Cursor – Die komplette Anleitung

Stand: Juli 2026

Autor: Christian Drapatz

> *Diese Anleitung stützt sich auf offizielle Cursor-Dokumentation, Blogposts und Changelogs sowie, wo keine offizielle Quelle auffindbar war, auf Community-Berichte und Presseartikel. Eine Übersicht der am schwächsten belegten Einzelpunkte steht am Ende unter „Hinweis zur Recherchequalität" (Kapitel 19).*

Inhaltsverzeichnis

(Rechtsklick > Felder aktualisieren, um das Inhaltsverzeichnis zu aktualisieren.)

1. Was ist Cursor?

Cursor ist ein KI-gestützter Code-Editor der Firma Anysphere, technisch ein Fork der quelloffenen Codebasis von Visual Studio Code. Cursor beschreibt sich selbst als Coding-Agenten für den Bau ambitionierter Software, der sowohl autonom im Hintergrund als auch parallel an mehreren Aufgaben arbeiten kann. Anders als ein reiner Chat-Assistent ist Cursor direkt in den Editor integriert: Es liest und bearbeitet Dateien, führt Terminalbefehle aus, arbeitet mit Git, durchsucht eine ganze Codebasis semantisch und läuft, wie VS Code selbst, auf macOS, Windows und Linux.

Die enge Verwandtschaft zu VS Code gilt als allgemein bekannt und ist mehrfach unabhängig bestätigt, auch wenn eine wörtliche Selbstaussage „wir sind ein VS-Code-Fork" auf einer offiziellen Cursor-Seite bei der Recherche zu dieser Anleitung nicht auffindbar war. Wer VS Code kennt, findet sich in Cursor sofort zurecht: gleiche Tastenkürzel-Logik, gleiches Ordner-/Workspace-Modell, weitgehend kompatible Erweiterungen (siehe Kapitel 3).

Homepage: https://cursor.com/

2. Wem gehört Cursor?

Cursor wird von Anysphere Inc. entwickelt, gegründet 2022 von Michael Truell, Sualeh Asif, Arvid Lunnemark und Aman Sanger, die sich während ihres Studiums am MIT kennenlernten. Firmensitz ist San Francisco.

Anysphere ist zum Stand dieser Anleitung noch ein eigenständiges Unternehmen, allerdings mit einer weitreichenden Ankündigung im Rücken: Am 16. Juni 2026 gab SpaceX bekannt, Anysphere in einem reinen Aktiendeal im Wert von 60 Milliarden US-Dollar zu übernehmen; Anysphere würde dabei zur hundertprozentigen SpaceX-Tochter. Der Abschluss wurde für das dritte Quartal 2026 erwartet, vorbehaltlich behördlicher Genehmigung, und ist zum Stand dieser Anleitung noch nicht vollzogen. Vorausgegangen war eine Option, die sich SpaceX bereits am 21. April 2026 gesichert hatte. Die Meldung wurde unabhängig von mehreren Presseorganen (u. a. CNBC, Forbes, Yahoo Finance) mit übereinstimmenden Eckdaten bestätigt, auch wenn keine eigene Anysphere-Pressemitteilung dazu auffindbar war. Strategischer Hintergrund laut diesen Berichten: SpaceX war vier Handelstage zuvor, am 12. Juni 2026, an die Nasdaq gegangen und wollte mit dem Deal Zugriff auf Cursors Coding-Daten für das Training seiner xAI/Grok-Modelle erhalten, im Gegenzug bekäme Cursor Zugriff auf Rechenkapazität aus dem „Colossus"-Rechenzentrum von SpaceX/xAI. Dass Cursor bereits ein eigenes Modell zusammen mit „SpaceXAI" trainiert (siehe Kapitel 5), passt zu dieser Darstellung.

Bis zu dieser Ankündigung hatte Anysphere mehrere Finanzierungsrunden durchlaufen: eine Seed-Runde im Oktober 2023 (8 Mio. USD, angeführt vom OpenAI Startup Fund), eine Series A im August 2024 (60+ Mio. USD, Bewertung 400 Mio. USD), eine Series B im Dezember 2024 (100 Mio. USD, Bewertung rund 2,5 Mrd. USD), eine Series C im Juni 2025 (900 Mio. USD, Bewertung 9,9 Mrd. USD, ARR damals über 500 Mio. USD) sowie eine Series D am 13. November 2025 (2,3 Mrd. USD, Bewertung 29,3 Mrd. USD, angeführt von Accel und Coatue, mit ARR über 1 Mrd. USD). Presseberichten zufolge lag der annualisierte Umsatz zum Zeitpunkt des SpaceX-Deals bei rund 4 Mrd. USD; eine offizielle Unternehmenszahl dazu gibt es nicht, einzelne Quellen weichen leicht voneinander ab. Eine belastbare Mitarbeiterzahl war ebenfalls nicht auffindbar, Schätzungen Dritter schwanken stark.

3. Architektur: Wie hängen Cursor und VS Code zusammen?

Cursor übernimmt die VS-Code-Codebasis und damit deren Bedienkonzept, Ordner-/Workspace-Modell und Erweiterungssystem. Seit Juni 2025 bezieht Cursor Erweiterungen nicht mehr über den offiziellen Microsoft-Marketplace (Microsoft beschränkt dessen Nutzung inzwischen auf eigene Produkte), sondern über die anbieterneutrale Open-VSX-Registry, vermittelt über einen eigenen Marketplace-Proxy mit automatisierter Malware-/Supply-Chain-Prüfung. Für Nutzer bedeutet das: Nicht jede VS-Code-Erweiterung ist zwangsläufig auch über Cursors Marketplace verfügbar, die meisten verbreiteten Erweiterungen aber schon.

Die KI-Funktionen sind in mehrere Bausteine gegliedert:

- **Tab** – die Autovervollständigung. Sie schlägt Code passend zu kürzlich vorgenommenen Änderungen, umgebendem Code und Linter-Fehlern vor, auch mehrzeilig und dateiübergreifend, ergänzt fehlende Imports und springt nach Annahme eines Vorschlags per erneutem Tab-Druck automatisch zur nächsten relevanten Codestelle.
- **Ask** – ein reiner Lesemodus zum Verstehen einer Codebasis: Fragen beantworten und Code erkunden, ohne etwas zu verändern.
- **Agent** – der Standardmodus für eigenständiges Arbeiten: Dateien lesen und bearbeiten, Terminalbefehle ausführen, das Web durchsuchen, Fehler selbständig beheben. Laut Dokumentation gibt es keine feste Obergrenze für Werkzeugaufrufe innerhalb einer Aufgabe.
- **Manual** – laut mehreren Community-Quellen ein Modus, der jede einzelne Änderung einzeln bestätigen lässt; eine eigene, ausführliche offizielle Dokumentationsseite dafür wurde nicht gefunden, die Existenz des Modus gilt aber als gesichert.
- **Plan-Modus** – der Agent recherchiert zunächst die Codebasis, stellt bei Bedarf Rückfragen und erzeugt daraus einen editierbaren, als Markdown-Datei gespeicherten Plan mit Datei-Referenzen, Todo-Punkten und ggf. Mermaid-Diagrammen, bevor überhaupt Code geschrieben wird. Aktivierung laut Dokumentation über Shift+Tab oder den Mode-Picker; bei als komplex erkannten Aufgaben schlägt Cursor den Modus auch von sich aus vor.
- **Debug-Modus** – der Agent formuliert zunächst mehrere Hypothesen zur Fehlerursache, instrumentiert die App mit Laufzeit-Logging, bittet aktiv um Reproduktion des Bugs, wertet die gesammelten Logs aus, nimmt einen gezielten Fix vor und entfernt die Instrumentierung danach wieder. Gedacht für schwer reproduzierbare Bugs, Race Conditions, Performance-/Speicherprobleme und Regressionen.
- **Multitask** – kein eigener Antwortmodus wie Plan/Debug/Ask, sondern ein Orchestrierungs-Befehl: zerlegt eine größere Aufgabe in unabhängige Teilschritte und lässt asynchrone Subagenten diese parallel statt sequenziell abarbeiten, abhängige Schritte bleiben dabei in korrekter Reihenfolge. Erreichbar über den Slash-Befehl `/multitask` oder den Button „Build in Parallel" bei einem bestehenden Plan.
- **Bild-Generierung („Image")** – kein Modus im engeren Sinn, sondern ein Agent-Werkzeug: Aus einer Textbeschreibung oder einem Referenzbild lässt sich ein Bild erzeugen (laut Changelog über das Modell „Google Nano Banana Pro"), das Ergebnis erscheint als Inline-Vorschau im Chat und wird standardmäßig im `assets/`-Ordner des Projekts abgelegt – nützlich für UI-Mockups, Produkt-Assets oder Architektur-Visualisierungen.

Alle fünf zusätzlichen Bausteine sind seit der Oberflächen-Überarbeitung mit Version 2.x/3.x über ein gemeinsames „+"-Menü im Composer-/Chat-Eingabefeld erreichbar, direkt neben der Modell-Auswahl – zusammen mit Skills (7.5) und MCP-Servern (7.3), die sich darüber offenbar ebenfalls gezielt für eine einzelne Konversation aus- oder anpinnen lassen. Plan-Modus, Debug-Modus, Multitask und Ask sind dabei jeweils mit einer eigenen offiziellen Dokumentationsseite belegt; dass alle diese Einträge exakt so in einem einzigen „+"-Menü zusammengeführt sind, ist dagegen eine naheliegende, aber nicht durch eine einzelne Quelle mit Screenshot bestätigte Schlussfolgerung aus mehreren Einzel-Changelogs – für die exakte Menüstruktur bleibt ein eigener Blick in die aktuelle Live-App die verlässlichste Quelle.

„Composer" bezeichnet seit Cursor 2.0 (Ankündigung 29. Oktober 2025) zwei unterschiedliche, aber verwandte Dinge: zum einen die Oberfläche für paralleles Arbeiten mit mehreren Agents gleichzeitig, zum anderen den Namen von Anysphere-eigenen Coding-Modellen (siehe Kapitel 5). Composer wurde nicht in „Agent" umbenannt, beide Begriffe existieren nebeneinander.

Für asynchrones Arbeiten gibt es **Cloud Agents** (früher „Background Agents" genannt): eigenständige, isolierte Remote-Agenten, die in virtuellen Maschinen laufen statt lokal, ein Repository (GitHub, GitLab, Azure DevOps Services oder Bitbucket Cloud) klonen, auf einem eigenen Branch arbeiten und ihre Änderungen zurückpushen. Mehrere Cloud Agents können parallel laufen, ohne dass die eigene Maschine dafür online sein muss; sie können bauen, testen, per Computer-Use auch Desktop/Browser steuern und liefern laut Dokumentation „merge-ready PRs" inklusive Demo-Artefakten. Seit einem Changelog-Eintrag vom 29. Juni 2026 (Version 3.9) lassen sich Cloud Agents zusätzlich über eine iOS-App sowie „Remote Control" vom Smartphone aus steuern, inklusive Live Activities und Push-Benachrichtigungen.

Bugbot ist Anyspheres eigenes automatisiertes PR-Review-Tool: Es analysiert Pull-Request-Diffs, hinterlässt Kommentare mit Erklärungen und Fix-Vorschlägen („Fix in Cursor"/„Fix in Web"), läuft automatisch bei jedem PR-Update oder auf Zuruf per Kommentar („cursor review"/„bugbot run"). Projektspezifische Review-Vorgaben lassen sich in einer Datei `.cursor/BUGBOT.md` hinterlegen. Laut einem Blogpost vom Juni 2026 ist die aktuelle Version über dreimal schneller, 22 % günstiger und findet 10 % mehr Fehler als die vorherige.

Der Feature-Stand entwickelt sich derzeit schnell weiter. Laut Changelog Stand Juli 2026: Version 3.0 (2. April 2026) brachte eine neu gestaltete Oberfläche mit parallelen Agents über mehrere Repos/Umgebungen hinweg, ein „Agents Window", einen „Design Mode" und „Agent Tabs". Version 3.9 (22. Juni 2026) fügte eine zentrale „Customize Cursor"-Seite für Plugins, Skills, MCPs, Subagents und Rules hinzu. Version 3.10 (30. Juni 2026) erweiterte Team-MCP-Server auf Cloud Agents, das Agents-Fenster, die IDE und die CLI. Version 3.11 (10. Juli 2026) brachte „Side Chats" für parallele Agent-Konversationen. Aktuellste, zum Zeitpunkt dieser Anleitung als „Latest" markierte Version ist 3.12.

4. Installation

4.1 Systemanforderungen

- **macOS:** mindestens macOS 12 (Monterey), native Unterstützung für Apple Silicon und Intel. Die Downloadseite bietet drei Varianten an: Mac (ARM64), Mac (x64), Mac Universal. Konkrete RAM- oder Speicherplatz-Mindestanforderungen werden auf der offiziellen Seite nicht genannt.
- **Windows:** mindestens Windows 10, natives `.exe`, als System- oder Nutzer-Installation, zusätzlich eine ARM64-Variante.
- **Linux:** drei offizielle Wege – ein apt-Repository (empfohlen für Debian/Ubuntu, inklusive GPG-Key-Einrichtung), ein dnf/yum-Repository für RHEL/Fedora, sowie ein portables AppImage. Die Paket-Repositories werden gegenüber dem AppImage empfohlen, da sie Desktop-Icons, automatische Updates und CLI-Tools gleich mitbringen.

4.2 Schritt-für-Schritt-Installation

1. Downloadseite öffnen: https://cursor.com/downloads
2. Passenden Installer für das eigene Betriebssystem und die eigene Architektur wählen (auf dem Mac z. B. „Mac Universal", sofern nicht bekannt ist, ob das Gerät Apple Silicon oder Intel nutzt)
3. Installer wie eine gewöhnliche Anwendung installieren (macOS: `.dmg` öffnen und in den Programme-Ordner ziehen)
4. Beim ersten Start mit GitHub, Google oder Microsoft anmelden
5. Wer bereits VS Code nutzt: In den Cursor-Einstellungen (⌘/Strg Shift J) unter **General > Account** den Punkt **„VS Code Import"** verwenden, um Erweiterungen, Themes, Einstellungen und Tastenkürzel zu übernehmen. Alternativ funktioniert das auch manuell über VS-Code-Profile (Export/Import, auch über einen GitHub Gist)

Ein ausführlicher, zusammenhängender offizieller Text zum genauen Onboarding-Ablauf (Gatekeeper-Warnung unter macOS, erster Einrichtungsassistent) war zum Zeitpunkt der Recherche nicht auffindbar; der VS-Code-Import-Schritt dagegen ist offiziell dokumentiert.

4.3 Aktualisieren

Cursor bietet drei offiziell dokumentierte Update-Kanäle, umschaltbar über ⌘/Strg Shift J im Beta-Bereich der Einstellungen:

Kanal	Inhalt
Default	Stabile Releases mit aus Pre-Release-Tests bereinigten Bugfixes; für Team-/Enterprise-Accounts verpflichtend
Early Access	Features in Entwicklung, potenziell mit Bugs; nicht für Team-Accounts verfügbar
Nightly	Experimentelle Bleeding-Edge-Builds mit häufigen Updates, potenziell mit Breaking Changes; nicht für Team-Accounts verfügbar

Ob automatische Updates standardmäßig aktiv sind, wird auf der offiziellen Seite nicht ausdrücklich gesagt. Community-Berichte beschreiben automatische Updates übereinstimmend als Standardverhalten, das sich nur über die interne Einstellung `update.mode` in der `settings.json` abschalten lässt – dieser Punkt ist allerdings nur sekundär belegt, nicht offiziell dokumentiert.

5. Ein KI-Modell auswählen

Anders als LM Studio ist Cursor kein primär auf lokale Modelle ausgelegtes Werkzeug: Der Standardweg führt über Cloud-Modelle mehrerer Anbieter sowie Anyspheres eigene Modelle. Lokale Modelle sind möglich, aber nur über einen Umweg (siehe 5.3).

5.1 Modellübersicht nach Anbieter

Laut offizieller Modell-/Preisseite (Stand Juli 2026) stehen unter anderem folgende Modelle zur Auswahl:

- **Anthropic:** Claude 4 Sonnet, Claude 4 Sonnet 1M, Claude 4.5 Haiku, Claude 4.5 Opus, Claude 4.5 Sonnet, Claude 4.6 Opus, Claude 4.6 Sonnet, Claude 4.7 Opus, Claude Fable 5, Claude Opus 4.7 (Fast Mode, laut Doku „limited research preview"), Claude Opus 4.8, Claude Sonnet 5 (mit Einführungspreis bis 31.08.2026)
- **OpenAI:** GPT-5, GPT-5 Fast, GPT-5 Mini, GPT-5-Codex sowie mehrere Nachfolgeversionen bis GPT-5.6 Luna/Sol/Terra
- **Google:** Gemini 2.5 Flash bis Gemini 3.5 Flash/3.1 Pro
- **xAI:** Grok 4.5 – laut Changelog vom 8. Juli 2026 gemeinsam von Cursor und „SpaceXAI" trainiert, ein Mixture-of-Experts-Modell für langlaufende Software-, Daten-, Finanz-, Rechts- und allgemeine Computer-Use-Aufgaben, verfügbar auf Desktop, Web, iOS, CLI und SDK, zu 2 USD/Mio. Input- und 6 USD/Mio. Output-Token
- **Weitere Drittanbieter im Modellpool:** GLM 5.2 (Z.ai), Kimi K2.7 Code (Moonshot)

Zusätzlich bietet Anysphere eigene Modelle unter dem Namen **Composer** an: Composer (vorgestellt mit Cursor 2.0), laut offiziellem Blogpost „bis zu 4x schneller als ähnlich leistungsfähige Modelle" und für die meisten Anfragen unter 30 Sekunden ausgelegt; Composer 2.5 (Mai 2026), laut Anysphere zuverlässiger bei komplexen, langlaufenden Aufgaben und günstiger als die „Fast"-Stufen anderer Frontier-Modelle. Composer 2.5 basiert weiterhin auf dem offenen Kimi-K2.5-Checkpoint von Moonshot AI, trainiert mit 25-mal mehr synthetischen Aufgaben als Composer 2 (Preise: 0,50 USD/Mio. Input, 2,50 USD/Mio. Output; Fast-Variante 3,00/15,00 USD). Laut demselben Blogpost trainiert Cursor zusammen mit „SpaceXAI" aktuell ein deutlich größeres Nachfolgemodell mit dem zehnfachen Rechenaufwand, unter Nutzung der Rechenkapazität von „Colossus 2".

Ein Modell namens „Cheetah" tauchte laut Community-Berichten im Oktober 2025 kommentarlos im Modell-Menü auf; eine offizielle Cursor-Quelle dazu wurde nicht gefunden, Herkunft und aktuelle Verfügbarkeit bleiben unklar.

5.2 Auto, Max Mode und mehrere Modelle parallel

Auto ist kein trainiertes Modell, sondern ein Auswahlmodus: Cursor wählt pro Anfrage automatisch ein Modell, das laut Dokumentation Intelligenz, Kosten und Zuverlässigkeit ausbalanciert, zu einem festen Token-Preis unabhängig vom intern tatsächlich gewählten Modell. Der Modell-Auswahldialog sitzt oben im Chat-/Agent-Panel; über ⌘/Strg / lässt sich durch die Modelle durchschalten, die Auswahl bleibt über eine Konversation hinweg gespeichert. Neben „Auto" gibt es das Preset „Premium", das automatisch die leistungsfähigsten verfügbaren Modelle wählt.

Seit Cursor 2.0 lassen sich über die „Agents"-Oberfläche auch mehrere Modelle **parallel** an derselben Aufgabe arbeiten lassen, isoliert über Git-Worktrees oder Remote-Maschinen; laut Anysphere verbessert das Ergebnis spürbar, „especially for harder tasks". Details wie die genaue Obergrenze paralleler Agents sind nur über Drittquellen belegt.

Max Mode erweitert laut Dokumentation das Kontextfenster eines Modells über dessen Standardgrenze hinaus, ist aber ausdrücklich nur auf älteren, anfragebasierten („legacy") Tarifen verfügbar; Modelle, die es zwingend benötigen, aktivieren es automatisch. Abgerechnet wird zum API-Satz des Modells plus 20 %. Für aktuelle Tarife gibt es keinen offiziell so benannten Nachfolger; erweiterter Kontext ist zunehmend direkt in einzelne Modelle eingepreist (z. B. „Claude 4.5 Sonnet – bis zu 1 Mio. Token mit erweitertem Kontext zum gleichen Satz").

5.3 Lokale Modelle und eigene API-Keys

Eine eigene, offizielle Cursor-Dokumentationsseite zur Anbindung lokaler Modelle (etwa über Ollama oder LM Studio) wurde nicht gefunden. Mehrere übereinstimmende Community-Quellen beschreiben einen Mechanismus namens **„Override Base URL"** unter **Settings > Models > OpenAI API**, über den eine OpenAI-kompatible lokale Endpunkt-URL eingetragen werden kann (Ollama typischerweise `http://localhost:11434/v1`, LM Studio `http://localhost:1234/v1`); das API-Key-Feld darf dabei laut diesen Quellen nicht leer bleiben, akzeptiert aber einen Platzhaltertext. Offiziell dokumentiert ist dagegen das allgemeine Routing-Prinzip: Anfragen laufen „durch Cursors Server für den finalen Prompt-Aufbau" – dieses Prinzip gilt vermutlich auch für Custom-Endpunkte. Mehrere Drittquellen berichten zusätzlich, dass eine öffentlich erreichbare URL nötig sei, reines `localhost` also nicht ausreiche, etwa über Tunneling-Dienste. Wer lokale Modelle wirklich lokal und ohne Umweg über Cursors Server ausführen möchte, ist mit einem reinen Terminal-Agenten (siehe Kapitel 18) unter Umständen besser bedient.

Custom API Keys sind dagegen offiziell dokumentiert und unterstützen die Anbieter OpenAI, Anthropic, Google, Azure OpenAI und AWS Bedrock. Ein eigener Schlüssel wird nicht dauerhaft auf Cursor-Servern gespeichert, aber bei jeder Anfrage ans Cursor-Backend mitgesendet, da der Prompt-Aufbau final dort stattfindet. Für die eigentlichen Daten gilt dann die Datenschutzrichtlinie des jeweiligen Anbieters statt Cursors eigener Zero-Data-Retention-Zusage (siehe Kapitel 15). Nutzung mit eigenem Key zählt laut übereinstimmenden Quellen nicht gegen das Cursor-Abo-Kontingent und wird direkt zum Standardtarif des Anbieters abgerechnet. Wichtige Einschränkung: Tab-Autovervollständigung, „Apply from Chat" und Composer benötigen zwingend Cursors eigene Modelle und funktionieren unabhängig von hinterlegten Custom Keys nicht über diese.

Für Anthropic/Claude gilt dabei eine Besonderheit, die insbesondere beim Umstieg von Claude Code relevant wird: Ein claude.ai-Consumer-Abo (Claude Pro/Max) lässt sich nicht als Custom API Key verwenden, dafür ist zwingend ein separater, nutzungsbasiert abgerechneter Key aus der Anthropic Console nötig. Details dazu, mit Quellenbelegen, in Kapitel 14.1.

5.4 Erste Testfrage stellen (Funktionstest)

6. Einen Ordner in Cursor öffnen (siehe Kapitel 6)
7. Im Chat-/Agent-Panel prüfen, dass ein Modell ausgewählt ist (Standard ist meist „Auto")
8. In den Agent-Modus wechseln, sofern nicht bereits aktiv

9. Eine einfache Testfrage stellen, z. B. „Erkläre kurz, was dieses Projekt macht" oder in einem leeren Ordner „Erstelle eine Datei `hello.py`, die 'Hallo Welt' ausgibt"
10. Mit Enter abschicken und die Antwort bzw. den vorgeschlagenen Diff prüfen

Kommt innerhalb weniger Sekunden eine sinnvolle Antwort oder ein plausibler Diff zurück, ist die Einrichtung erfolgreich. Bleibt die Antwort aus, liegt es meist an einer fehlenden Anmeldung, einem aufgebrauchten Kontingent (siehe Kapitel 9) oder – bei Custom-API-Key-Nutzung – an einem ungültigen Schlüssel.

6. Cursor einrichten: der erste Workspace

Anders als Agenten mit eigener Projektverwaltung (etwa LM Studio Bionic, siehe dortiges Tutorial) übernimmt Cursor unverändert das VS-Code-Workspace-Modell: Ein geöffneter Ordner ist ein Workspace, eine eigenständige „Projects"-Verwaltung als zusätzliche Cursor-spezifische Ebene wurde in der offiziellen Dokumentation nicht gefunden.

11. Cursor öffnen, über **File > Open Folder** (bzw. **Open Recent**) ein bestehendes Projektverzeichnis öffnen
12. Bei Bedarf über **File > Add Folder to Workspace** einen weiteren Ordner ergänzen und als **Multi-Root-Workspace** in einer `.code-workspace`-Datei speichern – innerhalb des Chats lassen sich einzelne Repos dann z. B. über `@repo-name/pfad` referenzieren. Dieser Mechanismus ist nur über Community-Quellen belegt, nicht über eine eigene offizielle Cursor-Doku-Seite
13. Modell im Chat-/Agent-Panel wählen (siehe Kapitel 5)
14. Aufgabe im Agent-Modus formulieren – Cursor liest dafür bei Bedarf selbständig relevante Dateien im geöffneten Ordner

Ein Changelog-Eintrag vom 10. Juli 2026 (Version 3.11, „Side Chats and Conversation Search") erwähnt „neu gestaltete Projekt- und Repository-Auswahldialoge" – das bezieht sich laut Kontext auf die Repository-Auswahl für Cloud Agents (Kapitel 8) sowie auf die im Folgenden beschriebene, überarbeitete Sidebar-Navigation und den Repo-Picker beim Start einer neuen Agent-Session, nicht auf eine eigenständige lokale Projektverwaltung.

Sidebar-Navigation. Links in der Cursor-Oberfläche liegen laut diesem Changelog fünf zentrale Einstiegspunkte, deren einzelne Funktionen offiziell dokumentiert sind, auch wenn die genaue gemeinsame Anordnung als Sidebar-Layout in keiner Quelle mit Screenshot belegt ist:

- **New Agent** – startet eine neue Agent-Konversation/-Session im „Agents Window" (lokal, Cloud, Worktree oder Remote-SSH), offiziell primär über ⌘/Strg I bzw. ⌘/Strg Shift P → „Agents Window" erreichbar.
- **Search** – zwei Ebenen: die Codebase-/Kontextsuche des Agents (Instant Grep, semantische Suche, „Explore"-Subagent) sowie, seit Version 3.11 neu, eine Konversations-/Transkript-Suche über ⌘/Strg K im Agents Window, die frühere Agent-Chats über einen lokalen Suchindex durchsucht (laut Dokumentation skalierend auf tausende Konversationen); innerhalb eines laufenden Chats zusätzlich ⌘/Strg F mit Treffer-Zähler.
- **Automations** – lässt Cloud Agents automatisch im Hintergrund laufen, zeitgesteuert (Cron/vordefinierte Intervalle) oder ausgelöst durch GitHub, GitLab, Bitbucket, Slack, Linear, Sentry, PagerDuty oder eigene Webhooks. Einrichtung in fünf Schritten: Trigger wählen, Instructions-Prompt schreiben, optionale Tools konfigurieren, Repo-Scope festlegen (kein Repo/ein Repo/Multi-Repo), speichern/aktivieren; der `/automate`-Skill erlaubt die Konfiguration auch in natürlicher Sprache. Jede Automation läuft in einer frischen Cloud-Sandbox, geklont vom Repo-Stand zum Trigger-Zeitpunkt, Änderungen werden zur Review gestellt statt direkt angewendet. Typische Nutzung: automatisiertes Code-Review, autonome Bugfixes/Tests, Incident-Untersuchung, Slack-Triage. Aus Sicherheitsgründen laufen PR-Trigger nicht auf Fork-PRs (außer Merge-Trigger). Seit Version 3.11 auch direkt im Agents Window verwaltbar, nicht mehr nur über die separate Seite cursor.com/automations.
- **Customize** – die zentrale, seit dem 22. Juni 2026 (Version 3.9) eingeführte Verwaltungsseite für Plugins (gebündelte Rules, Skills, Subagents, Commands, MCP-Server, Hooks), einzelne Rules, Skills, Subagents, Commands und MCP-Server-Verbindungen, mit Scope-Filter nach

Nutzer/Workspace/Team, einem Leaderboard meistgenutzter Setups sowie Installation per Klick über den Marketplace (siehe auch Kapitel 7.1, 7.2, 7.5).

- **Repositories** – am schwächsten als eigener Dokumentationsartikel belegt, Existenz und Funktion aber über mehrere Quellen konsistent bestätigt: eine Gruppierungs-Ansicht im Agents-Window-Sidebar, die die verbundenen/aktiven Repositories auflistet, über die Agents laufen (lokal, Worktree, Cloud). Repos werden entweder lokal geöffnet oder über Settings → Background Agent → „Connect GitHub" angebunden (GitHub App mit Lese-/Schreibrechten), GitLab und Azure DevOps lassen sich ebenfalls direkt im Picker verbinden.

Neue Agent-Session: Ausführungsort wählen. Beim Start einer neuen Aufgabe bietet der überarbeitete Repo-/Projekt-Picker laut demselben Changelog eine vereinfachte Dreiteilung:

- **No Repo** – die Session läuft ohne geklontes Repository/ohne Code, gedacht für Workflows, die nur Slack, MCP, Webhooks, Linear oder PagerDuty benötigen (frühere „Home"-Option, jetzt als eigene explizite Wahl); auch über Eingabe von „none" oder „no repo" auffindbar.
- **On This Computer** (auf dem Mac vermutlich als „On This Mac" beschriftet) – lokale Ausführung des Agents auf dem eigenen Rechner statt in der Cloud-Sandbox, Teil eines neuen „Run on"-Pickers, der zusätzlich zwischen Cloud, This Computer und Remote Machines wählen lässt.
- **Cloud** – Ausführung in Cursors isolierter Cloud-VM (Cloud Agent, Kapitel 3/8): jeder Agent erhält eine eigene, dedizierte Linux-VM mit vollständigem Terminal, Browser und Desktop-Umgebung.

Innerhalb des lokalen Zweigs lassen sich vermutlich ein bestehender Ordner/ein bestehendes Repository auswählen oder ein neuer, leerer Ordner anlegen – die dafür beschriebenen Beschriftungen „Use Existing" und „New Folder" ließen sich allerdings in keiner offiziellen Quelle, keinem Changelog-Text und keinem geprüften Forumsbeitrag mit exakt diesem Wortlaut verifizieren; das ist an dieser Stelle also eine plausible, aber unbestätigte Einordnung, keine belegte Tatsache. Für die exakte aktuelle Beschriftung bleibt ein Blick in die eigene Live-App die verlässlichste Quelle.

7. Konfiguration

7.1 Die Settings-Oberfläche im Überblick

Die Cursor-Einstellungen öffnen sich über ⌘/Strg Shift J. Cursor pflegt zwei getrennte offizielle Doku-Bäume – cursor.com/docs (technisch, Agent/Konfiguration) und cursor.com/help (Account/Billing/Customization) –, eine einzelne, vollständige offizielle Gliederungstabelle über alle Settings-Kategorien hinweg wurde nicht gefunden. Die folgende Tabelle fasst die mit Stand Juli 2026 (Version 3.11/3.12) sichtbaren Kategorien zusammen; wo ein Thema bereits in einem eigenen Unterkapitel behandelt wird, verweist die Tabelle dorthin, statt es zu duplizieren.

Kategorie	Inhalt	Beleglage
General	U. a. ein Telemetry-Schalter (Settings → Telemetry → Off); weitere Optionen vermutlich VS-Code-Erbe	Keine eigene offizielle Referenzseite mit vollständiger Optionsliste gefunden, nur community-basiert
Profile	Öffentliches Profil unter `cursor.com/@handle`: einmalig festlegbarer Handle (min. 3 Zeichen, danach nicht mehr änderbar), Name/Avatar aus dem Account, Public-Profile-Toggle, bis zu 4 externe Links (X, GitHub, LinkedIn etc.); Team-Admins können die Sichtbarkeit zusätzlich einschränken	Offiziell dokumentiert
Appearance	Farbthema (Dark+/Light+/Monokai/Solarized etc.), Light/Dark-Umschaltung, Schriftgröße	Offiziell, aber nur rudimentär dokumentiert
Plan & Usage	Included Usage (im Abo enthalten) vs. On-Demand Usage (Überschreitung, nachträglich abgerechnet, deaktivierbar oder mit Spend-Limit deckelbar); bei Teams zusätzlich ein Admin-Dashboard mit aggregierten Nutzungsmetriken (siehe auch Kapitel 9)	Offiziell dokumentiert, über mehrere Help-Seiten verteilt statt einer einzelnen Unterseite
Agents	Kernstück Auto-Run/„YOLO"-Modus: automatisches Ausführen von Terminalbefehlen ohne Einzelbestätigung, gesteuert über eine Command-Allowlist unter Settings → Agents → Auto-Run; siehe auch die granularen Run Modes in Kapitel 7.7	Changelog-gestützt, Detailtext teils community-basiert
Cloud Agents	Admin-Dashboard mit Default-Modell/-Repo/-Base-Branch, Environment-Verwaltung (Snapshots, Setup-Skripte, Secrets), Netzwerk-Policy, Team-Controls für Long-Running-Agents und Computer Use (nur Enterprise)	Offiziell dokumentiert
Models	Modell-Aktivierung für Chat/⌘K, inklusive Auto-Select (siehe Kapitel 5)	Offiziell dokumentiert

Kategorie	Inhalt	Beleglage
Git & PRs	PR-Inbox, gruppiert Pull Requests nach Status; Default-Repositories bestimmen den Inhalt der Inbox (Free: bis 3 Repos, Team/Enterprise: bis 30), eigene Sections mit Filtern/Repo-Auswahl konfigurierbar	Offiziell dokumentiert
Worktree	Lässt Agents in isolierten Git-Checkouts arbeiten (eigene Dateien/Abhängigkeiten je Aufgabe, Hauptcheckout bleibt unberührt); Konfiguration über `.cursor/worktrees.json` (`setup-worktree`, `setup-worktree-unix`, `setup-worktree-windows`), maschinenweite Settings `cursor.worktreeCleanupIntervalHours` (Default alle 6 h) und `cursor.worktreeMaxCount` (Default 25 pro Maschine); Bedienung über `/worktree` (ein isolierter Task) und `/best-of-n` (gleiche Aufgabe parallel über mehrere Modelle/Worktrees)	Offiziell dokumentiert
Plugin	Plugins bündeln MCP-Server, Skills, Subagents, Rules und Hooks zu einem installierbaren Paket, verwaltet über „Customize" (User-, Team- oder Workspace-Ebene), Bezug über cursor.com/marketplace mit manuellem Review jedes Plugins; für Team-Marketplaces eigene „Marketplace Settings" (Zugriff, optionales Auto Refresh)	Offiziell dokumentiert
Rules, Skills, Subagents	Rules = projektweite Anweisungen unter `.cursor/rules/` (Kapitel 7.2); Skills = bei Bedarf abrufbares Spezialwissen via `SKILL.md` (Kapitel 7.5); Subagents = parallele Spezial-Agents mit eigenem Kontext/eigenen Werkzeugen	Offiziell dokumentiert
Tools & MCPs	„Enable MCP Servers"-Schalter, jeder Server aufklappbar mit einzeln an-/ausschaltbaren Tools, UI zum Hinzufügen neuer MCP-Server als Alternative zum manuellen Editieren der `mcp.json` (Kapitel 7.3)	Offiziell dokumentiert
Hooks	Stdio-Prozesse, die vor/nach Phasen des Agent-Loops laufen (z. B. `beforeShellExecution`, `afterFileEdit`, `stop`) und Verhalten beobachten, blockieren oder verändern können (Kapitel 7.6)	Offiziell dokumentiert
Browser & Network	Lokaler Agent-Browser mit drei Freigabemodi (manuelle Bestätigung je Aktion, Allowlist mit Auto-Ausführung, volles Auto-Run), Allow-/Blocklisten unter Settings → Agents → Auto-Run, Enterprise-Origin-Allowlist für automatisch navigierbare Domains; separat Proxy-Konfiguration über Umgebungsvariablen oder `settings.json`; für Cloud Agents zusätzlich drei eigene Netzwerkmodi	Offiziell dokumentiert
Tab	„Snooze" (temporäres Deaktivieren für eine gewählte Dauer), globales Deaktivieren, Deaktivieren für bestimmte Dateiendungen (z. B. Markdown, JSON), frei belegbares Tastenkürzel für „Accept Cursor Tab Suggestion"	Offiziell dokumentiert
Indexing	Codebase wird standardmäßig vollständig indiziert und automatisch mit Änderungen synchronisiert; unter „Show Settings" zusätzlich automatische Indizierung neuer Repos an/aus sowie zusätzliche Ignore-Regeln über `.cursorignore` (ergänzt `.gitignore`, siehe auch Kapitel 15)	Offiziell dokumentiert

Kategorie	Inhalt	Beleglage
Beta	Experimentelle Features (früher z. B. „Agent Window", „Cursor Browser") erscheinen hier laut Community-Berichten zuerst, bevor sie stabil in andere Kategorien wandern; siehe auch die Update-Kanäle in Kapitel 4.3	Schwach belegt – die entsprechende Doku-Seite leitet inzwischen nur noch auf die Doku-Startseite um, keine vollständige aktuelle Optionsliste auffindbar
Docs (`@docs`-Feature)	Eigene Dokumentationsquellen per URL crawlen/indizieren lassen („Add new doc"), eine URL mit abschließendem Slash indiziert auch Unterseiten, verwaltete Docs sind bearbeitbar/löschbar, ein „Share with team"-Toggle macht eine Quelle teamweit verfügbar, Nutzung im Prompt über `@`	Offiziell dokumentiert, Detailtext community-nah zusammengefasst

Am stärksten offiziell abgesichert sind demnach Worktree, Plugin, Hooks, Cloud Agents, MCP/Tools & MCPs, Indexing, Profile, Git & PRs sowie Browser & Network – jeweils mit eigener Doku-Seite. Am schwächsten belegt sind General und Beta (keine aktuelle vollständige offizielle Referenzseite) sowie Appearance (nur rudimentär dokumentiert); Detailoptionen können hier vom tatsächlichen Stand der eigenen App-Version abweichen.

7.2 Rules (.cursor/rules)

Project Rules liegen als `.mdc`-Dateien unter `.cursor/rules` im Projektstamm; eine reine `.md`-Datei an dieser Stelle wird ignoriert. Das Frontmatter jeder Regel kennt genau drei Felder:

```
---
description: Kurzbeschreibung, wann diese Regel gilt
globs: "*.swift"
alwaysApply: false
---

Regeltext in natürlicher Sprache.
```

Aus der Kombination dieser drei Felder ergeben sich vier Ladearten, die auch so in der Cursor-Oberfläche heißen:

Ladeart	Bedingung
Always	`alwaysApply: true` – Regel wird immer geladen, `globs`/`description` werden dabei ignoriert
Auto Attached	`globs` gesetzt und eine passende Datei ist gerade im Kontext

Ladeart	Bedingung
Agent Requested	nur `description` gesetzt – das Modell entscheidet selbst, ob die Regel relevant ist, und fordert sie bei Bedarf an
Manual	weder `description` noch `globs` gesetzt – Regel wird nur eingebunden, wenn sie im Chat explizit per `@`-Erwähnung referenziert wird

Die ältere, einzelne `.cursorrules`-Datei im Projektstamm gilt laut einem Forumsbeitrag eines als Anysphere-nah eingeordneten Mitglieds (10. März 2026) offiziell als Legacy-Format und soll perspektivisch abgekündigt werden, funktioniert vorerst aber weiter; bei Überschneidungen haben `.mdc`-Regeln Vorrang. Die aktuelle offizielle Rules-Seite erwähnt `.cursorrules` nicht mehr.

> **Wichtige Klarstellung zum Geltungsbereich:** `.cursor/rules` unterstützt **kein** automatisches, ordnerbasiertes Scoping über mehrere verschachtelte `.cursor/rules`-Verzeichnisse hinweg (also kein `backend/.cursor/rules/` mit eigenem, automatisch nur dort geltendem Regelsatz). Unterordner innerhalb von `.cursor/rules` dienen laut offizieller Dokumentation ausdrücklich nur der Ablage/Organisation: „Cursor discovers rules by scanning this folder, so nested subfolders work but a flat structure is simpler and easier to manage." Der tatsächliche Geltungsbereich einer Regel läuft unabhängig vom physischen Speicherort ausschließlich über das `globs`-Feld im Frontmatter. Über GitHub importierte Remote-Regeln landen zusätzlich automatisch unter `.cursor/rules/imported/<repoName>/`.

Zusätzlich zu Project Rules gibt es **User Rules**: global, projektübergreifend, konfiguriert unter **Customize > Rules** bzw. den Cursor-Einstellungen. Wichtige Einschränkung laut Dokumentation: User Rules gelten ausdrücklich nur im Agent-Chat, nicht für Inline Edit (⌘/Strg K). User Rules haben außerdem **keinen** eigenen Dateipfad im Dateisystem (kein Pendant zu `~/.cursor/rules/`), sie liegen ausschließlich in den lokalen Cursor-Einstellungen und sind laut Dokumentation explizit **nicht** Teil von Profil-Exports – beim Wechsel auf eine andere Maschine müssen sie manuell neu eingegeben oder in eine Projekt-Regel überführt werden.

`AGENTS.md` und `CLAUDE.md`: das eigentliche Pendant zu Claude Codes verschachtelten Regeldateien. Wer aus Claude Code das Modell kennt, in dem `CLAUDE.md` sowohl im Projektstamm als auch in beliebigen Unterverzeichnissen liegen und automatisch geladen werden kann, findet dieses Verhalten bei Cursor nicht bei `.cursor/rules`, sondern bei `AGENTS.md`: „Cursor supports AGENTS.md in the project root and subdirectories … You can place AGENTS.md files in any subdirectory of your project, and they will be automatically applied when working with files in that directory or its children. Instructions from nested AGENTS.md files are combined with parent directories, with more specific instructions taking precedence." Das ist strukturell exakt Claude Codes Vererbungsmodell für Unterverzeichnisse, nur mit `AGENTS.md` statt `CLAUDE.md` als Dateiname.

Bemerkenswerter Zusatzpunkt: Cursor liest darüber hinaus **`CLAUDE.md`-Dateien selbst nativ**, ohne Umbenennung. Laut offizieller Dokumentation: „Cursor reads CLAUDE.md files the same way it reads AGENTS.md. Place a CLAUDE.md file in your project root and Cursor picks it up automatically. CLAUDE.md files are always applied to every conversation, regardless of any alwaysApply frontmatter setting. This ensures compatibility with projects that also use Claude Code." Da diese Datei „genauso wie AGENTS.md" behandelt wird, ist naheliegend, dass auch verschachtelte `CLAUDE.md`-Dateien in Unterverzeichnissen

entsprechend automatisch geladen werden; eine gesonderte, explizit auf verschachtelte `CLAUDE.md`-Dateien bezogene Einzelaussage wurde in der Recherche zu dieser Anleitung aber nicht gefunden, das ist also eine naheliegende Schlussfolgerung aus der „genauso wie AGENTS.md"-Formulierung, keine wörtlich zitierbare Einzelaussage.

Ein echtes, mehrstufiges Vererbungsmodell mit automatischem Scoping nach Verzeichnistiefe existiert bei Cursor außerdem – unabhängig von Rules – bei zwei weiteren, in diesem Tutorial behandelten Bausteinen: bei `BUGBOT.md` (Kapitel 3, mit einer offiziell dokumentierten Reihenfolge „Team Rules → repository rules → project BUGBOT.md (including nested files) → User Rules") und bei Agent Skills, die ebenfalls verschachtelte `.cursor/skills/`- bzw. `.agents/skills/`-Ordner in Unterverzeichnissen unterstützen und dabei automatisch auf Dateien in diesem Unterverzeichnis begrenzt werden (siehe 7.5).

7.3 MCP-Server

MCP-Server werden über eine JSON-Datei mit dem Objekt `mcpServers` konfiguriert, global unter `~/.cursor/mcp.json`, projektbezogen unter `.cursor/mcp.json`. Stdio-Server nutzen die Felder `command`, `args`, `env`; Remote-Server die Felder `url` und `headers`. Zusätzlich gibt es eine teamweite Ebene über ein Web-Dashboard mit eigenem Team-Marketplace.

Für „Add to Cursor"-Buttons existiert ein offiziell dokumentiertes Deeplink-Schema:

```
cursor://anysphere.cursor-deeplink/mcp/install?name=$NAME&config=$BASE64_ENCODED_CONFIG
```

Die Konfiguration wird dafür als JSON serialisiert und base64-kodiert. OAuth wird für Remote-Server unterstützt, inklusive statischer Client-Registrierung über ein `auth`-Objekt mit `CLIENT_ID`, `CLIENT_SECRET` und `scopes` im jeweiligen Servereintrag, abgesichert über zwei feste Redirect-URLs, die beim jeweiligen OAuth-Anbieter freigeschaltet werden müssen. Die genaue Priorität bei Namenskonflikten zwischen projektbezogener und globaler Konfiguration ist nicht mit einer offiziellen Primärquelle belegt.

MCP-Server können auf lokale Dateien, Netzwerkdienste oder vertrauliche Daten zugreifen. Nur vertrauenswürdige Server installieren und deren Berechtigungen sorgfältig prüfen.

7.4 Checkpoints und Sandbox

Checkpoints sind ein von Git unabhängiges, rein lokales Snapshot-System: Der Agent legt vor größeren Änderungen automatisch einen Checkpoint an, der den Zustand aller geänderten Dateien erfasst. Über einen Klick auf einen Checkpoint im Chat-Verlauf bzw. den „Restore Checkpoint"-Button lässt sich der komplette Dateizustand zurücksetzen. Nur vom Agenten vorgenommene Änderungen werden erfasst, manuelle Bearbeitungen nicht – Git bleibt deshalb für echte Versionskontrolle weiterhin nötig; Checkpoints sind als temporäre, sitzungsbezogene Rollback-Punkte gedacht.

Die **Sandbox** betrifft Terminalbefehle des Agenten: Ein sandboxed Agent agiert frei innerhalb einer kontrollierten Umgebung und fragt nur um Erlaubnis, wenn er diese verlassen muss, meist für Internetzugriff. Laut einem Blogpost vom Februar 2026 senkt das die Zahl der Freigabe-Unterbrechungen

um rund 40 % gegenüber ungesandboxten Agents. Die technische Umsetzung ist plattformabhängig: macOS nutzt Seatbelt/`sandbox-exec`, Linux eine Kombination aus Landlock (Dateisystem-Restriktion) und seccomp (Syscall-Filterung, inklusive eines Overlay-Dateisystems für über `.cursorignore` ausgeschlossene Dateien), Windows den Linux-Sandbox-Mechanismus innerhalb von WSL2.

7.5 Agent Skills

Mit Version 2.4 (22. Januar 2026) führte Cursor **Agent Skills** ein: laut Dokumentation ein „portables, versionskontrolliertes Paket, das Agenten beibringt, domänenspezifische Aufgaben auszuführen". Ein Skill ist strukturell ein Ordner mit einer `SKILL.md`-Datei und YAML-Frontmatter (Pflichtfelder `name` und `description`, optional `paths`, `disable-model-invocation`, `metadata`), ergänzt um optionale Unterordner `scripts/`, `references/` und `assets/` – strukturell sehr nah an Claude-Code-Skills.

Ablageorte sind projektbezogen `.agents/skills` oder `.cursor/skills`, global `~/.agents/skills` oder `~/.cursor/skills`, mit ausdrücklich genannter Kompatibilität zu älteren Konventionen wie `.claude/skills` und `.codex/skills`. Cursor bezeichnet Agent Skills als offenen Standard mit Verweis auf agentskills.io, macht aber keine ausdrückliche Aussage, dass es sich exakt um dasselbe Format wie Anthropics Claude-Code-Skills handelt; die identische Ordnerkonvention legt technische Kompatibilität nahe, ohne dass eine offizielle Adoption desselben Formats bestätigt wäre.

Anders als Project Rules unterstützen Agent Skills laut Dokumentation ausdrücklich **echtes, automatisches Verzeichnis-Scoping**: „Cursor also discovers skills inside nested project subdirectories. A .cursor/skills/ (or .agents/skills/) folder anywhere inside your repository is picked up, so monorepos can colocate skills with the package they apply to" (Beispiel aus der Dokumentation: `apps/web/.cursor/skills/deploy-web/SKILL.md`). Skills in solchen verschachtelten Ordnern gelten laut Dokumentation automatisch nur für Dateien innerhalb dieses Unterverzeichnisses – anders als bei `.cursor/rules`, wo verschachtelte Ordner rein organisatorisch sind (siehe 7.2).

Ein eingebauter Befehl `/migrate-to-skills` kann bestehende Rules und Commands automatisch in Skills umwandeln.

7.6 Hooks

Seit Version 1.7 (29. September 2025, zunächst als Beta markiert) verfügt Cursor über ein **Hook-System** zum Beobachten, Steuern und Erweitern des Agent-Loops per eigene Skripte. Die Dokumentation listet unter anderem folgende Events: `sessionStart`, `sessionEnd`, `preToolUse`, `postToolUse`, `postToolUseFailure`, `subagentStart`, `subagentStop`, `beforeShellExecution`, `afterShellExecution`, `beforeMCPExecution`, `afterMCPExecution`, `beforeReadFile`, `afterFileEdit`, `beforeSubmitPrompt`, `preCompact`, `stop`, `afterAgentResponse`, `afterAgentThought`, dazu Tab-spezifische Hooks (`beforeTabFileRead`, `afterTabFileEdit`) sowie den App-Lifecycle-Hook `workspaceOpen`.

Konfiguriert wird über `hooks.json`, projektbezogen unter `.cursor/hooks.json`, global unter `~/.cursor/hooks.json` (dazu eigene Enterprise-Pfade für Linux, macOS und Windows). Das Schema besteht aus einem `version`-Feld und einem `hooks`-Objekt mit Kommandodefinitionen, die unter

anderem `command`, `type`, `timeout`, `loop_limit`, `failClosed` und `matcher` unterstützen. Ob der ursprüngliche Beta-Status inzwischen offiziell aufgehoben wurde, ließ sich nicht eindeutig klären: Die aktuelle Doku-Seite zeigt kein Beta-Label mehr, ein explizites „Graduation"-Announcement wurde aber nicht gefunden.

7.7 Permission-Modell (Run Modes, permissions.json)

Der frühere, informelle Begriff „YOLO mode" wurde durch **„Auto-Run"** ersetzt; ein exaktes offizielles Umbenennungsdatum ließ sich nicht ermitteln, die aktuelle Dokumentation nutzt konsistent den Begriff **„Run Modes"**. Konfiguriert wird unter **Settings > Agents > Approvals and Execution**. Es gibt drei Run Modes:

- **Auto-review** – nicht gelistete Shell-Befehle laufen sandboxed; die Dokumentation betont ausdrücklich, dass Auto-review keine Sicherheitsgrenze darstellt
- **Allowlist** – gelistete Aktionen laufen ohne Rückfrage
- **Run Everything** – vollautomatische Ausführung ohne Rückfrage

Unabhängig vom gewählten Modus gelten laut Dokumentation immer drei feste Schutzmechanismen: File-Deletion Protection, External-File Protection und Browser Protection.

Für feingranulare, dauerhaft gespeicherte Regeln gibt es eine eigene Datei **`permissions.json`**, an zwei Orten möglich – `~/.cursor/permissions.json` (nutzerweit) und `.cursor/permissions.json` im Workspace (projektspezifisch). Sind beide Dateien vorhanden, werden ihre Listen zusammengeführt; die Priorität ist Team-Admin-Dashboard vor `permissions.json` vor der IDE-Settings-Oberfläche. Wichtige Felder: `terminalAllowlist` (Präfix-Matching für Shell-Befehle), `mcpAllowlist` (Format `server:tool`, mit Wildcard-Unterstützung) sowie ein `autoRun`-Objekt mit frei formulierten `allow_instructions`/`block_instructions`, die den Auto-review-Klassifizierer lenken, laut Dokumentation aber keine harte Garantie erzwingen. Das ist ein direktes, offiziell dokumentiertes Äquivalent zum `permissions`-Block in Claude Codes `.claude/settings.json` (siehe Kapitel 13 und 14.6).

7.8 Verzeichnisstruktur im Überblick

Alle in diesem Kapitel behandelten Bausteine liegen projektbezogen gemeinsam unter einem einzigen `.cursor/`-Verzeichnis im Projektstamm, mit der Ausnahme von `AGENTS.md`/`CLAUDE.md` (die außerhalb von `.cursor/` liegen) und zwei zusätzlichen, an dieser Stelle noch nicht erwähnten Bausteinen: projektbezogenen Subagents unter `.cursor/agents/*.md` sowie einer optionalen `.cursor/environment.json` (plus optionalem `.cursor/Dockerfile`) zur Konfiguration der Cloud-Agent-Umgebung (Kapitel 3 und 8). Zusammengefasst, mit `#`-Kommentaren zur Fundstelle in diesem Tutorial:

```
mein-projekt/
├── AGENTS.md                    # Projektregeln, projektweit (Kap. 7.2)
├── CLAUDE.md                    # wird von Cursor nativ genauso wie AGENTS.md gelesen (Kap. 7.2)
├── backend/
│   ├── AGENTS.md                # gilt nur für backend/ und tiefer, überschreibt Root bei Konflikt
│   └── .cursor/
│       ├── BUGBOT.md            # gilt nur für Reviews von backend-Dateien (Kap. 3)
```

```
│       └── skills/
│           └── deploy-backend/
│               └── SKILL.md        # automatisch auf backend/ begrenzt (Kap. 7.5)
└── .cursor/
    ├── rules/
    │   ├── *.mdc                   # Project Rules, Geltungsbereich über globs (Kap. 7.2)
    │   └── imported/<repoName>/    # per GitHub importierte Remote-Regeln
    ├── skills/<name>/SKILL.md      # Agent Skills, alternativ .agents/skills/ (Kap. 7.5)
    ├── commands/<name>.md          # veraltete Custom Commands, faktisch von Skills abgelöst (Kap. 13)
    ├── agents/<name>.md            # projektbezogene Subagents
    ├── hooks.json                  # Hooks-Konfiguration (Kap. 7.6)
    ├── hooks/format.sh             # zugehörige Hook-Skripte
    ├── mcp.json                    # MCP-Server, projektbezogen (Kap. 7.3)
    ├── permissions.json            # Permission-Regeln, projektbezogen (Kap. 7.7)
    ├── BUGBOT.md                   # projektweite Bugbot-Vorgaben (Kap. 3)
    └── environment.json (+Dockerfile)  # Cloud-Agent-Umgebung (Kap. 8)
```

Global, nutzerweit unter dem Home-Verzeichnis:

```
~/.cursor/
├── mcp.json           # MCP-Server, für alle Projekte (Kap. 7.3)
├── hooks.json          # Hooks, für alle Projekte (Kap. 7.6)
├── hooks/
├── permissions.json    # Permission-Regeln, für alle Projekte (Kap. 7.7)
├── agents/             # globale Subagents
└── cli-config.json     # Konfiguration der Cursor CLI (Kap. 14.8)
```

> **User Rules bilden die einzige Ausnahme:** Sie haben keinen Dateipfad, weder lokal noch global – sie liegen ausschließlich in den lokalen Cursor-Einstellungen (**Customize > Rules**) und sind laut Dokumentation ausdrücklich nicht Teil eines Profil-Exports.
>
> **Kernunterschied zu Claude Codes `CLAUDE.md`-Modell:** Cursors eigenes Rules-Format (`.cursor/rules/*.mdc`) unterstützt kein automatisches Vererben über verschachtelte Verzeichnisse – dafür aber `AGENTS.md`, und zusätzlich liest Cursor `CLAUDE.md`-Dateien nativ „genauso wie AGENTS.md". Wer eine bestehende, auch über Unterverzeichnisse verteilte Claude-Code-Struktur mit mehreren `CLAUDE.md`-Dateien hat, kann diese Dateien nach aktuellem Dokumentationsstand unverändert im Projekt liegen lassen; Cursor liest sie automatisch mit, ganz ohne Umbenennung in `AGENTS.md` oder Umzug nach `.cursor/rules` (siehe auch Kapitel 14.2).

8. Lokal oder Cloud – Agent-Modi und Cloud Agents

Cursor unterscheidet weniger zwischen „lokal" und „Cloud" als LM Studio Bionic, da die Standard-Modelle ohnehin Cloud-Modelle sind (siehe Kapitel 5). Die relevantere Unterscheidung ist, **wo** der Agent läuft:

- **Lokal, im Editor:** Agent-, Ask- und Manual-Modus laufen direkt in der geöffneten Cursor-Instanz, mit sofortigem Zugriff auf den geöffneten Workspace, Terminal und (je nach Run Mode) automatischer Ausführung von Befehlen
- **Cloud Agents:** asynchrone, isolierte Remote-Agenten in eigenen virtuellen Maschinen (Kapitel 3), die unabhängig von der eigenen Maschine laufen, mehrere Aufgaben parallel bearbeiten und fertige Pull Requests liefern
- **Bugbot:** ein spezialisierter, ebenfalls in der Cloud laufender Agent ausschließlich für automatisiertes PR-Review (Kapitel 3)
- **Eigene, lokal ausgeführte Modelle** sind nur über den in Kapitel 5.3 beschriebenen, nicht vollständig offiziell dokumentierten Umweg möglich und laufen dabei technisch trotzdem über Cursors Server für den Prompt-Aufbau

Wer wirklich komplett lokal und offline arbeiten will, ist mit Cursor also grundsätzlich schlechter bedient als mit einem auf lokale Modelle spezialisierten Werkzeug wie LM Studio Bionic.

9. Preise

Stand Juli 2026, laut offizieller Preisseite und ergänzenden Blogposts:

Stufe	Preis	Enthalten
Hobby	0 USD	Kostenlos, keine Kreditkarte nötig; begrenzte Agent-Anfragen und Tab-Vervollständigungen
Pro	20 USD/Monat	20 USD API-Agent-Guthaben, großzügige Nutzung des First-Party-Modellpools (Auto, Composer 2.5, Grok 4.5), Zugriff auf Frontier-Modelle, MCPs/Skills/Hooks, Cloud Agents, Bugbot nutzungsbasiert
Pro+	60 USD/Monat	70 USD API-Agent-Guthaben
Ultra	200 USD/Monat	400 USD API-Agent-Guthaben
Teams Standard	40 USD/Nutzer/Monat (monatlich), 32 USD (jährlich)	Team-Funktionen, gepooltes Guthaben
Teams Premium	120 USD/Nutzer/Monat (monatlich), 96 USD (jährlich)	„5x die Nutzung von Standard"; Preise gelten für Abrechnungszyklen ab 1. Juli 2026
Enterprise	individuell	Pooled Usage, Invoice/PO-Abrechnung, SCIM-Seat-Management, Repository-/Modell-/MCP-Zugriffskontrollen, Auto-Run-/Browser-/Netzwerk-Kontrollen, Audit-Logs, Service Accounts, AI-Code-Tracking-API, priorisierter Support

Exakte Jahrespreise für Pro/Pro+/Ultra werden auf der Preisseite nicht als konkrete Zahl ausgewiesen.

Ist das inkludierte Guthaben aufgebraucht, erscheint eine Benachrichtigung; Nutzer können nutzungsbasierte Zusatzabrechnung (Pay-as-you-go) aktivieren oder einen höheren Tarif wählen. Die Abrechnung erfolgt zu öffentlichen Modell-API-Preisen, im Nachhinein. Bei Nutzung eigener API-Keys (Kapitel 5.3) entfällt die Abrechnung über Cursors Kontingent vollständig, dafür gelten Cursors Zero-Data-Retention-Zusagen dann nicht mehr, sondern die Datenschutzbestimmungen des jeweiligen Modellanbieters.

Bugbot wurde im Juni 2026 von einem festen 40-USD-Sitzplatzmodell auf reine Nutzungsabrechnung umgestellt; die durchschnittlichen Kosten liegen laut Anysphere bei 1,00–1,50 USD pro Review.

> **Upgrade-Hinweis im Hobby-Plan.** Mehrere übereinstimmende Community-Berichte beschreiben, dass Cursor Hobby-Nutzer nach Erreichen des monatlichen Kontingents über eine Meldung im Chat-/Agent-Bereich („you need to sign up for a pro account" bzw. „You've hit your usage limit") auf ein Upgrade hinweist und die Weiternutzung bis zum Upgrade oder bis zum nächsten Plan-Reset blockiert. Eine offiziell dokumentierte, dauerhaft feste UI-Position dieses Hinweises (etwa unten in der

> Sidebar oder im Settings-Panel) ließ sich dagegen nicht verifizieren – der Hinweis scheint kontextuell beim Erreichen des Limits zu erscheinen, nicht als permanenter Banner.

10. Möglichkeiten: Was kann man mit Cursor tun?

Im Editor (Agent-Modus):

- Bestehenden Code lesen, erklären und gezielt ändern, dateiübergreifend
- Autovervollständigung über Tab, inklusive mehrzeiliger und dateiübergreifender Vorschläge
- Terminalbefehle ausführen (Build, Tests, Linting), gesteuert über die Run Modes (Kapitel 7.7)
- Mit Git arbeiten, Diffs vor der Übernahme prüfen
- Auf projektspezifische Rules, MCP-Server und Agent Skills zugreifen

Cloud Agents:

- Aufgaben asynchron in isolierten VMs bearbeiten, während die eigene Maschine offline sein kann
- Mehrere Aufgaben parallel bearbeiten, auch über mehrere Repositories hinweg
- Fertige, „merge-ready" Pull Requests inklusive Demo-Artefakten liefern
- Über die iOS-App bzw. Remote Control auch vom Smartphone aus steuern

Bugbot:

- Automatisiertes Review jedes Pull Requests mit Erklärungen und Fix-Vorschlägen

Plattformübergreifend:

- Wahlfreiheit zwischen zahlreichen Cloud-Modellen unterschiedlicher Anbieter sowie Anyspheres eigenen Composer-Modellen
- Weitgehende Kompatibilität zu VS-Code-Erweiterungen über die Open-VSX-Registry

11. Arbeiten mit Xcode (Beispiel)

Cursor hat kein Xcode-Plugin und ist kein Ersatz für Xcode als Build-/Signing-/Simulator-Werkzeug, sondern ein eigenständiger Editor. Für Swift-Entwicklung gibt es dafür sowohl eine offizielle Anleitung auf swift.org („Setting up Cursor for Swift Development") als auch eine eigene Cursor-Dokumentationsseite („iOS and macOS Swift"):

15. In Cursor den Ordner des Swift-Projekts öffnen (bei einem SwiftPM-Projekt der Ordner mit der `Package.swift`)
16. Die offiziell vom swiftlang-Projekt gepflegte Swift-Erweiterung installieren (Repository `swiftlang/vscode-swift`), die auf `sourcekit-lsp` als Sprachserver aufbaut; für Debugging wird LLDB verwendet. Die Erweiterung ist über die Open-VSX-Registry verfügbar, laut offiziellem Swift-Blog ausdrücklich für agentische IDEs wie Cursor gedacht
17. Vor Swift 6.1 einmal `swift build` im Terminal ausführen, damit der Sprachindex gefüllt wird – ohne diesen Schritt bleiben Codevervollständigung und Fehleranzeige unvollständig
18. Aufgabe im Agent-Modus formulieren, z. B.: „Füge der Struct `Calculator` eine Methode `subtract` hinzu, die zwei `Int`-Werte subtrahiert, und schreibe dazu einen passenden Test mit Swift Testing"
19. Cursor durchsucht das Projekt, schlägt Änderungen als Diff vor und kann `swift test` über die eingebaute Terminal-Integration ausführen
20. Diff und Testergebnis prüfen, Änderung übernehmen, bei Bedarf über einen Checkpoint zurückspringen (Kapitel 7.4)

> **Wichtige Einschränkung:** Die offizielle swift.org-Anleitung unterstützt in erster Linie SwiftPM-Projekte mit `Package.swift`; klassische `.xcodeproj`-Projekte mit Targets, Schemes und Simulatoren werden dort nicht als direkt unterstützt beschrieben. Für vollständige klassische Xcode-Projekte empfehlen mehrere unabhängige, nicht offizielle Blogs die Community-Erweiterung **SweetPad**, die `xcodebuild` im Hintergrund kapselt. Eine offizielle Lösung von Apple oder Cursor dafür wurde nicht gefunden. Wer an einer bestehenden iOS-/macOS-App mit Storyboards, Asset-Katalogen oder Interface Builder arbeitet, bleibt für diese Teile weiterhin auf Xcode selbst angewiesen und nutzt Cursor eher parallel für reine Code-Bearbeitung.

12. Vergleich mit Claude Code

Kriterium	Cursor	Claude Code
Grundform	Grafischer Editor, VS-Code-Fork mit eingebauten KI-Funktionen	Terminal-/CLI-Agent, laut Anthropic-Doku mit Anbindungen an IDE, Desktop-App und Browser
Modellbindung	Modellagnostisch: GPT, Claude, Gemini, Grok, weitere Drittmodelle, plus eigene Composer-Modelle	Primär auf Anthropic-Modelle ausgelegt; laut Doku auch über Amazon Bedrock oder Google Cloud Vertex AI routbar, Nutzung mit fremden Modellen wie GPT/Gemini nicht offiziell vorgesehen
Preismodell	Hobby kostenlos, Pro 20 USD/Monat, Pro+ 60 USD, Ultra 200 USD, Teams ab 40 USD/Nutzer, Enterprise individuell	Free, Pro 20 USD/Monat, Max ab 100/200 USD/Monat, Team-/Enterprise-Stufen mit Claude-Code-Zugriff
Reifegrad einzelner Bausteine	Rules seit früh etabliert, Hooks seit September 2025, Agent Skills erst seit Januar 2026	Rules/Skills/Hooks/Commands als Kernkonzept schon länger etabliert
Rules, Hooks, Skills, MCP, Permissions	Eigene Formate/Dateipfade für jeden Baustein, im Detail vgl. Kapitel 13	Eigene Formate/Dateipfade für jeden Baustein, im Detail vgl. Kapitel 13
Cloud-/Remote-Ausführung	Cloud Agents (eigene VMs, mobile Steuerung), Bugbot für PR-Review	Kein direkt vergleichbares, gleichnamiges Cloud-VM-Feature dokumentiert
CLI-Pendant	Eigene Cursor CLI seit August 2025	Claude Code selbst ist von Grund auf CLI-first

Bemerkenswert: Laut offizieller Anthropic-Dokumentation lässt sich Claude Code sogar direkt als Erweiterung in Cursor installieren – beide Werkzeuge schließen sich also nicht gegenseitig aus, sondern lassen sich kombinieren. Cursor punktet dort, wo eine vertraute, grafische VS-Code-Oberfläche, Modellfreiheit und eine sehr breite Erweiterungslandschaft im Vordergrund stehen. Claude Code punktet bei tiefem Terminal-/Scripting-Workflow und einem länger erprobten, granularen Rechte-/Hook-System, wobei Cursor mit Rules, Hooks, Skills, MCP und einem eigenen Permission-Modell inzwischen ein sehr ähnliches Instrumentarium aufgebaut hat – die Bausteine im Einzelnen vergleicht Kapitel 13.

13. Rules, Hooks, Commands, Skills – was davon gibt es bei Cursor?

Anders als bei manchen jüngeren, primär lokal ausgerichteten Agenten ist Cursor in diesem Bereich inzwischen fast vollständig auf Augenhöhe mit Claude Code, teils mit eigenen Namen für vergleichbare Konzepte:

Claude-Code-Konzept	Zweck	Entsprechung bei Cursor
`CLAUDE.md` / `.claude/rules/*.md`	Projektspezifische, dauerhaft geltende Regeln	**Vorhanden, sogar doppelt:** `.cursor/rules/*.mdc` (Project Rules, vier Ladearten, aber ohne automatisches Verzeichnis-Scoping) plus globale User Rules, UND zusätzlich `AGENTS.md` mit echter, zu `CLAUDE.md` analoger Verzeichnis-Vererbung – Details inklusive der nativen `CLAUDE.md`-Unterstützung in Kapitel 7.2/7.8
`.claude/settings.json` (Hooks)	Automatisierte Aktionen bei bestimmten Ereignissen	**Vorhanden:** `hooks.json`, projekt- oder nutzerweit, mit umfangreicher Event-Liste (Kapitel 7.6), seit September 2025
`.claude/commands/*.md` (Slash Commands)	Eigene, wiederverwendbare Befehlsabkürzungen	**Historisch vorhanden, inzwischen faktisch abgelöst:** Custom Commands (`.cursor/commands/*.md`) kamen im September 2025, wurden aber im Januar 2026 ohne formelle Abkündigung durch Agent Skills ersetzt; ein eingebauter Befehl `/migrate-to-skills` konvertiert bestehende Commands automatisch
`.claude/skills/*` (Skills)	Bei Bedarf ladbare Fähigkeitspakete	**Vorhanden:** Agent Skills (`SKILL.md` + Frontmatter), seit Januar 2026, mit genannter Kompatibilität zu `.claude/skills` (Kapitel 7.5)
MCP-Server	Anbindung externer Werkzeuge/Datenquellen	**Vorhanden:** `mcp.json`, projekt- oder nutzerweit, plus Team-Marketplace und dokumentiertes Deeplink-Install-Schema (Kapitel 7.3)

Claude-Code-Konzept	Zweck	Entsprechung bei Cursor
Permission-Regeln (allow/deny/ask)	Feingranulare, dauerhaft gespeicherte Freigaberegeln	**Vorhanden:** Run Modes plus `permissions.json` mit `terminalAllowlist`, `mcpAllowlist` und `autoRun`-Feldern (Kapitel 7.7)

Ein bemerkenswerter Unterschied zu früheren Notizzettel-Ansätzen: Cursor hatte mit **Notepads** ein weiteres Feature für wiederkehrende Prompts, das laut einem als Mitarbeiterbeitrag eingeordneten Forumspost Ende Oktober 2025 offiziell abgekündigt wurde – mit der Begründung, Rules, Commands, Memories und verbesserte Agent-Kontextfunktionen würden es vollständig ersetzen.

Fazit: Wer von Claude Code zu Cursor wechselt, muss die eigenen Artefakte nicht wie bei rein lokal ausgerichteten Agenten manuell nachbauen, sondern kann sie größtenteils eins zu eins in ein sehr ähnliches, ebenfalls dateibasiertes und versionierbares System überführen (siehe Kapitel 14). Der größte Unterschied liegt weniger im Vorhandensein dieser Bausteine als in Details der genauen Ladelogik, den Dateipfaden und der Tatsache, dass Cursors Skills-Feature erst seit wenigen Monaten existiert und sich entsprechend noch weiterentwickeln dürfte.

14. Umstieg von Claude Code: wie arbeitet man jetzt?

Die gute Nachricht vorweg: Da Cursor für fast jedes Claude-Code-Konzept eine dokumentierte, dateibasierte Entsprechung hat (Kapitel 13), ist der Umstieg deutlich direkter als bei einem Agenten ohne vergleichbares System. Die eigenen Claude-Code-Artefakte lassen sich größtenteils übertragen, nicht bloß als Rohmaterial für einen manuellen Nachbau verwenden.

14.1 Claude-Modelle in Cursor: Abo oder eigener API-Key?

Das ist meist die erste praktische Frage beim Umstieg, und ein häufiges Missverständnis: Ein bestehendes **Claude-Pro- oder Claude-Max-Abonnement** lässt sich **nicht** einfach in Cursor „einloggen" und dessen Kontingent weiterverwenden. Grund dafür ist eine bewusste Produkttrennung bei Anthropic selbst, nicht eine Einschränkung von Cursor: Laut einem offiziellen Anthropic-Support-Artikel gilt ausdrücklich „A paid Claude subscription enhances your chat experience but doesn't include access to the Claude API or Console" – das Consumer-Abo deckt claude.ai (Web/Desktop/Mobile) sowie, als Sonderfall, Claude Code selbst ab, aber keinen API-Zugriff über die Anthropic Console. API-Nutzung wird separat und nutzungsbasiert abgerechnet, unabhängig vom Consumer-Abo.

Für Cursor bedeutet das konkret zwei getrennte Wege, Claude-Modelle zu nutzen:

21. **Claude über das Cursor-Abo nutzen:** Claude-Modelle lassen sich direkt im Modell-Auswahldialog von Cursor wählen (Kapitel 5.1); die Nutzung läuft dann über das eigene Cursor-Guthaben (Hobby/Pro/Pro+/Ultra/Teams, Kapitel 9), abgerechnet zu Cursors Konditionen, nicht zu Anthropics Konsolenpreisen
22. **Eigenen Anthropic-API-Key hinterlegen:** Unter **Cursor Settings > Models** lässt sich beim Anbieter Anthropic ein eigener API-Key eintragen (offiziell dokumentiert unter „Bring Your Own API Key"). Ein solcher Key kann ausschließlich über die Anthropic Console (console.anthropic.com, Settings > API Keys) erzeugt werden – das setzt zwingend ein separates Console-Konto mit eigener, nutzungsbasierter Abrechnung voraus, unabhängig von einem eventuell vorhandenen claude.ai-Pro/Max-Abo. Diese Notwendigkeit eines Console-Kontos steht nicht wörtlich auf der Cursor-Hilfeseite, ergibt sich aber zwingend daraus, wie Anthropic API-Keys grundsätzlich ausstellt

Wichtige Einschränkung, ebenfalls offiziell dokumentiert: Ein eigener API-Key gilt laut Cursor „only with chat models" – **Tab-Autovervollständigung nutzt weiterhin ausschließlich Cursors eigene Modelle und Infrastruktur**, unabhängig davon, welcher Key hinterlegt ist. Wer einen eigenen Anthropic-Key primär nutzt, um Kosten zu sparen oder Anthropics eigene Datenschutzbedingungen statt Cursors Policy in Anspruch zu nehmen (siehe Kapitel 15), profitiert davon also nur bei Chat-/Agent-Anfragen, nicht bei Tab.

Zur Einordnung im Vergleich zu Claude Code selbst: Dort ist die Lage anders. Laut offiziellem Anthropic-Support lässt sich Claude Code direkt mit einem Pro- oder Max-Abo verbinden („With Pro and Max plans, you now have access to both Claude on the web, desktop, and mobile apps and Claude Code in your terminal with one unified subscription"), ganz ohne separaten API-Key. Ist zusätzlich eine `ANTHROPIC_API_KEY`-Umgebungsvariable gesetzt, nutzt Claude Code laut Community- und Anthropic-Quellen stattdessen diesen Key statt des Abos, was zu unerwarteter, separater API-Abrechnung führen kann – ein Hinweis darauf, dass Abo und API-Key auch innerhalb von Claude Code zwei getrennte Abrechnungswege bleiben, die sich gegenseitig überschreiben können. Beim Umstieg von Claude Code zu Cursor lohnt sich deshalb ein bewusster Blick auf die eigene Kostenstruktur: Wer schon ein Cursor-Abo

hat, fährt für Claude-Nutzung meist einfacher mit Weg 1; wer gezielt zu Anthropics eigenen API-Preisen und Datenschutzbedingungen wechseln will, braucht zwingend einen separaten Console-Account nach Weg 2.

14.2 Rules übernehmen: CLAUDE.md → .cursor/rules oder einfach liegen lassen

Der einfachste Weg zuerst, weil er oft übersehen wird: Wie in Kapitel 7.2 beschrieben, liest Cursor bestehende `CLAUDE.md`-Dateien nativ mit. Sie müssen also nicht zwingend umbenannt, verschoben oder inhaltlich angepasst werden, um in Cursor zu wirken. Das ist ein deutlicher Unterschied zu Agenten ohne dokumentierten Auto-Loader für Projektregeln.

Wer die Regeln trotzdem in Cursors natives Format überführen möchte, etwa um Cursors feineren Lademechanismus (Kapitel 7.2) zu nutzen, hat zwei Wege:

23. **Als `AGENTS.md` (empfohlen bei mehreren Unterverzeichnissen):** Bestehende `CLAUDE.md`-Dateien 1:1 in gleichnamig platzierte `AGENTS.md`-Dateien kopieren oder umbenennen – Projektstamm und Unterverzeichnisse funktionieren dabei genauso wie bei Claude Code, inklusive automatischer Vererbung mit Vorrang der spezifischeren, tieferliegenden Datei
24. **Als `.cursor/rules/*.mdc` (empfohlen bei feinerer Steuerung):** Für global geltende Konventionen (Codestil, Teamregeln) eine oder mehrere `.mdc`-Dateien unter `.cursor/rules` anlegen, `alwaysApply: true` setzen, wenn sie immer gelten sollen; für Regeln, die nur bei bestimmten Dateitypen relevant sind, `globs` statt `alwaysApply` setzen; für Regeln, die das Modell nur bei erkennbarer Relevanz selbst anfordern soll, nur `description` setzen. Da `.cursor/rules` kein automatisches Verzeichnis-Scoping kennt (Kapitel 7.2), eignet sich dieser Weg vor allem für Regeln, die ohnehin projektweit oder klar dateityp-bezogen gelten, weniger für eine tief verschachtelte Unterverzeichnis-Struktur
25. Persönliche, projektübergreifende Vorlieben (z. B. bevorzugte Erklärtiefe) als User Rules unter **Customize > Rules** hinterlegen statt im Projekt

Der Inhalt bestehender `CLAUDE.md`-Regeln lässt sich in allen drei Fällen meist unverändert übernehmen, nur Dateiname/-ort und bei `.mdc`-Dateien zusätzlich das Frontmatter sind neu zu entscheiden.

14.3 Skills übernehmen: Agent Skills statt .claude/skills

Da Cursor für `.claude/skills` ausdrücklich Kompatibilität nennt, ist im Idealfall gar kein manueller Nachbau nötig: den Ordner `.claude/skills` unverändert lassen oder nach `.cursor/skills/`.`agents/skills` kopieren und in Cursor prüfen, ob die Skills automatisch erkannt werden. Vor einer produktiven Nutzung sollte das im eigenen Projekt getestet werden, da „genannte Kompatibilität" nicht zwangsläufig hundertprozentige Formatgleichheit bedeutet. Alternativ hilft der eingebaute Befehl `/migrate-to-skills`.

14.4 Commands: von Custom Commands zu Skills

Da Cursors eigenes Custom-Commands-Feature selbst schon von Agent Skills abgelöst wurde, führt der pragmatische Weg direkt zu Skills: einen wiederkehrenden Claude-Code-Command

(`.claude/commands/*.md`) als eigenständigen Skill-Ordner mit `SKILL.md` neu anlegen, statt ihn als reinen Text-Baustein zu behandeln.

14.5 Hooks übernehmen: hooks.json statt settings.json

Die Event-Namen unterscheiden sich von Claude Code, das Prinzip aber nicht: Für jede bisherige Automatisierung (z. B. „nach jedem Edit automatisch linten") das passende Cursor-Event identifizieren – meist `afterFileEdit` oder `postToolUse` – und als Kommandodefinition in `.cursor/hooks.json` (projektbezogen) oder `~/.cursor/hooks.json` (global) eintragen. Für sicherheitsrelevante Prüfungen vor Shell-Ausführung eignet sich `beforeShellExecution` mit `failClosed`, um im Zweifel zu blockieren statt durchzulassen.

14.6 Permission-Regeln übernehmen: permissions.json

Ein bestehendes Claude-Code-`permissions`-Regelwerk (allow/deny/ask) lässt sich konzeptionell direkt auf `permissions.json` übertragen: erlaubte Terminal-Präfixe nach `terminalAllowlist`, erlaubte MCP-Werkzeuge nach `mcpAllowlist` im Format `server:tool`, und weiche Steuerungsanweisungen für den Auto-review-Klassifizierer nach `autoRun.allow_instructions`/`block_instructions`. Wichtig: Diese weichen Anweisungen sind laut Dokumentation keine harte Garantie, für wirklich kritische Aktionen bleibt ein bewusst gewählter Run Mode (Kapitel 7.7) das verlässlichere Mittel.

14.7 MCP-Server übernehmen

MCP ist ein offener, herstellerübergreifender Standard – eine bereits für Claude Code eingerichtete `mcp.json` lässt sich als Ausgangspunkt für Cursors `mcp.json` verwenden. Die Struktur (`mcpServers`-Objekt mit `command`/`args`/`env` bzw. `url`/`headers`) ist grundsätzlich kompatibel, sollte aber vor produktivem Einsatz gegen Cursors aktuelle Doku geprüft werden, insbesondere bei OAuth-Servern mit fest hinterlegten Redirect-URLs.

14.8 Cursor CLI als Alternative zu Claude Code

Wer den Terminal-zentrierten Arbeitsstil von Claude Code grundsätzlich beibehalten möchte, muss dafür nicht zwingend die grafische Cursor-Oberfläche nutzen: Seit dem 7. August 2025 gibt es eine eigenständige **Cursor CLI**, die einen Agenten direkt im Terminal oder headless in beliebigen Umgebungen (etwa CI/CD) ausführt. Sie bringt einen Shell Mode mit eingebauten Sicherheitsprüfungen, GitHub-Actions-Integration, MCP-Unterstützung und headless-fähigen Betrieb mit und ist damit konzeptionell das direkte Cursor-Pendant zu Claude Code, nutzt dabei aber dieselben Cursor-Modelle, -Rules, -Skills und -Hooks wie die grafische App.

15. Datenschutz und Sicherheit

- **Privacy Mode:** Bei aktiviertem Privacy Mode wird Code laut cursor.com/security nie zum Training verwendet, weder durch Cursor noch durch die angebundenen Modellanbieter, abgesichert durch technische Kontrollen und Zero-Data-Retention-Verträge mit den Modellanbietern. Bei Enterprise-Teams ist Privacy Mode standardmäßig aktiv, bei Einzelnutzern ist er Opt-in (auf Team-Ebene lässt er sich erzwingen). Wichtige Einschränkung: Diese Garantie gilt ausdrücklich für LLM-Anfragen; für den separaten Codebase-Indexierungsprozess ist sie nicht in gleicher Klarheit dokumentiert
- **SOC 2:** Ein SOC 2 Type II Attestationsbericht ist laut cursor.com/security auf Anfrage verfügbar, zusätzlich finden mindestens jährliche Penetrationstests durch Drittanbieter statt; der vollständige Bericht liegt im Trust Portal
- **Serverstandorte:** Anysphere nutzt und betreibt laut eigener Aussage keine Infrastruktur in China und setzt keine dort ansässigen Unternehmen als Subprozessoren ein. Optional gibt es „US-only Data Residency" für Inferenz/Vorschläge unterstützter Modelle; EU/EEA-Optionen befinden sich laut Dokumentation in Entwicklung. Für die Codebase-Indexierung gilt ausdrücklich: Liegt die Codebasis außerhalb der USA, kann US-only-Indexierung nicht garantiert werden
- **Codebase-Indexierung:** Cursor zerlegt Code für die semantische Suche in logische Blöcke (Funktionen, Klassen), erzeugt daraus Vektor-Embeddings und speichert diese in einer Vektordatenbank; Code-Inhalte werden laut Dokumentation nur während der Indexierung im Arbeitsspeicher gehalten und danach verworfen, Embeddings werden ohne dauerhafte Speicherung von Dateinamen oder Quellcode erzeugt, Dateipfade werden vor dem Versand verschlüsselt/obfuskiert. Indizierte Codebasen werden nach sechs Wochen Inaktivität gelöscht. Der konkrete Vektordatenbank-Anbieter wird auf der offiziellen Seite nicht genannt; mehrere technische Sekundärquellen nennen Turbopuffer, offiziell bestätigt ist das nicht
- **Checkpoints und Diffs zur Kontrolle:** Vorgeschlagene Änderungen erscheinen als Diff zur Prüfung vor der Übernahme, Checkpoints erlauben ein Zurückrollen (Kapitel 7.4). Welche Aktionen automatisch laufen und welche eine Bestätigung brauchen, hängt vom gewählten Run Mode ab (Kapitel 7.7) — die Dokumentation selbst weist ausdrücklich darauf hin, dass Auto-review keine Sicherheitsgrenze im strengen Sinn ist

16. Vorteile und Nachteile

Vorteile:

- Vertraute, ausgereifte VS-Code-Oberfläche mit breiter Erweiterungslandschaft über Open VSX
- Modellagnostisch: Zugriff auf praktisch alle relevanten Frontier-Modelle mehrerer Anbieter in einem Werkzeug, plus eigene, auf niedrige Latenz optimierte Composer-Modelle
- Inzwischen ein zu Claude Code weitgehend gleichwertiges Rules-/Hooks-/Skills-/MCP-/Permission-System, größtenteils dateibasiert und versionierbar
- Cloud Agents für asynchrone, parallele Hintergrundarbeit inklusive mobiler Steuerung, plus ein eigenständiges automatisiertes PR-Review-Tool (Bugbot)
- Eigene CLI für alle, die den Terminal-Workflow bevorzugen, ohne auf Cursors Modelle/Rules/Skills verzichten zu müssen

Nachteile:

- Kein primär lokal ausgerichtetes Werkzeug: echte lokale Modellnutzung läuft nur über einen nicht vollständig offiziell dokumentierten Umweg, und selbst dabei laufen Anfragen technisch über Cursors Server
- Reines Cloud-/Abo-Preismodell ohne kostenlose, voll funktionsfähige lokale Basisnutzung wie bei manchen Konkurrenzprodukten
- Kein Xcode-Ersatz; für klassische `.xcodeproj`-Projekte mit Targets/Schemes/Simulatoren ist man auf eine inoffizielle Community-Erweiterung (SweetPad) oder weiterhin auf Xcode selbst angewiesen
- Manche Bausteine (Agent Skills, aktueller Hooks-Reifegrad) sind erst seit wenigen Monaten verfügbar und dokumentarisch noch nicht so gefestigt wie bei länger etablierten Werkzeugen
- Größere Unternehmensunsicherheit durch die angekündigte, aber noch nicht abgeschlossene SpaceX-Übernahme, mit unklaren langfristigen Folgen für Produktausrichtung und Datenverarbeitung

17. FAQ

17.1 Kann ich vorhandene Ollama-Modelle nutzen?

Nur über einen Umweg und nicht mit einer dedizierten, offiziell dokumentierten Anbindung. Community-Quellen beschreiben übereinstimmend den „Override Base URL"-Mechanismus unter Settings > Models > OpenAI API, über den sich Ollamas OpenAI-kompatibler Endpunkt eintragen lässt (siehe Kapitel 5.3). Da Anfragen dabei laut offiziellem Routing-Prinzip trotzdem über Cursors Server für den Prompt-Aufbau laufen, ist das nicht mit einer wirklich vollständig lokalen, offline-fähigen Ausführung gleichzusetzen.

17.2 Brauche ich zusätzlich VS Code, um Cursor zu nutzen?

Nein. Cursor ist eine vollständig eigenständige Anwendung, kein Aufsatz, der VS Code voraussetzt. Wer bereits VS Code nutzt, kann über den offiziell dokumentierten „VS Code Import" lediglich vorhandene Erweiterungen, Themes, Einstellungen und Tastenkürzel übernehmen (Kapitel 4.2).

17.3 Wo werden Rules gespeichert, und muss ich sie irgendwo aktivieren?

Project Rules liegen als `.mdc`-Dateien unter `.cursor/rules` im Projekt und werden automatisch geladen, abhängig von ihrem Frontmatter (`alwaysApply`, `globs`, `description`, siehe Kapitel 7.2). Eine zusätzliche manuelle „Aktivierung" ist für automatisch geladene Rules nicht nötig; nur Regeln ohne `globs`/`description`/`alwaysApply` müssen im Chat explizit per `@`-Erwähnung eingebunden werden. Damit unterscheidet sich Cursor deutlich von Werkzeugen ohne dokumentierten Auto-Loader für Projektregeln.

17.4 Funktioniert Cursor auch offline?

Nur sehr eingeschränkt. Die KI-Kernfunktionen (Chat, Agent, Tab) benötigen eine Internetverbindung, da die verwendeten Modelle als Cloud-Dienste laufen und Anfragen ohnehin über Cursors Server geleitet werden. Reines Editieren ohne KI-Funktionen ist als VS-Code-Fork grundsätzlich auch offline möglich, das ist aber nicht der eigentliche Zweck des Produkts.

17.5 Was ist der Unterschied zwischen einer Rule, einem Skill und einer AGENTS.md-Datei?

Eine Rule (`.cursor/rules/*.mdc`, Kapitel 7.2) ist eine kurze, meist thematisch fokussierte Anweisung, deren Geltungsbereich unabhängig vom Speicherort über Frontmatter-Felder (`alwaysApply`/`globs`/`description`) gesteuert wird, aber ohne automatisches Verzeichnis-Scoping. Ein Agent Skill (Kapitel 7.5) ist ein umfangreicheres, strukturiertes Paket für eine wiederkehrende, komplexere Aufgabe, inklusive optionaler Skripte und Referenzdateien, das zusätzlich auch verschachtelt in Unterverzeichnissen liegen und dann automatisch nur dort gelten kann. `AGENTS.md` ist dagegen kein Cursor-spezifisches Konzept, sondern ein herstellerübergreifender, offener Standard für

Projektanweisungen an Coding-Agenten allgemein; Cursor unterstützt `AGENTS.md` laut Dokumentation ausdrücklich mit echter, automatischer Vererbung über Unterverzeichnisse hinweg, strukturell identisch zu Claude Codes `CLAUDE.md`-Modell – inklusive der nativen `CLAUDE.md`-Unterstützung selbst (Kapitel 7.2, 7.8, 14.2).

18. Vergleich: Cursor, Claude Code, Windsurf/Devin Desktop, GitHub Copilot & Co.

Einordnung vorab: Wie schon im entsprechenden Kapitel des LM-Studio-Bionic-Tutorials ist dieser Abschnitt eine persönliche, subjektive Einschätzung, keine reine Faktendarstellung. Stand ist Juli 2026, das Feld verändert sich in diesem Bereich derzeit sehr schnell (siehe die Windsurf-Umbenennung unten).

18.1 Vergleichstabelle

Werkzeug	Kategorie	Lizenz/ Offenheit	Oberfläche	Modellbindung	Reifegrad (Juli 2026)
Cursor	Grafischer VS-Code-Fork mit KI-Agent	Proprietär, kostenlose Basisnutzung	Desktop-App (macOS/Windows/Linux) + eigene CLI	Modellagnostisch (GPT/Claude/Gemini/Grok u. a.) + eigene Composer-Modelle	Ausgereift, sehr schnell weiterentwickelt
Claude Code	Terminal-/CLI-Agent	Proprietär	CLI + IDE-Extensions	Primär Anthropic-Modelle, optional über Bedrock/Vertex AI	Ausgereift, lange am Markt
Windsurf / Devin Desktop	IDE-Agent im Cursor-Stil	Proprietär	Desktop-App	Providerabhängig je nach Anbieter	In Umbenennung/ Umbau (Juni 2026)
GitHub Copilot (Coding Agent + CLI)	Cloud-Agent + CLI, tief in GitHub integriert	Proprietär	GitHub-Weboberfläche, CLI, IDE-Extensions	Claude-, GPT- und Gemini-Modelle wählbar	Copilot CLI seit Februar 2026 allgemein verfügbar

18.2 Cursor

Cursor wirkt Mitte 2026 wie das am schnellsten wachsende und am breitesten ausgestattete Produkt in diesem Vergleich: eine vertraute, ausgereifte Editor-Oberfläche, praktisch freie Modellwahl, dazu ein inzwischen zu Claude Code weitgehend gleichwertiges Rules-/Hooks-/Skills-/MCP-System (Kapitel 13). Die größte Unsicherheit ist derzeit weniger technischer als unternehmerischer Natur: die angekündigte, aber noch nicht abgeschlossene SpaceX-Übernahme (Kapitel 2), deren langfristige Auswirkungen auf Produktausrichtung und Datenverarbeitung sich zum Stand dieser Anleitung noch nicht abschätzen lassen.

18.3 Claude Code

Claude Code bleibt Stand Juli 2026 eines der technisch ausgereiftesten Terminal-zentrierten Werkzeuge, mit einem lange erprobten Zusammenspiel aus Permission-System, Hooks, Skills und Subagenten.

Bemerkenswert ist, dass sich Claude Code laut offizieller Anthropic-Dokumentation auch direkt als Erweiterung in Cursor nutzen lässt – beide Werkzeuge stehen also nicht zwingend in direkter Konkurrenz zueinander, sondern lassen sich kombinieren, etwa Claude Code als Agent innerhalb der Cursor-Oberfläche.

18.4 Windsurf / Devin Desktop

Windsurf wurde entgegen früherer Gerüchte nicht von OpenAI, sondern im Juli 2025 von Cognition (Hersteller des Agenten Devin) übernommen. Laut mehreren unabhängigen Tech-Quellen wurde Windsurf am 2. Juni 2026 in „Devin Desktop" umbenannt, die bisherige lokale KI „Cascade" durch einen neu geschriebenen Nachfolger „Devin Local" ersetzt, Cascade blieb nur noch bis zum 1. Juli 2026 als Übergangsoption verfügbar. Diese Details stammen überwiegend aus Tech-Blogs, nicht aus einer direkt abgerufenen offiziellen Cognition-Pressemitteilung, sollten vor Veröffentlichung also nochmal geprüft werden.

18.5 GitHub Copilot

GitHub Copilot Workspace wurde als reine Preview eingestellt. An dessen Stelle stehen heute der GitHub Copilot Coding Agent (auch „Cloud Agent" genannt), der eigenständig recherchiert, plant und Pull Requests öffnet, sowie die GitHub Copilot CLI, seit dem 25. Februar 2026 allgemein verfügbar, mit Plan Mode und Autopilot Mode und Unterstützung unter anderem für Claude-, GPT- und Gemini-Modelle. Naheliegende Wahl für Teams, die ohnehin durchgängig auf GitHub setzen.

18.6 Weitere nennenswerte Agenten

Ohne Anspruch auf Vollständigkeit, kurz eingeordnet:

- **Cline** – vollständig quelloffener, modellagnostischer Agent als VS-Code-Extension mit Anbindung an über 30 Provider
- **Continue** – ebenfalls Open Source und modellagnostisch, verfügbar als VS-Code-Extension, JetBrains-Plugin und CLI
- **Aider** – schlanker, sehr Terminal-fokussierter Open-Source-Pionier unter den CLI-Coding-Agenten; das Veröffentlichungstempo hat sich 2025/2026 laut Release-Historie spürbar verlangsamt, in der Community gibt es Kritik an nachlassender Pflege
- **OpenCode** – provider-agnostischer Open-Source-Agent fürs Terminal, nutzbar mit Claude-, OpenAI- und Google-Modellen sowie lokalen Modellen, mit Build- und Plan-Agent-Modi sowie Multi-Session-Fähigkeit
- **LM Studio Bionic** – anders gelagert als die übrigen hier genannten Werkzeuge: eigenständiger GUI-Agent mit Fokus auf lokale, offene Modelle statt auf Cloud-Modellvielfalt (siehe separates Tutorial „LM Studio Bionic – Die komplette Anleitung")

18.7 Fazit

Für eine vertraute, grafische Editor-Umgebung mit größtmöglicher Modellfreiheit und einem inzwischen sehr vollständigen Konfigurationssystem ist Cursor derzeit eines der umfassendsten Werkzeuge im

Vergleich. Wer dagegen einen tief in einen Terminal-/Scripting-Workflow integrierten, lange erprobten Agenten bevorzugt, bleibt mit Claude Code gut bedient – und kann es bei Bedarf sogar innerhalb von Cursor nutzen. Quelloffene, providerunabhängige Alternativen ohne Bindung an einen einzelnen kommerziellen Anbieter bieten Cline, Continue oder OpenCode. Und wer konsequent lokale Ausführung offener Modelle priorisiert, ist mit einem darauf spezialisierten Werkzeug wie LM Studio Bionic besser bedient als mit Cursor.

19. Quellen

Diese Anleitung basiert auf einer Web-Recherche (Stand Juli 2026). Die wichtigsten verwendeten Quellen, gegliedert nach Themenblock:

Cursor – offizielle Dokumentation (docs.cursor.com, cursor.com/help):

- Cursor – offizielle Website
- Cursor-Dokumentation (Startseite)
- Rules
- Rules – Help Center (Ablage, User Rules, Profil-Export)
- Subagents
- Cloud Agent – Setup (`environment.json`)
- Hooks
- Agent Skills
- MCP
- MCP Install Links
- Permissions
- Run Modes
- Agent Overview
- Checkpoints
- Cloud Agent
- Bugbot
- Installation
- VS-Code-Migration
- Extensions
- iOS and macOS Swift
- Update Access
- Models and Pricing
- Max Mode
- Codebase Indexing
- Enterprise Privacy and Data Governance
- Downloads
- Pricing
- Security
- Trust Portal
- Tab
- Agent-Modus
- Ask-Modus
- Verfügbare Modelle
- API-Keys
- Nutzungslimits

Cursor – Blog & Changelog:

- Cursor 2.0 Ankündigung
- Composer
- Composer 2.5

- Cursor CLI
- Agent Sandboxing
- Secure Codebase Indexing
- Bugbot Updates Juni 2026
- Bugbot-Preisänderung Mai 2026
- Teams-Preise Juni 2026
- Changelog
- Changelog 1.6
- Changelog 1.7
- Changelog 2.0
- Changelog 3.0
- Cursor CLI (Produktseite)

Anysphere: Unternehmen, Finanzierung, SpaceX-Übernahme:

- Anysphere – Wikipedia
- Forbes: Cursor-Mitgründer werden Milliardäre (13.11.2025)
- CNBC: SpaceX übernimmt Cursor (16.06.2026)
- Forbes: SpaceX-Cursor-Deal
- Yahoo Finance: SpaceX-Cursor-Deal
- Yahoo Finance: Einordnung SpaceX-Cursor-Deal
- TechCrunch: Seed-Runde (2023)
- TechCrunch: Series A (2024)
- TechCrunch: Series B (2024)
- TechCrunch: Series C (2025)
- Institutional Investor: Series D (2025)
- DevClass: VS-Code-Marketplace-Streit

Swift/Xcode-Integration:

- Swift.org: Setting up Cursor for Swift Development
- Swift.org-Blog: Expanding Swift IDE Support
- swiftlang/vscode-swift
- swiftlang/sourcekit-lsp
- SweetPad

Claude Code / Anthropic:

- Claude Code – offizielle Doku
- Claude – Preise
- Anthropic Support: Warum ein Claude-Abo kein API-/Console-Zugriff enthält
- Anthropic Support: Claude Code mit Pro-/Max-Abo nutzen

Andere Agenten (Kapitel 18):

- Cognition: Windsurf-Übernahme
- Windsurf wird Devin Desktop (Sekundärquelle)
- GitHub Copilot Coding/Cloud Agent
- GitHub Copilot CLI – General Availability
- Cline
- Continue

- Aider
- OpenCode

Zur Oberfläche (Sidebar, „+"-Menü, Settings-Kategorien, Repo-Picker), Stand Version 3.11/3.12:

- Changelog: Side Chats and Conversation Search (Version 3.11, 10.07.2026)
- Plan Mode
- Debug Mode
- Multi-Agent / Multitask
- Automations
- Customize Cursor
- GitHub-Integration
- Worktrees
- Plugins
- PR Inbox
- Browser-Tool
- Netzwerkkonfiguration (Enterprise)
- Tab-Einstellungen
- Codebase-Indexierung (Help Center)
- Öffentliche Profile
- Themes/Appearance
- Overages/On-Demand Usage
- Cloud-Agent-Settings
- Forum: Hobby-Plan und Upgrade-Hinweis
- Forum: Usage-Limit-Meldung

Hinweis zur Recherchequalität

Mehrere Details in dieser Anleitung sind nur über Community-/Forumsquellen oder unabhängige Presseberichte belegt, nicht über eine offizielle Cursor-Primärquelle – dazu gehören unter anderem: die genaue Priorität bei MCP-Namenskonflikten zwischen Projekt- und globaler Konfiguration, das exakte Umbenennungsdatum von „YOLO mode" zu „Auto-Run", ob der Beta-Status von Hooks offiziell aufgehoben wurde, die genauen Details der Windsurf-Umbenennung in „Devin Desktop", der Ollama/LM-Studio-Anbindungsmechanismus über „Override Base URL", die Existenz und Herkunft des Modells „Cheetah", exakte Jahrespreise für Pro/Pro+/Ultra sowie die Mitarbeiterzahl und die exakte ARR-Zahl von Anysphere. Der SpaceX-Übernahme-Deal selbst gilt dagegen als durch mehrere unabhängige, seriöse Presseorgane mit übereinstimmenden Eckdaten belegt.

Zusätzlich bei der Oberflächen-Beschreibung (Sidebar, „+"-Menü, Repo-Picker, Settings) nur als plausible Ableitung, nicht wörtlich mit einer Quelle belegt: die exakte Gesamtstruktur des „+"-Menüs im Composer-Eingabefeld als ein zusammenhängendes UI-Element mit allen sieben Einträgen plus Modell-Picker; die exakte Klick-Interaktion von „Image", „Skills" und „MCP Servers" in diesem Menü; die Beschriftungen „Use Existing" und „New Folder" im Workspace-Picker; ob „On This Mac" (statt „On This Computer") tatsächlich die aktuelle macOS-Beschriftung ist; „Repositories" als exakter Sidebar-Label-Text; die vollständige, aktuelle Optionsliste unter Settings → Beta und Settings → General; sowie die exakte UI-Position eines dauerhaften „Upgrade to Pro"-Hinweises im Hobby-Plan. Für alle diese Punkte gilt: Einzelfunktionen sind überwiegend offiziell belegt, ihre exakte Anordnung und Beschriftung in der aktuellen Live-App aber nicht mit Screenshot-Quellen verifiziert.