Contents

The Complete Guide to Cursor AI

Table of Contents

Introduction

Cursor AI is a revolutionary AI-powered code editor built as a fork of Visual Studio Code. It seamlessly integrates advanced artificial intelligence capabilities directly into your development workflow, making coding faster, more efficient, and more intuitive than ever before.

What Makes Cursor Special?

- **AI-First Design**: Every feature is designed with AI assistance in mind
- **Context-Aware Intelligence**: Understands your entire codebase
- **Natural Language Interface**: Interact with your code using plain English
- **Real-Time Assistance**: Get help as you type, not after
- **Privacy-Focused**: Your code can stay completely private

Getting Started

Installation

1. Visit cursor.com and download the installer for your operating system
2. Run the installer and follow the setup instructions
3. Import your existing VS Code settings, extensions, and themes if desired
4. Sign up for a Cursor account to access AI features

First Launch

- Cursor will automatically index your codebase when you open a project
- The AI features become more accurate as it learns your project structure
- You can start using AI assistance immediately with keyboard shortcuts

---

Core Features

1. Intelligent Code Completion

Cursor provides context-aware code completions that go far beyond traditional autocomplete:

Real-Time Suggestions - Predicts your next edit as you type - Suggests multiple lines of code at once - Adapts to your coding patterns and project style - Works across all programming languages

Multi-Line Edits - Suggests entire code blocks - Handles complex refactoring scenarios - Maintains code consistency across your project

2. AI-Powered Error Detection

- Automatically identifies syntax errors and logic issues
- Provides explanations for detected problems
- Suggests multiple fix options
- Learns from your corrections

AI-Powered Chat Interface

The chat interface is your primary way to interact with Cursor's AI assistant.

How to Access

- **Keyboard Shortcut**: `Ctrl+L` (Windows/Linux) or `Cmd+L` (Mac)
- **Menu**: View → AI Chat
- **Sidebar**: Click the chat icon

Chat Modes

1. Agent Mode

- **Purpose**: Autonomous task completion with minimal supervision
- **Best for**: Complex, multi-step tasks across multiple files

- **Features**:
  - Learns your codebase automatically
  - Makes codebase-wide changes
  - Handles end-to-end task completion
  - Keeps you in the loop with progress updates

2. Ask Mode

- **Purpose**: Get explanations and answers about your code
- **Best for**: Understanding existing code, debugging, learning
- **Features**:
  - Explains complex code sections
  - Answers questions about your project
  - Provides documentation and examples

3. Manual Mode

- **Purpose**: Focused edits with precise control
- **Best for**: Specific changes where you want to control the context
- **Features**:
  - Uses only the context you provide
  - More predictable outputs
  - Better for sensitive code sections

Chat Commands and Symbols

@ Symbols for Context

- `@filename.js` - Reference a specific file
- `@folder/` - Reference an entire folder
- `@function_name` - Reference a specific function
- `@Codebase` - Search across your entire codebase
- `@Web` - Get information from the internet
- `@Docs` - Reference documentation

Special Commands

- `/fix` - Fix errors in selected code
- `/explain` - Explain selected code
- `/optimize` - Optimize selected code for performance
- `/test` - Generate tests for selected code
- `/doc` - Generate documentation

Agent Mode

Agent Mode is Cursor's most powerful feature, capable of autonomous task completion.

Key Capabilities

- **Autonomous Learning**: Automatically understands your codebase structure
- **Multi-File Operations**: Makes changes across multiple files simultaneously
- **Task Planning**: Breaks down complex requests into manageable steps
- **Progress Tracking**: Shows you what it's doing in real-time

Best Use Cases

- Implementing new features across multiple files
- Large-scale refactoring operations
- Setting up new project structures
- Integrating third-party libraries

How to Use Agent Mode

1. Open the chat interface (`Ctrl+L`)
2. Select "Agent" mode from the dropdown
3. Describe your task in natural language
4. Review the proposed plan
5. Approve or modify the approach
6. Monitor progress and provide feedback

Intelligent Code Completion

Tab Completion

- **Trigger**: Start typing and press `Tab` to accept suggestions
- **Multi-line**: Cursor can suggest entire code blocks
- **Context-aware**: Suggestions adapt to your current file and project

Ghost Text

- Appears as you type in a lighter color
- Shows predicted code completions
- Press `Tab` to accept, `Esc` to dismiss
- Updates in real-time based on context

Customization Options

- Adjust suggestion frequency in settings
- Configure which file types to enable/disable
- Set custom triggers for different languages

Smart Code Rewrites

Automatic Error Correction

Cursor can detect and fix common coding mistakes automatically:

- **Syntax Errors**: Missing semicolons, brackets, quotes
- **Type Errors**: Incorrect variable types, missing imports
- **Logic Errors**: Infinite loops, unreachable code
- **Style Issues**: Inconsistent formatting, naming conventions

How It Works

1. Type code naturally, even with mistakes
2. Cursor identifies issues in real-time
3. Hover over underlined errors to see suggestions
4. Click to apply fixes automatically
5. Learn from corrections to improve future suggestions

Refactoring Assistance

- **Extract Functions**: Automatically identify code that should be functions
- **Rename Variables**: Consistent renaming across files
- **Optimize Imports**: Remove unused imports, organize import statements
- **Code Splitting**: Break large files into smaller, manageable modules

---

Terminal Integration

AI-Powered Terminal Commands

Ctrl+K in Terminal

- Write commands in plain English
- Cursor converts to proper terminal syntax
- Supports complex multi-step operations
- Confirms commands before execution

Examples: - "install express and cors" → `npm install express cors` - "create a new git branch called feature-auth" → `git checkout -b feature-auth` - "find all javascript files modified today" → `find . -name "*.js" -mtime -1`

Built-in Terminal

- Integrated directly into the editor
- Share context with AI chat
- Command history and suggestions
- Support for all shell types (bash, zsh, powershell, etc.)

Command Confirmation

- By default, Cursor asks for confirmation before running commands

- Can be disabled in settings for trusted operations
- Shows command preview before execution
- Maintains command history for reference

Context Management

Codebase Indexing

Cursor automatically indexes your entire project to understand: - **File Structure**: How files relate to each other - **Dependencies**: Import/export relationships - **Patterns**: Common code patterns in your project - **Conventions**: Your coding style and naming conventions

@ Symbol References

Use @ symbols to provide specific context to the AI:

File References

- `@src/components/Button.tsx` - Reference a specific file
- `@src/utils/` - Reference all files in a directory
- `@package.json` - Reference configuration files

Code Symbol References

- `@MyComponent` - Reference a React component
- `@calculateTotal` - Reference a specific function
- `@UserInterface` - Reference a type or interface
- `@API_ENDPOINT` - Reference a constant

Special References

- `@Codebase` - Search across entire project
- `@Web` - Get information from the internet
- `@Docs` - Reference documentation
- `@Git` - Reference git history and changes

Context Window Management

- Cursor intelligently manages what context to include
- Prioritizes relevant files and functions
- Automatically excludes irrelevant information
- You can manually add/remove context as needed

Keyboard Shortcuts

Essential Shortcuts

Chat and AI

- `Ctrl+L` (Windows/Linux) or `Cmd+L` (Mac) - Open AI Chat
- `Ctrl+K` (Windows/Linux) or `Cmd+K` (Mac) - Inline AI editing
- `Ctrl+I` (Windows/Linux) or `Cmd+I` (Mac) - AI code generation
- `Ctrl+Enter` - Ask about codebase (equivalent to @Codebase)

Code Completion

- `Tab` - Accept AI suggestion
- `Esc` - Dismiss AI suggestion
- `Ctrl+Space` - Trigger manual completion
- `Alt+]` - Next suggestion
- `Alt+[` - Previous suggestion

Terminal

- `Ctrl+K` in terminal - Natural language command input
- ‘Ctrl+Shift+“ - Open new terminal
- ‘Ctrl+“ - Toggle terminal visibility

Navigation

- `Ctrl+P` - Quick file open
- `Ctrl+Shift+P` - Command palette
- `Ctrl+G` - Go to line
- `F12` - Go to definition
- `Alt+F12` - Peek definition

Editing

- `Ctrl+D` - Select word (repeat for multiple selections)
- `Ctrl+Shift+L` - Select all occurrences
- `Alt+Click` - Multiple cursors
- `Ctrl+/` - Toggle comment
- `Shift+Alt+F` - Format document

Custom Shortcuts

You can customize any keyboard shortcut in: - **Menu**: File → Preferences → Keyboard Shortcuts - **Command**: `Ctrl+K Ctrl+S`

Privacy & Security

Privacy Mode

Cursor offers comprehensive privacy protection:

Local Processing - Code never leaves your machine in Privacy Mode - AI processing happens locally when possible - No data stored on remote servers - Complete control over your intellectual property

How to Enable Privacy Mode 1. Open Settings (`Ctrl+,`) 2. Navigate to "Cursor" settings 3. Enable "Privacy Mode" 4. Restart Cursor for changes to take effect

Security Features

SOC 2 Certification

- Industry-standard security practices
- Regular security audits
- Compliance with enterprise security requirements
- Data encryption in transit and at rest

Enterprise Features

- Single Sign-On (SSO) support
- Team management and permissions
- Audit logs and activity tracking
- Custom deployment options

Data Handling

- **Code Analysis**: Minimal code snippets sent for AI processing
- **Context Filtering**: Sensitive information automatically filtered
- **Retention Policies**: Data retention limits and deletion policies
- **User Control**: Full control over what data is shared

Customization & Extensions

VS Code Compatibility

Since Cursor is built on VS Code, it supports: - **All VS Code Extensions**: Install from the marketplace - **Themes**: Visual and syntax themes - **Keybindings**: Import existing shortcuts - **Settings**: Migrate configurations seamlessly

Cursor-Specific Customization

AI Behavior Settings

- **Suggestion Frequency**: How often AI makes suggestions
- **Context Window Size**: How much code context to include
- **Model Selection**: Choose different AI models for different tasks
- **Language Preferences**: Optimize for specific programming languages

UI Customization

- **Chat Panel Position**: Left, right, or bottom
- **AI Indicators**: Show/hide AI suggestion indicators
- **Completion Style**: Inline vs popup suggestions
- **Theme Integration**: AI elements match your theme

Rules System

Create custom rules for consistent AI behavior:

Project-Specific Rules

```
- Always use TypeScript strict mode
- Follow React functional component patterns
- Use Tailwind CSS for styling
- Write comprehensive JSDoc comments
```

Language-Specific Rules

```
- Python: Follow PEP 8 style guide
- JavaScript: Use ES6+ features
- CSS: Use BEM methodology
- SQL: Use uppercase keywords
```

Advanced Features

Codebase Analysis

Cursor can analyze your entire codebase to provide insights:

Code Quality Metrics - Complexity analysis - Code duplication detection - Performance bottlenecks identification - Security vulnerability scanning

Architecture Understanding - Component dependency mapping - Data flow analysis - API usage patterns - Database schema relationships

Multi-Language Support

Cursor works with virtually any programming language:

Fully Supported Languages - JavaScript/TypeScript - Python - Java - C# - Go - Rust - PHP - Ruby - Swift - Kotlin

Framework-Specific Features - React/Vue/Angular component understanding - Django/Flask/Express.js patterns - Spring Boot/ASP.NET conventions - Mobile development (React Native, Flutter)

Integration Capabilities

Version Control

- **Git Integration**: Built-in git support with AI assistance
- **Commit Message Generation**: AI-generated commit messages
- **Code Review**: AI-powered code review suggestions
- **Branch Management**: Intelligent branch naming and merging

Database Integration

- **Query Generation**: Natural language to SQL conversion
- **Schema Understanding**: Database structure analysis

- **Migration Assistance**: Database migration generation
- **Performance Optimization**: Query optimization suggestions

API Integration

- **Documentation Generation**: Auto-generate API docs
- **Test Generation**: Create API tests automatically
- **Client Generation**: Generate API client code
- **Mock Data**: Create realistic test data

Best Practices

Effective AI Interaction

Writing Good Prompts

1. **Be Specific**: "Add error handling to the login function" vs "fix this"
2. **Provide Context**: Use @ symbols to reference relevant files
3. **Break Down Complex Tasks**: Split large requests into smaller parts
4. **Review Suggestions**: Always review AI-generated code before accepting

Context Management

- **Reference Relevant Files**: Use @filename to include specific files
- **Use Descriptive Names**: Clear function and variable names help AI understand
- **Maintain Clean Code**: Well-organized code gets better AI suggestions
- **Document Your Code**: Comments help AI understand your intentions

Project Organization

File Structure

```
src/
  components/       # Reusable UI components
  pages/           # Page components
  hooks/           # Custom React hooks
  utils/           # Utility functions
  types/           # TypeScript type definitions
  constants/       # Application constants
  api/             # API integration
  tests/           # Test files
```

Naming Conventions

- **Files**: Use kebab-case for files (`user-profile.tsx`)
- **Components**: Use PascalCase (`UserProfile`)
- **Functions**: Use camelCase (`getUserProfile`)
- **Constants**: Use UPPER_SNAKE_CASE (`API_BASE_URL`)

Code Quality

AI-Assisted Code Reviews

1. Use `/explain` to understand complex code sections
2. Ask for optimization suggestions with `/optimize`
3. Generate tests with `/test` command
4. Request documentation with `/doc` command

Continuous Improvement

- Regularly ask AI to review your code patterns
- Use AI to identify potential refactoring opportunities
- Generate comprehensive test suites
- Maintain consistent coding standards across the project

Troubleshooting

Common Issues

AI Not Responding **Symptoms**: Chat doesn't respond, no code suggestions **Solutions**: 1. Check internet connection 2. Restart Cursor 3. Clear cache: `Ctrl+Shift+P` → "Reload Window" 4. Update to latest version

Slow Performance **Symptoms**: Laggy typing, delayed suggestions **Solutions**: 1. Close unused files and tabs 2. Exclude large directories from indexing 3. Reduce context window size in settings 4. Disable extensions temporarily to identify conflicts

Incorrect Suggestions **Symptoms**: AI suggestions don't match your coding style **Solutions**: 1. Provide more context with @ symbols 2. Create custom rules for your project 3. Use more specific prompts 4. Train the AI by accepting/rejecting suggestions consistently

Privacy Mode Issues **Symptoms**: Limited functionality in privacy mode **Solutions**: 1. Some features require internet connectivity 2. Consider hybrid mode for better functionality 3. Check firewall settings 4. Verify privacy mode is properly configured

Getting Help

Official Resources

- **Documentation**: docs.cursor.com
- **Community Forum**: community.cursor.com
- **GitHub Issues**: Report bugs and feature requests
- **Discord Server**: Real-time community support

Debugging Tips

1. **Check Console**: `Ctrl+Shift+I` to open developer tools
2. **Log Files**: Find logs in Cursor's data directory

3. **Safe Mode**: Start Cursor with extensions disabled
4. **Reset Settings**: Restore default configuration if needed

---

Conclusion

Cursor AI represents a fundamental shift in how we approach software development. By seamlessly integrating artificial intelligence into every aspect of the coding workflow, it empowers developers to:

- **Code Faster**: AI-powered completions and suggestions
- **Code Smarter**: Context-aware assistance and error detection
- **Learn Continuously**: AI explanations and best practice suggestions
- **Focus on Logic**: Let AI handle boilerplate and repetitive tasks

Getting the Most Out of Cursor

1. **Start Small**: Begin with simple AI interactions and gradually use more advanced features
2. **Experiment**: Try different modes and approaches to find what works best for you
3. **Provide Feedback**: Help improve the AI by accepting/rejecting suggestions thoughtfully
4. **Stay Updated**: Cursor is rapidly evolving with new features and improvements

The Future of AI-Assisted Development

Cursor is at the forefront of AI-powered development tools. As the technology continues to evolve, we can expect: - Even more sophisticated code understanding - Better integration with development workflows - Enhanced collaboration features - Improved performance and reliability

Welcome to the future of coding with Cursor AI!

This guide was created to help you get started with Cursor AI. For the most up-to-date information, always refer to the official documentation at docs.cursor.com.