Cursor 101 Demo Flow

An introductory walkthrough of Cursor's core features for new users.

Prerequisites

Before starting the demo:

1. **Start the server:** Run /start-demo to launch the development server and open the blog app in the browser
2. **Reset workspace:** Use /reset between sections to ensure a clean state for deterministic results

> 📝 **Deep Links:** Throughout this demo, clickable links like `[Example Link]` open a "Create chat with prompt" dialog. Click **"Create Chat"** to start the agent with the pre-filled prompt.

---

1. Layout Walkthrough

> Cursor is an IDE (Integrated Development Environment) forked from VS Code, now fully built out with AI-native features.
>
> Understanding the layout helps you navigate efficiently and customize your workspace for different tasks.

Overview

Cursor extends VS Code with AI-powered panels and features. The main areas include:

- **File Explorer** (left sidebar) - Navigate your project files
- **Editor** (center) - View and edit code
- **Terminal** (bottom panel) - Run commands
- **Agent Panel** (right sidebar) - AI chat and agent interactions

Demo

Opening the Layout

When you first open Cursor, you'll see an empty editor. Use the layout controls to customize your view:

Flow

1. **File Explorer:** Click the folder icon (left) to see your project files
2. **Terminal:** Click the terminal area (bottom) to open the integrated terminal
3. **Agent Panel:** Click the chat icon (right) to open the AI agent panel
4. **Custom Layouts:** Save your preferred layout for quick switching

Saving Custom Layouts

1. Arrange your panels as desired
2. Click "Change Layout" in the view menu
3. Save as a custom layout
4. Switch between saved layouts as needed

What to Highlight

- Familiar VS Code experience with AI enhancements
- Customizable panel arrangement
- Quick layout switching for different workflows
- Integrated terminal for running commands alongside AI

Tips

- Use keyboard shortcuts to toggle panels quickly
- Save different layouts for coding vs. AI-assisted tasks
- The agent panel can be resized or detached

---

2. Tab + Quick Edit (Cmd+K)

> Cursor Tab provides context-aware, multi-line code suggestions as you type. It can modify multiple lines at once, add import statements when missing, and predict your next editing location within or across files.
>
> Inline Edit (Cmd+K) lets you edit code or generate new code directly within your editor using natural language. With code selected, it edits that specific code; without a selection, it generates new code at your cursor position.

Docs: Tab Overview | Inline Edit

How Tab Works

Cursor indexes your entire codebase to understand: - Your project's conventions and patterns - Existing components and utilities - Import structures and dependencies

Tab is a specialized Cursor model for autocompletion. The more you use it, the better it becomes as you inject intent by accepting (`Tab`) or rejecting (`Escape`) suggestions.

Tab Capabilities

Capability	Description
Multi-line edits	Modify multiple lines at once
Auto-imports	Add import statements when missing
Cross-file jumps	Jump within and across files for coordinated edits
Contextual awareness	Suggestions based on recent changes, linter errors, and accepted edits

Tab Autocomplete

Overview

Tab predicts what you want to write next and suggests complete code blocks. The more you use it, the more it learns your style—each `Tab` accept or `Escape` reject teaches the model your preferences.

Demo

Open an empty file (like `Footer.tsx`) and start typing:

1. Type the first character of a component name
2. Watch Tab suggest the entire component structure
3. Press `Tab` to accept suggestions
4. Continue iterating—each suggestion builds on your intent

Flow

1. Open `app/components/Footer.tsx`
2. Start typing (e.g., `f` or `function`)
3. Tab detects the file context and suggests a Footer component
4. Press `Tab` repeatedly to accept and build out the component
5. Type hints like "sign up for newsletter" to guide suggestions
6. Tab uses existing components from your codebase

What to Highlight

- **Codebase understanding:** Tab uses full codebase context to understand conventions

- **Contextual awareness:** Knows what file you're in, your recent changes, and linter errors
- **Learns your style:** The more you use it, the more it adapts to your preferences
- **Uses your existing patterns:** Suggestions incorporate your components and utilities
- **Feels like mind-reading:** Power users report Tab predicting exactly what they want

Quick Edit (Cmd+K)

Overview

Select code and press `Cmd+K` to make targeted edits using natural language. Only the selected section is modified.

Demo

1. Select a section of code
2. Press `Cmd+K`
3. Describe your change (e.g., "make this more colorful")
4. Review the diff
5. Click "Accept" to apply

What to Highlight

- Surgical edits—only changes what you select
- Natural language instructions
- Preview changes before accepting
- Great for refactoring specific sections

Best Practices

- **Tab:** Start with hints in comments or partial code to guide suggestions
- **Accept/Reject signals:** Each `Tab` accept or `Escape` reject teaches the model your preferences
- **Quick Edit:** Be specific about what you want changed
- **Combine them:** Use Tab for building, Quick Edit for refining

---

3. Agent Chat

> Agent Chat is the command center for agentic workflows within Cursor. An agent is a large language model (LLM) with access to tool calls—in this case, the ability to make changes directly to your codebase.

Docs: Agent Chat

Overview

This demo covers all the key configurations and features of Agent Chat.

3.1 Agent Modes

> Cursor provides different modes optimized for different tasks. Choose the right mode for your workflow.

Mode	Use Case
Agent	Execute code changes across your codebase (default)
Plan	Create editable implementation plans for complex features
Debug	Find, test, and resolve difficult bugs
Ask	Learn about the codebase without making changes (read-only)

3.2 Model Selection

> You have full control over which model to use. Choose based on your task complexity and speed requirements.

- **Composer 1**: Cursor's own coding model, optimized for speed
- **Auto**: Let Cursor smartly select the best model for each task
- **Frontier models**: Claude, GPT-4, Gemini, and more

Quick Selection

Click the model dropdown in the chat input to quickly switch between models.

Model Settings

For more control, configure models in **Cursor Settings > Models**:

1. Open **Cursor** menu > **Cursor Settings**
2. Navigate to **Models**
3. Click **View All Models** to see all available options
4. Enable or disable models based on your needs

This is useful for: - Enabling new models as they become available - Disabling models you don't use to declutter the selector - Setting default models for different workflows

3.3 Adding Context

> Provide the agent with relevant context so it can make informed decisions.

Why This Matters

Context is everything when working with AI. Without the right context, even the best model will give generic or incorrect answers. The friction of adding context—switching between apps, copying documentation from ChatGPT or Claude's web interface, pasting error messages—creates constant context switching that slows you down more than you realize.

Cursor makes adding context fast and seamless. Instead of juggling browser tabs and copy-pasting between applications, you simply type `@` and reference what you need. This eliminates the hidden productivity tax of context switching, letting you stay in flow while giving the agent exactly what it needs to help you.

How to Add Context

Use the **Add Context** button (or type `@`) to include:

- **Files** – Specific files to reference or modify
- **Folders** – Entire directories for broader context
- **Docs** – Up-to-date documentation from popular libraries
- **Terminal** – Terminal output for debugging errors
- **Past Chats** – Reference previous conversations

3.4 Image Upload

> Upload images as reference for the agent—mockups, screenshots, or error messages.

Click the image upload button to attach images to your prompt.

3.5 Voice Input

> Speak your prompts instead of typing for faster iteration.

Click the microphone button to record your prompt.

3.6 Execution Mode

> Choose where your agent runs based on your needs.

Mode	Description
Local	Runs on your machine with full access to local files
Cloud	Runs on Cursor's cloud infrastructure
Worktree	Run parallel agents in isolated worktrees (advanced)

3.7 Starting the Server

> Let the agent handle common tasks like starting your development server.

- Start the development server

The agent will inspect your project configuration and run the appropriate command.

Note: Configure terminal command permissions in **Cursor Settings > Agents**: - Ask every time (default) - Run in sandbox - Run everything

3.8 Making Code Changes

> Reference specific files when asking for changes to help the agent find the right code immediately.

- Change sidebar background to blue

After the agent makes changes: 1. Click **Review Changes** to see a diff 2. Click **Keep** to accept or **Undo** to revert

3.9 Fixing Bugs from Terminal Output

> Quickly fix bugs by piping terminal errors directly into the agent.

Step 1: Introduce a bug (for demo purposes) - Use /introduce-runtime-bug slash command

Step 2: View the error 1. Open the terminal panel (View > Terminal) 2. Select the terminal with the error output

Step 3: Send to agent 1. Select the error text in the terminal 2. Press `Cmd+L` to add to chat 3. Submit to have the agent fix it

3.10 Using Documentation

> Reference up-to-date library documentation so the agent uses current APIs and patterns.

Built-in Docs

Popular libraries are pre-indexed. Type `@docs` and search:

- Add validation with Zod

Adding Custom Docs

1. Click **Add Context** > **Docs**
2. Scroll to bottom, click **Add new doc**
3. Paste the documentation URL
4. Cursor will index it for future use

3.11 Git Diff Context

> Review changes against your main branch and generate commit messages.

Use `@Git (Diff with Main Branch)` to: - Understand what your feature branch changed - Generate commit messages based on actual changes - Get an overview of your current work status

- Explain changes against main

3.12 Context Window Management

> Monitor your agent's context usage to ensure optimal performance.

As conversations grow, the context window fills up. When it reaches capacity: 1. Cursor summarizes the conversation 2. Resets with the summary as context 3. Continues the task seamlessly

Best Practice: Use atomic tasks—one focused task per agent session. Start new agents for new tasks to keep context clean.

3.13 Demo: Implementing Dark Mode

> Walk through implementing a complete feature using Agent Chat.

This demo shows the full workflow: clicking a deep link, selecting a model, running the agent, and reviewing changes.

Steps

1. Click the deep link below to open the prompt
2. Click **Create Chat** in the dialog
3. Select **composer-1** for faster results
4. Submit and watch the agent implement the feature
5. Click **Review** to see all changes
6. Accept or undo changes as needed

- Implement Dark Mode

What to Highlight

- Deep links pre-fill prompts for consistent demos
- Model selection affects speed and quality
- Agent reads existing code to understand patterns
- Review mode shows all changes across files
- Changes can be accepted or reverted individually

Quick Reference

Action	Shortcut/Command
Open Agent Chat	`Cmd+L`
Quick Edit	`Cmd+K`
Accept Tab suggestion	`Tab`
Reject Tab suggestion	`Escape`
Chat History	`Opt+Cmd+'`
New Chat	Click “New Chat” button
Add Selection to Chat	Select text + `Cmd+L`
Voice Input	Click microphone button
Start demo server	/start-demo
Reset workspace	/reset

Next Steps

After completing Cursor 101, explore:

- **Cursor 2.0 Demo** – Advanced features like Browser, Worktrees, Bugbot
- **Plan Mode** – Detailed implementation planning
- **Debug Mode** – Hypothesis-driven debugging
- **Browser** – Built-in browser testing