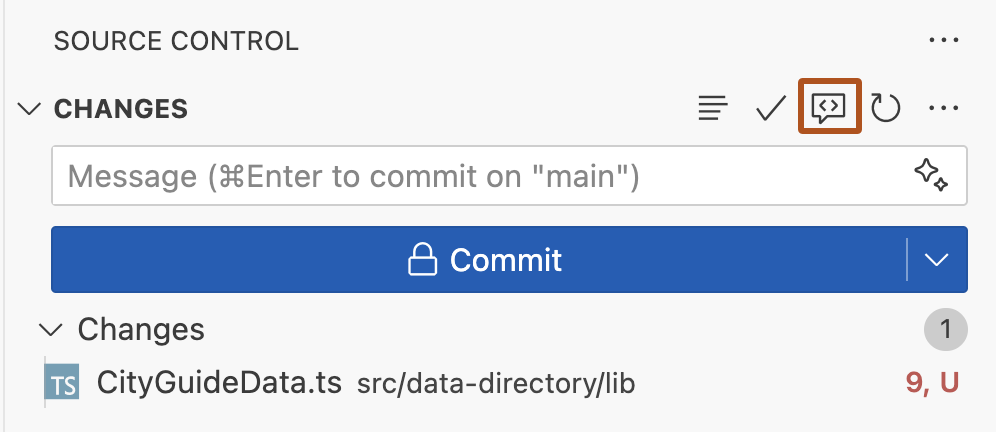


1. Wait for Copilot to review your changes. This usually takes less than 30 seconds.
2. If Copilot has any comments, they will be shown inline in your file, and in the **Problems** tab.

**Reviewing all uncommitted changes**

You can request a review of your uncommitted changes in Visual Studio Code.

1. In VS Code, click the **Source Control** button in the Activity Bar.
2. At the top of the **Source Control** view, hover over **CHANGES**, then click the  **Copilot Code Review - Uncommitted Changes** button.



1. Wait for Copilot to review your changes. This usually takes less than 30 seconds.
2. If Copilot has any comments, they will be shown inline in your file(s), and in the **Problems** tab.

**Working with suggested changes provided by Copilot**

Where possible, Copilot's feedback includes suggested changes which you can apply with a single click.



If you're happy with the change, you can accept a suggestion from Copilot by clicking the **Apply and Go To Next** button. Any changes you apply will not be automatically committed.

If you don't want to apply Copilot's suggested change, click the **Discard and Go to Next** button.

**Providing feedback on Copilot's reviews**

You can provide feedback on Copilot's comments directly within each comment. We use this information to improve the product and the quality of Copilot's suggestions.

To provide feedback, hover over the comment and click the thumbs up or thumbs down button.



**Customizing Copilot's reviews with custom instructions**

You can customize Copilot code review by adding a .github/copilot-instructions.md file in your repository containing information that you want Copilot to consider when reviewing code. This is the same copilot-instructions.md used by Copilot Chat. See Adding repository custom instructions for GitHub Copilot (below).

**Example**

This example of a .github/copilot-instructions.md file contains three instructions that will be applied to all Copilot code reviews in the repository.

When performing a code review, respond in Spanish.

When performing a code review, follow our internal security checklist.

When performing a code review, focus on readability and avoid nested ternary operators.

**Adding repository custom instructions for GitHub Copilot**

Create a file in a repository that gives Copilot additional context for the work it does in that repository.

**About repository custom instructions for Copilot**

Repository custom instructions let you provide Copilot with repository-specific guidance and preferences.

Repository custom instructions are currently supported for:

* **Copilot Chat** in VS Code
* **Copilot coding agent**
* **Copilot Chat** in Visual Studio, JetBrains IDEs, Xcode, Eclipse, and on the GitHub website (copilot-instructions.md file only)
* **Copilot code review** (copilot-instructions.md file only)

**Prerequisites for repository custom instructions**

* You must have a custom instructions file (see the instructions below).
* The **Use Instruction Files** option must be enabled in your settings. This is enabled by default.

**Creating a repository custom instructions file**

VS Code supports either:

* A single .github/copilot-instructions.md custom instructions file stored in the repository
* One or more .instructions.md files stored within .github/instructions in the repository. Each file can specify applyTo frontmatter to define what files or directories its instructions apply to.

**Using a single .github/copilot-instructions.md file**

1. In the root of your repository, create a file named .github/copilot-instructions.md.

Create the .github directory if it does not already exist.

1. Add natural language instructions to the file, in Markdown format.

Whitespace between instructions is ignored, so the instructions can be written as a single paragraph, each on a new line, or separated by blank lines for legibility.

**Using one or more .instructions.md files**

1. Create the .github/instructions directory if it does not already exist.
2. Create one or more .instructions.md files, adding natural language instructions to the file(s).

Whitespace between instructions is ignored, so the instructions can be written as a single paragraph, each on a new line, or separated by blank lines for legibility.

1. Specify what files or directories the instructions apply to by adding applyTo frontmatter to the Markdown files, using glob syntax.

---

**applyTo: "app/models/\*\*/*\*.rb"***

***---***

1. ***Add custom instructions here***

To apply the instructions to all files, use the \*\* pattern.

Did you successfully add a custom instructions file to your repository?

**Writing effective repository custom instructions**

The instructions you add to your custom instruction file(s) should be short, self-contained statements that provide Copilot with relevant information to help it work in this repository. Because the instructions are sent with every chat message, they should be broadly applicable to most requests you will make in the context of the repository.

The exact structure you utilize for your instructions file(s) will vary by project and need, but the following guidelines provide a good starting point:

* Provide an overview of the project you're working on, including its purpose, goals, and any relevant background information.
* Include the folder structure of the repository, including any important directories or files that are relevant to the project.
* Specify the coding standards and conventions that should be followed, such as naming conventions, formatting rules, and best practices.
* Include any specific tools, libraries, or frameworks that are used in the project, along with any relevant version numbers or configurations.

The following instructions file(s) is an example of these practices in action:

**# Project Overview**

This project is a web application that allows users to manage their tasks and to-do lists. It is built using React and Node.js, and uses MongoDB for data storage.

**## Folder Structure**

- `/src`: Contains the source code for the frontend.

- `/server`: Contains the source code for the Node.js backend.

- `/docs`: Contains documentation for the project, including API specifications and user guides.

**## Libraries and Frameworks**

- React and Tailwind CSS for the frontend.

- Node.js and Express for the backend.

- MongoDB for data storage.

**## Coding Standards**

- Use semicolons at the end of each statement.

- Use single quotes for strings.

- Use function based components in React.

- Use arrow functions for callbacks.

**## UI guidelines**

- A toggle is provided to switch between light and dark mode.

- Application should have a modern and clean design.

You should also consider the size and complexity of your repository. The following types of instructions may work for a small repository with only a few contributors, but for a large and diverse repository, **these may cause problems**:

* Requests to refer to external resources when formulating a response
* Instructions to answer in a particular style
* Requests to always respond with a certain level of detail

For example, the following instructions **may not have the intended results**:

Always conform to the coding styles defined in styleguide.md in repo my-org/my-repo when generating code.

Use @terminal when answering questions about Git.

Answer all questions in the style of a friendly colleague, using informal language.

Answer all questions in less than 1000 characters, and words of no more than 12 characters.

**Repository custom instructions in use**

The instructions in the file(s) are available for use by Copilot Chat as soon as you save the file(s). The complete set of instructions will be automatically added to requests that you submit to Copilot in the context of that repository. For example, they are added to the prompt you submit to Copilot Chat.

Custom instructions are not visible in the Chat view or inline chat, but you can verify that they are being used by Copilot by looking at the References list of a response in the Chat view. If custom instructions were added to the prompt that was sent to the model, the .github/copilot-instructions.md file is listed as a reference. You can click the reference to open the file.



**Enabling or disabling repository custom instructions**

You can choose whether or not you want Copilot to use repository-based custom instructions.

**Enabling or disabling custom instructions for Copilot Chat**

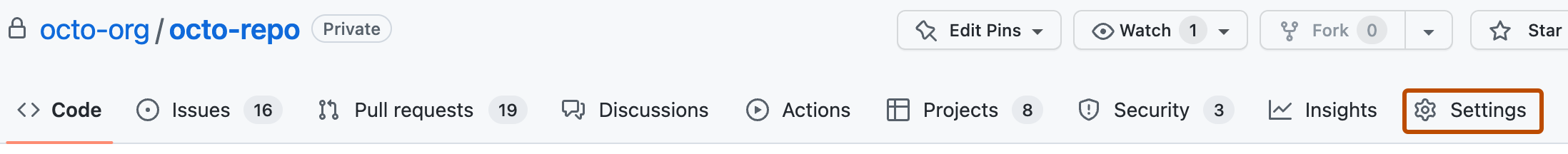
Custom instructions are enabled for Copilot Chat by default but you can disable, or re-enable, them at any time. This applies to your own use of Copilot Chat and does not affect other users.

1. Open the Setting editor by using the keyboard shortcut Command+, (Mac) / Ctrl+, (Linux/Windows).
2. Type instruction file in the search box.
3. Select or clear the checkbox under **Code Generation: Use Instruction Files**.

**Enabling or disabling custom instructions for Copilot code review**

Custom instructions are enabled for Copilot code review by default but you can disable, or re-enable, them in the repository settings on GitHub.com. This applies to Copilot's use of custom instructions for all code reviews it performs in this repository.

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**. If you cannot see the "Settings" tab, select the  dropdown menu, then click **Settings**.



1. In the "Code & automation" section of the sidebar, click **Copilot**, then **Code review**.
2. Toggle the “Use custom instructions when reviewing pull requests” option on or off.

**Enabling and using prompt files**

*Note: Prompt files are public preview and subject to change.*

Prompt files let you build and share reusable prompt instructions with additional context. A prompt file is a Markdown file, stored in your workspace, that mimics the existing format of writing prompts in Copilot Chat (for example, Rewrite #file:x.ts). You can have multiple prompt files in your workspace, each of which defines a prompt for a different purpose.

**Enabling prompt files**

To enable prompt files, configure the workspace settings.

1. Open the command palette by pressing Ctrl+Shift+P (Windows/Linux) / Command+Shift+P (Mac).
2. Type "Open Workspace Settings (JSON)" and select the option that's displayed.
3. In the settings.json file, add "chat.promptFiles": true to enable the .github/prompts folder as the location for prompt files. This folder will be created if it does not already exist.

**Creating prompt files**

1. Open the command palette by pressing Ctrl+Shift+P (Windows/Linux) / Command+Shift+P (Mac).
2. Type "prompt" and select **Chat: Create Prompt**.
3. Enter a name for the prompt file, excluding the .prompt.md file name extension. The name can contain alphanumeric characters and spaces and should describe the purpose of the prompt information the file will contain.
4. Write the prompt instructions, using Markdown formatting.

You can reference other files in the workspace by using Markdown links—for example, [index](../../web/index.ts)—or by using the #file:../../web/index.ts syntax. Paths are relative to the prompt file. Referencing other files allows you to provide additional context, such as API specifications or product documentation.

**Using prompt files**

1. At the bottom of the Copilot Chat view, click the **Attach context** icon ().
2. In the dropdown menu, click **Prompt...** and choose the prompt file you want to use.
3. Optionally, attach additional files, including prompt files, to provide more context.
4. Optionally, type additional information in the chat prompt box.

Whether you need to do this or not depends on the contents of the prompt you are using.

1. Submit the chat prompt.