How to write general rules in Cursor - Creating general rules in Cursor

This is a complete guide on how to create and write Cursor Rules (.mdc) in Cursor IDE 2026.

Cursor Rules is one of the most powerful features of Cursor IDE in 2026. It's a persistent instruction system that helps AI understand your coding style, project conventions, best practices, and specific requirements.

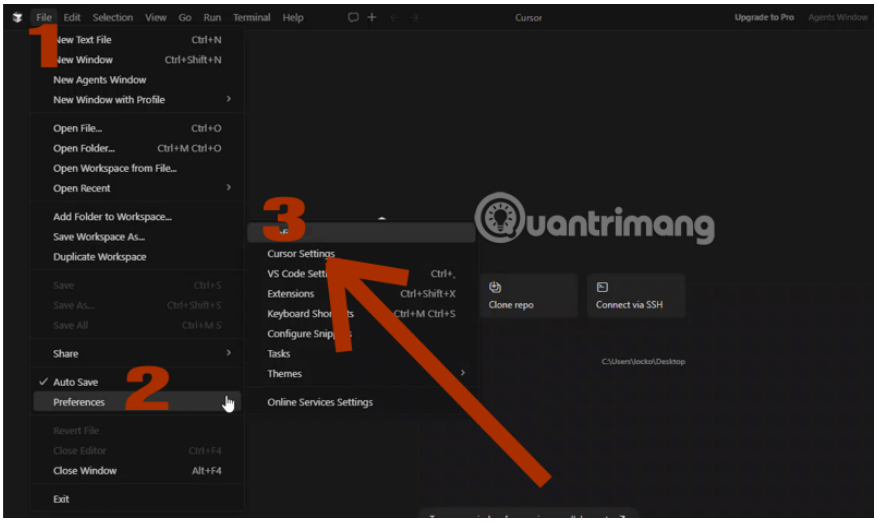
Instead of repeating the request in every prompt, you only need to write the rules once – Cursor will automatically apply them to Chat, Composer, Agent Mode, and Inline Edit.

Rules now use the **.mdc** format (Markdown with YAML frontmatter) and are primarily stored in the **.cursor/rules/** directory . The old .cursorrules file still supports it but is no longer the recommended method. If you only want to create individual rules for each project within Cursor, see: **Cursor Rules Setup Guide** .

Instructions on creating general rules in Cursor

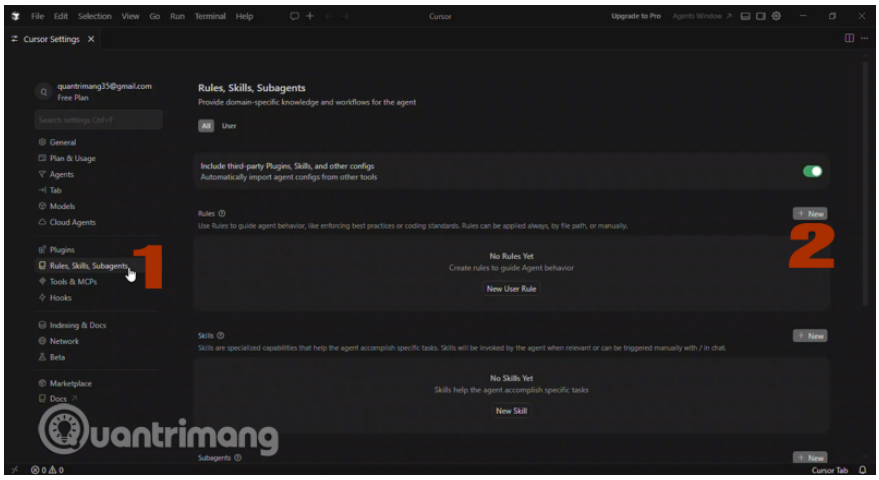
Step 1: Open Cursor IDE

Launch Cursor on your computer (Windows , macOS , or Linux). Press **Cmd + ,** (Mac) or **Ctrl + ,** (Windows/Linux), or go to the **File menu (1) ? Preferences (2) ? Cursor Settings (3)** . Alternatively, you can also use the Command Palette (**Cmd + Shift + P**) and type 'Cursor Settings'.



Step 2: Access the Rules tab and create a new rule.

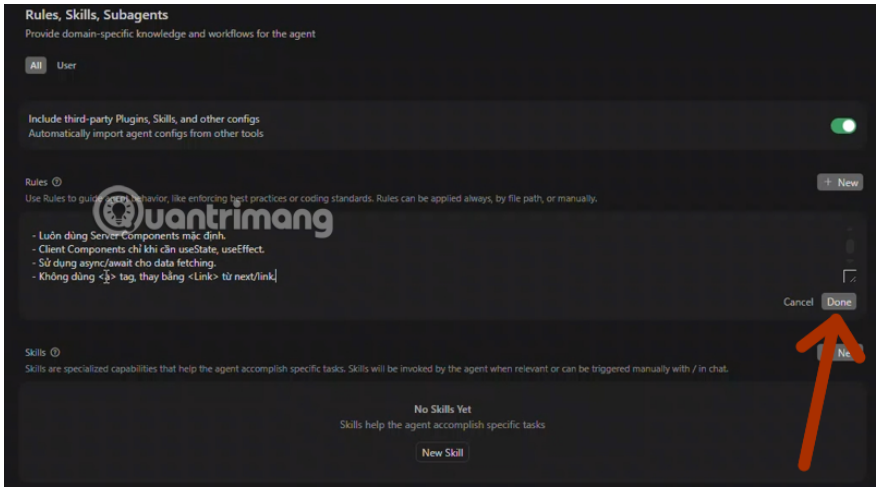
In the Settings window, find and select **Rules, Commands** (or **Rules, Skills, Subagents** depending on the interface) - (1). This is where all User Rules, Project Rules and Team Rules are managed. Select the type: **All** (applies to all projects) or User (applies only to the current project). Cursor will automatically create a .mdc file in the .cursor/rules/ folder.



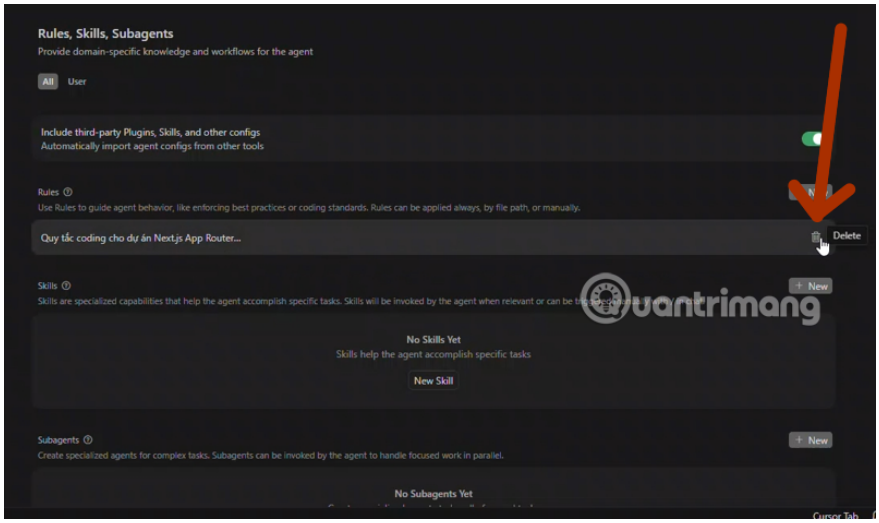
Enter a clear **Name** (rule name) and **Description** . The Description helps the AI ??understand when to apply this rule.

Write the rule content using the **.mdc** format with the YAML frontmatter at the beginning of the file, followed by the instructions in Markdown. Save the file, and Cursor will automatically load the rule.

After you have completed the content in the Rules section, select Done to save.



Select Delete if you want to cancel the created rules; you can also add new rules by selecting New.



Step 4: Check and apply

Open Composer or Chat and run a simple prompt to check if the rule is applied. You can see a list of active rules in Settings.

Standard .mdc file structure 2026

A basic .mdc file has the following format:

```
--- description: Quy t?c coding cho d?
 án Next.js App Router globs: ["**/*.tsx", "**/*.ts"] alwaysApply: true --- # H
??ng d?n cho AI - Luôn s? d?ng TypeScript strict mode. - Component ph?
i là functional component v?i React 19. - S? d?ng Tailwind CSS cho styling.
```

1. **Description** : Briefly describe the purpose of the rule.
2. **globs** : Apply rules only to certain file types (e.g., only .tsx files).

3. **alwaysApply** : true so that the rule is always loaded.

Tips for writing effective Cursor Rules

1. **The more specific the better** : Avoid vague statements. Provide code examples before and after writing.
2. **To use an example** : AI learns very well from real-world examples (before/after).
3. **Keep rules concise** : A rule should focus on one topic (naming, component structure, API handling, etc.).
4. **Combine multiple .mdc files** : Instead of one huge file, split it into several smaller files in the .cursor/rules/ directory.

Some examples of common Cursor Rules

1. Rules regarding Naming Convention

```
--- description: Enforce consistent naming for React components and files always/
?i dùng PascalCase. - File component ph?i kh?p tên: `UserProfile.tsx` ch?
a component `UserProfile`. - Bi?
n, hàm dùng camelCase. - Constants dùng UPPER_SNAKE_CASE.
```

2. Rules for Next.js Project

```
--- description: Next.js 15 App Router best practices globs: ["**/*.{ts,tsx}"] -
?c ??nh. - Client Components ch? khi c?n useState, useEffect. - S? d?
ng async/await cho data fetching. - Không dùng  tag, thay b?ng t?
 next/link.
```

3. Error Handling Rules

```
--- description: Consistent error handling pattern --- Khi x? lý l?i: - S?
 d?ng try/catch. - Log l?i v?i console.error và g?i report n?
u production. - Tr? v? thông báo thân thi?n cho user. Ví d?
: ```ts try { // code } catch (error) { console.error("Failed to fetch data:", e
```

Benefits of using good Cursor Rules

With well-written rules, Cursor Composer and Agent Mode will:

1. Automatically conforms to your code style.
2. Reduce recurring errors (inconsistent naming, wrong import, etc.).
3. Significantly increased growth rate.
4. Help the team work more effectively (using Project Rules or Team Rules).

Many developers report a 2-4 fold increase in productivity when they have personalized rule sets.

How to manage and optimize Rules in 2026

1. **User Rules** : Applied globally (personal coding style).
2. **Project Rules** : Specific to each project (framework, architecture).
3. Use the **/create-rule** command in Chat or Agent to have Cursor automatically generate rules based on the description.
4. Regularly review and update the rules as the project changes.
5. Combine this with your old **.cursorrules** if you are migrating.

Cursor Rules are essentially a personalized 'system prompt' for AI coding assistants. By writing clear, specific rules with examples, you're turning Cursor into a real programmer colleague – always understanding your intentions and writing code that meets your standards.