NEWS · OCTOBER 26, 2026

GitKraken für Entwickler 2026: Agenten, Konflikte und der Commit-Graph

Ein Arbeitstag in GitKraken Desktop im Jahr 2026 -- Agenten in Worktrees auf dem Commit-Graphen, Commit Composer macht aus ihrer Ausgabe prüfbare Commits, KI-Konfliktauflösung mit Konfidenzwerten und Konflikterkennung, bevor der Pull Request überhaupt existiert.

By Dr. Holger Flick



Vor ein paar Jahren hatte mein Arbeitstag in Git eine einfache Form. Code schreiben, committen, pushen, Pull Request öffnen, den einen Konflikt pro Woche beheben, von vorn. Der Commit-Graph in GitKraken Desktop war das, was ich morgens anschaute, um zu sehen, was das Team gemacht hatte, und danach verschwand er im Hintergrund. Der Code war die Arbeit.

Diese Form gibt es nicht mehr. An einem typischen Tag bin ich heute nicht mehr der Einzige, der in meinen Repositories Code schreibt. Ein Coding-Agent arbeitet in einem Branch an einem Bugfix, während ich prüfe, was ein anderer Agent gestern mit der Datenschicht angestellt hat, und der Pull Request einer Kollegin steuert auf dieselbe Header-Komponente zu, die einer dieser Agenten gerade anfasst. Das Tippen ist billig geworden. Den Überblick zu behalten, was getippt wurde, von wem, in welchen Branch, und ob es mit etwas anderem kollidiert, ist zur eigentlichen Aufgabe geworden. Das habe ich im Juni schon vertreten, als GitKraken sich zur Code-Flow-Company erklärte, und nach vier Monaten agentenlastiger Arbeit glaube ich es mehr, nicht weniger.

In diesem Beitrag gehen wir deshalb einen Arbeitstag in GitKraken Desktop durch, so wie er Ende 2026 aussieht. Der Tag ist eine Montage -- das Los-Pollos-Hermanos-Repository aus meinen beiden letzten Beiträgen, mit den Agenten und dem Pull Request der Kollegin so angeordnet, dass jede Station genau einmal

vorkommt --, aber jede Funktion verhält sich so, wie es die Dokumentation und meine eigene Nutzung beschreiben. Keine Funktionsliste, sondern die Abfolge, wie sie passiert: Agenten, die in Worktrees gestartet und auf dem Graphen beobachtet werden, ihre Ausgabe zu Commits komponiert, die ein Mensch prüfen kann, ein Konflikt mit KI aufgelöst und dann nachgeprüft, und der Konflikt, der nie einer wurde, weil das Werkzeug mich vorher gewarnt hat. Jede Station verweist auf einen tieferen Beitrag, in dem ich eine Funktion bereits auseinandergenommen habe. Wo ich meine, dass man GitKrakens KI vertrauen darf, sage ich es, und wo ich meine, dass man ihr ungelesen nicht vertrauen darf, sage ich das auch.

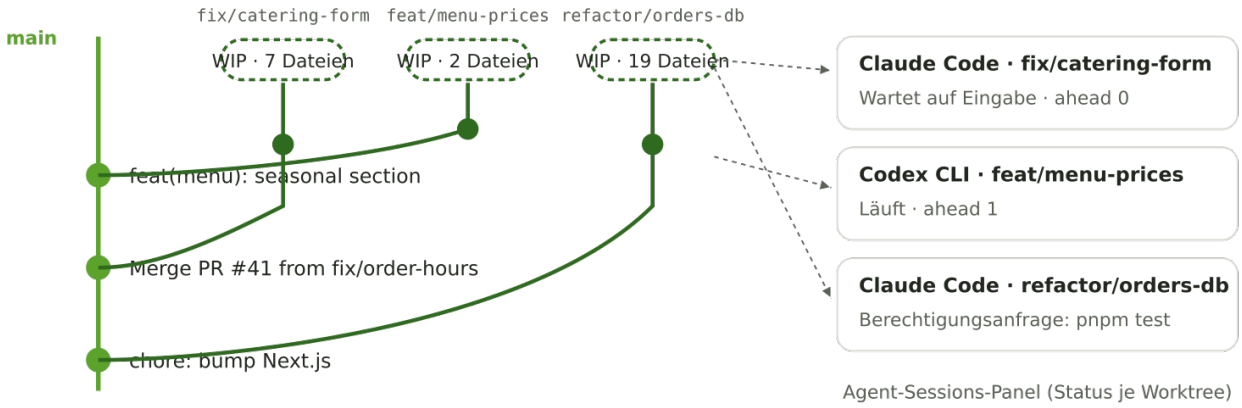
Volle Offenlegung, bevor wir anfangen: Ich bin GitKraken Ambassador. Das verschafft mir frühe Einblicke in Funktionen, aber keinen Grund, die Teile auszulassen, die mich nicht überzeugt haben.

Morgens: drei Agenten, drei Worktrees, ein Graph

Beginnen wir dort, wo die meisten meiner Tage inzwischen beginnen: damit, dass Arbeit delegiert und nicht erledigt wird. Ein Git-Worktree ist ein zweites Arbeitsverzeichnis, das an dasselbe Repository angehängt und auf einem eigenen Branch ausgecheckt ist, sodass zwei Dinge gleichzeitig im selben Repository passieren können, ohne sich gegenseitig auf die Füße zu treten. Git liefert das seit Version 2.5 im Juli 2015 mit. Hand aufs Herz: Vor 2026 hatte ich es vielleicht zweimal benutzt. Der Befehl ist in Ordnung. Der Grund, warum sich niemand darum kümmerte, war, dass ein Mensch ohnehin nur in ein Verzeichnis gleichzeitig tippen kann.

Coding-Agenten haben das über Nacht geändert. Ein Agent wie Claude Code oder Codex CLI arbeitet mit Vergnügen zwanzig Minuten lang in seinem eigenen Verzeichnis, und sobald Sie zwei davon im selben Checkout laufen lassen, überschreiben sie sich gegenseitig die Dateien. Worktrees sind die naheliegende Lösung, und sie sind von Hand mühsam einzurichten: Branch anlegen, Verzeichnis anlegen, Abhängigkeiten installieren, Agent starten, hinterher ans Aufräumen denken. Seit GitKraken Desktop 12.0 im April 2026 ist das eine einzige Aktion. "New Agent Session" legt den Worktree an, führt die von Ihnen konfigurierten Setup-Befehle aus und startet den Agenten Ihrer Wahl; das Release unterstützte Claude Code, Codex CLI, Copilot CLI, Gemini CLI und OpenCode, und die Release Notes ergänzten in 12.5 die Cursor CLI. Ist die Arbeit gemergt, entfernt ein Klick den Worktree und, seit 12.1, den Branch gleich mit.

Der Teil, der mir wichtig ist, ist nicht das Starten. Es ist das, was der Commit-Graph daraus macht. Seit 12.3 zeigt der Graph für jeden Worktree mit uncommitteten Änderungen oder einer laufenden Agent-Session einen eigenen Work-in-Progress-Knoten, und seit 12.6.1 nennt jede dieser Zeilen den Branch, auf dem der Worktree steht. Das folgende Diagramm zeigt sinngemäß, wie das aussieht: drei Agenten auf drei Branches und ich auf `main`.



Drei Agenten-Worktrees, jeder mit eigenem Work-in-Progress-Knoten, neben den Branches, aus denen sie wachsen

Lesen Sie das Diagramm von links nach rechts. Der Graph links ist schlichtes Git: `main`, drei Branches und drei gestrichelte Work-in-Progress-Knoten, die noch keine Commits sind. Die Karten rechts sind das Agent-Ses-

sions-Panel, das meldet, was jeder Agent gerade tut, und seit 12.4 zeigt es auch die Berechtigungsanfrage eines Agenten direkt dort an, mit Erlauben- und Ablehnen-Aktionen. Der Punkt ist, dass beide Hälften dasselbe Bild sind. "Läuft · ahead 1" auf der Karte und der gestrichelte Knoten auf `feat/menu-prices` beschreiben denselben Zustand. Sagt die Karte also, ein Agent warte auf Eingabe, kann ich auf dem Graphen sehen, was er bisher produziert hat, bevor ich antworte.

Das Erste, was GitKraken mir morgens gibt, ist also gar keine Git-Operation. Es ist die Möglichkeit, auf einen Bildschirm zu schauen und zu wissen, welcher der drei Arbeitsströme am weitesten ist, welcher sich von `main` entfernt hat und welcher `pnpm test` ausführen will und auf meine Erlaubnis wartet. Meiner Meinung nach ist das die größte Veränderung darin, *wofür* ein Git-Client da ist. 2024 zeigte mir der Graph die Vergangenheit.

2026 zeigt er mir die Gegenwart.

> **Für nichts davon brauchen Sie eine GUI**
>
> Alles oben ist darunter schlichtes Git. `git worktree add ../catering-form -b fix/catering-form` legt denselben Worktree an, und `git log --graph --oneline --all` zeigt dieselben Branches, wie ich im Forensik-Beitrag gezeigt habe. GitKraken führt Ihre Agenten nicht aus und schreibt keine Prompts für sie; die eigene Seite zum Agent-Management sagt, es starte die Agent-CLIs lediglich und zeige, was sie tun. Wer im Terminal lebt, findet in lazygit eine ausgezeichnete Open-Source-Alternative mit Graph, Worktree-Unterstützung und Merge-Ansicht, und `gitk` liefert Git selbst mit. Was GitKraken hinzufügt, ist die Integration: Worktree, Agentenstatus und Graph an einem Ort, mit den Setup-Befehlen, die für Sie ausgeführt werden.

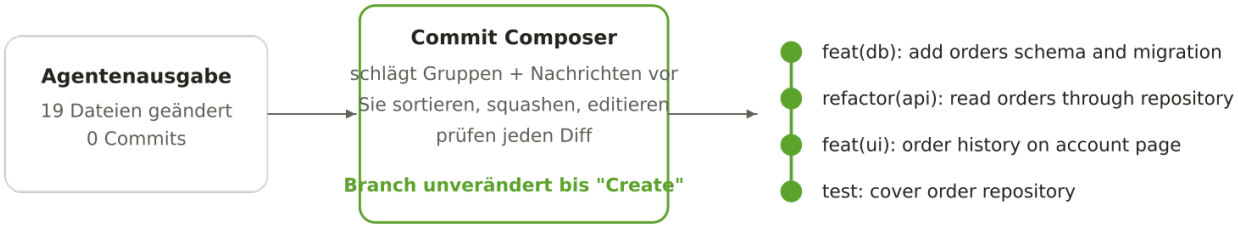
Später Vormittag: aus dem Änderungshaufen eines Agenten Commits machen

Jetzt kommt der Teil agentengetriebener Arbeit, dem meiner Ansicht nach zu wenig Aufmerksamkeit zukommt. Der Agent auf `refactor/orders-db` ist fertig, und ich blicke auf neunzehn geänderte Dateien in einem Arbeitsverzeichnis. Agenten sind schnell, aber sie sind nicht gut darin, zum Committen innezuhalten. Sich selbst überlassen, produzieren sie einen Commit namens "Refactor orders database layer", der ein Schema, drei API-Routen, zwei Komponenten und einen Test-Helfer berührt, oder, schlimmer, gar keinen Commit und eine Wand uncommitteter Änderungen. Ich habe schlechten Commit-Nachrichten einen ganzen Liebesbrief geschrieben, und Agenten haben ein neues Genre hinzugefügt: Nachrichten, die grammatisch perfekt sind und zwölf unzusammenhängende Dinge auf einmal beschreiben.

Commit Composer ist GitKrakens Antwort darauf, und es ist die KI-Funktion, die ich am meisten nutze.

Sie stagen die Änderungen, klicken "Compose Commit with AI", und das Werkzeug schlägt eine Reihe von Commits vor, jeder mit den Dateien und Hunks, die seiner Meinung nach zusammengehören, und einer Nachricht für jeden. Am Branch passiert nichts, bis Sie "Create commits" klicken. Dazwischen können Sie die vorgeschlagenen Commits umsortieren, zwei davon squashen, Nachrichten bearbeiten, den Diff jedes vorgeschlagenen Commits prüfen oder den ganzen Vorschlag zurücksetzen und neu beginnen. Dasselbe Werkzeug funktioniert auch rückwärts: Wählen Sie einen zusammenhängenden Bereich bestehender Commits auf dem aktuellen Branch, und "Recompose Commits with AI" gruppiert sie neu, mit der Einschränkung, dass der Bereich keine Merge-Commits enthalten darf.

Der Ablauf für die Ausgabe eines Agenten sieht so aus.



Von einem Haufen Agentenänderungen zu einer prüfbaren Folge von Commits

Der mittlere Kasten des Diagramms ist der Ort, an dem Sie Ihre Zeit verbringen, und die fette Zeile an seinem unteren Rand ist die Eigenschaft, auf die es mir ankommt: Der Vorschlag ist ein Entwurf, kein Commit. Die Release Notes zu 12.5 ergänzten drei Details, die das auf echter Agentenausgabe statt in Demos benutzbar machen. Die KI-Ausgabe wird automatisch validiert und erneut angefordert, statt zu scheitern, wenn das Modell etwas Fehlerhaftes zurückgibt; Lockfiles werden vom Token-Budget ausgenommen, sodass eine `pnpm-lock.yaml` mit tausend geänderten Zeilen die Anfrage nicht mehr auffrisst; und neu komponierte Commits behalten ihre ursprünglichen Autoren, mit Co-Autoren, wenn ein Commit die Arbeit mehrerer Personen zusammenführt. Letzteres ist wichtiger, als es klingt. Komponiere ich einen Branch neu, den eine Kollegin und ein Agent beide angefasst haben, sagt die Historie immer noch, wer was getan hat.

Ich würde meine Begeisterung allerdings eingrenzen. Commit Composer ist in der Dokumentation nach wie vor als Preview gekennzeichnet, und es gruppiert nach dem, wie der Diff aussieht, nicht nach dem, was Sie beabsichtigt haben. Bei einem Refactoring wie dem der Bestellungen rechnen Sie damit, dass es Migration und API-Änderung in einen Commit packt, wenn beide dieselben Typdefinitionen berühren. Das ist eine vertretbare Gruppierung und das Gegenteil davon, wie ich sie prüfen möchte, also trenne ich sie. Das Werkzeug macht daraus eine Bearbeitung von zehn Sekunden statt einer `git add -p`-Sitzung von zehn Minuten, und das ist die ehrliche Größe des Gewinns: Es kennt Ihre Review-Kultur nicht, aber es macht es billig, sie anzuwenden.

> **Sagen Sie ihm Ihre Konventionen einmal**
> GitKrakens KI-Einstellungen nehmen eigene Anweisungen je Funktion entgegen, und das Beispiel der Dokumentation selbst ist ein verpflichtendes Conventional-Commit-Präfix wie `feat:` oder `fix:-`. Ich habe das einmal gesetzt, zusammen mit "ein Commit pro Anliegen, Tests getrennt", und die Vorschläge entsprechen seither dem Hausstil. Dieselben Anweisungen gelten für Commit-Nachrichten, Stash-Nachrichten und Pull-Request-Beschreibungen, Sie schreiben sie also nur einmal.

Früher Nachmittag: der Konflikt, aufgelöst von der KI und dann von mir

Weiter zu dem Moment, den jeder Team-Entwickler kennt. Der Branch `feat/menu-prices` des Codex-Agenten ist fertig, aber der Pull Request einer Kollegin ist vor einer Stunde auf `main` gelandet und hat denselben Header berührt. Einen echten Konflikt genau dieser Art habe ich Anfang des Monats Hunk für Hunk auseinandergenommen, daher bleibe ich hier dabei, wie die KI in den Tag passt, statt die Auflösung zu wiederholen.

GitKrakens Merge-Tool zeigt die beiden Seiten oben und die Ausgabe unten, und Sie wählen Hunks per Checkbox oder bearbeiten die Ausgabe direkt. Seit Version 11.2 füllt die Schaltfläche "Auto-resolve with AI" die Ausgabe mit einer vorgeschlagenen Auflösung, einer schriftlichen Erklärung, warum sie gewählt hat, was sie gewählt hat, und einem Konfidenzwert für jeden konfliktbehafteten Hunk. Sie übernehmen, bearbeiten oder verwerfen dann. Die Funktion ist in der Dokumentation weiterhin als Preview gekennzeichnet, setzt einen bezahlten Plan voraus, und ein Fix in 12.6 brachte sie mit den neuesten Claude-Modellen zum Laufen, wenn Sie Ihren eigenen Anthropic-Schlüssel mitbringen.

Was ich in ein paar Monaten Nutzung gelernt habe: Die Konfidenzzahl bedeutet etwas Bestimmtes, und das ist nicht "wie richtig der Merge ist". Im Header-Konflikt behielt die KI beide Import-Blöcke und beide Menü-Links mit 100 % und führte zwei verschiedene Sätze von Tailwind-Klassen am `<nav>` mit 94 % zusammen. Alle drei waren korrekte Git-Merges. Sie entschied außerdem, mit 100 % Konfidenz, dass der Link "Order online" vor "Locations" gehört, weil das "typical primary-action ordering" sei. Das ist eine Produktentscheidung, und kein Modell liest die Gedanken des Product Owners. Die Zahl sagt Ihnen, wie sicher sich das Modell ist, die beiden Texte gut kombiniert zu haben. Sie sagt nichts darüber, ob der kombinierte Text das ist, was die beiden Menschen gemeint haben.

Meine Regel ist also einfach, und ich verteidige sie: Die KI-Auflösung ist der erste Entwurf für jeden Konflikt, und die Konfidenzwerte sagen mir, wo ich zuerst lesen soll, nicht, wo ich aufhören kann. Imports, Listen und Lockfiles sind Fleißarbeit, die ich gerne abgebe. Alles, wo die beiden Seiten das *Verhalten* geändert haben,

etwa eine Bedingung, einen Standardwert, eine Absicherung oder die Reihenfolge von Operationen, wird von einem Menschen gelesen, der weiß, was beide Autoren wollten, egal, was die Zahl sagt. 94 % auf einem `className`-String sind ein Hinweis. 94 % auf einem `if` sind eine Einladung, sehr genau hinzusehen.

> **Wo die KI-Auflösung nicht hilft**
>
> Der Fehlerfall, auf den Sie achten müssen, ist die Auflösung, die kompiliert, den Type-Check besteht und trotzdem ein Feature löscht. Ich habe eine im Merge-Konflikt-Beitrag gezeigt: Wählen Sie bei einem Hunk
>
> die falsche Seite, und der Build ist grün, während das responsive Menü der Kollegin verschwunden ist. Kein Konfidenzwert fängt eine Auflösung ab, die syntaktisch perfekt und semantisch falsch ist, denn das Modell führt Ihre Anwendung nicht aus. Ihre Tests vielleicht. Ein Reviewer, der die Absicht beider Branches liest, ganz sicher. Beachten Sie außerdem, dass in selbst gehosteten und Serverless-Installationen von GitKraken der eingebaute, von GitKraken gehostete Anbieter nicht verfügbar ist und laut Dokumentation
>
> eine Custom URL die einzige KI-Konfiguration ist.

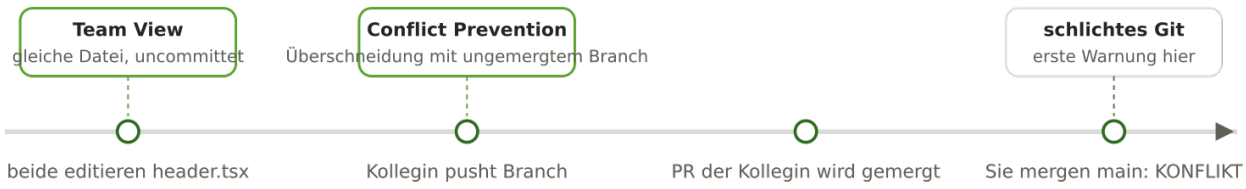
Mitten am Nachmittag: der Konflikt, der nie passiert ist

Der beste Konflikt ist der, den Sie auflösen, bevor Git ihn je sieht, und hier schlägt ein Client, der mit einem Dienst spricht, eine schlichte Git-Installation. Seit GitKraken Desktop 10.8 im März 2025 zeigt die Toolbar

ein Warnsymbol, wenn Ihr Branch Konflikte mit seinem Ziel-Branch hat, und, für Organisationen, wenn der *ungemergte* Branch eines Teammitglieds sich mit Ihrem überschneidet. Ein Klick darauf liefert die Liste der

konfliktbehafteten Branches und die nützlichen Aktionen: jetzt rebasen oder mergen, Ihre Änderungen pushen, sie als Cloud Patch teilen, damit die andere Person hineinschauen kann, oder eine Zusammenfassung der Überschneidung kopieren, um sie in einen Chat einzufügen. Die Team View setzt noch einen Schritt früher

an und markiert eine Datei, wenn Sie und ein Teammitglied beide uncommittete Änderungen darin haben; die Dokumentation führt das als Funktion der Stufe Advanced.



Konflikterkennung verlegt die Warnung vom Merge-Zeitpunkt zum Bearbeitungszeitpunkt

Die Zeitleiste führt das Argument besser, als ich es könnte. Schlichtes Git meldet einen Konflikt am äußersten rechten Punkt, wenn beide Arbeiten fertig sind und jemand einen Teil seiner Arbeit wiederholen muss. Die beiden GitKraken-Funktionen verlegen die Warnung nach links, zu dem Moment, in dem die billigste Lösung

ein Gespräch von dreißig Sekunden ist: "Ich bin auch im Header, nimmst du die Navigation und ich das Logo?" In unserem montierten Tag leuchtet das Conflict-Prevention-Symbol auf, während der Codex-Agent noch an den Menüpreisen arbeitet. Ich pushe seinen Branch, schicke meiner Kollegin die Zusammenfassung der Überschneidung, und sie rebased ihren Pull Request, bevor meiner überhaupt fertig ist. Git sieht nie einen Konflikt.

Fairerweise ist das die Funktion, die davon abhängt, dass alle GitKraken nutzen. Die Prüfung gegen den Ziel-Branch funktioniert allein, aber die Überschneidung mit Teammitgliedern braucht das Team auf einem Organisationsplan mit installiertem Client, und das ist für ein gemischtes Team ein legitimer Grund, mit den Schultern zu zucken. Die Gewohnheit, die sie aufbaut, ist dagegen kostenlos: `main` früh und oft in langlebige Branches ziehen, und den anderen sagen, wenn Sie eine geteilte Datei anfassen wollen. Das Werkzeug automatisiert das Sagen.

> **Alternativen für den ganzen Tag**
> Keine der vier Stationen setzt GitKraken voraus, und ich wäre ein schlechter Führer, wenn ich etwas anderes behaupten würde. Der eingebaute Drei-Wege-Merge-Editor von VS Code ist kostenlos und gut.
>
> Gits eigenes `rerere` merkt sich, wie Sie einen Konflikt aufgelöst haben, und spielt es beim nächsten Mal wieder ab, was einen erstaunlichen Anteil wiederkehrender Konflikte auf rebasten Branches löst. Agenten selbst können Konflikte auflösen, wenn Sie sie darum bitten, und dann prüfen Sie einen Diff statt eines Merge-Tools. Und ein selbst gehostetes Gitea mit Pull Requests gibt einem kleinen Team
>
> Konfliktwarnungen zum PR-Zeitpunkt umsonst. GitKrakens Argument ist nicht, dass es etwas tut, was anderswo unmöglich wäre. Es ist, dass Graph, Agentenstatus, Composer und Merge-Tool eine Anwendung sind, die voneinander wissen, und dass das die Zahl der Orte verringert, an die ich schauen muss.

Abends: den Tag auf dem Graphen lesen

Bevor ich den Laptop zuklappe, tue ich, was ich seit meinem ersten Beitrag über den Graphen vor einem Jahr

tue, nur dass der Graph jetzt mehr zu sagen hat. Drei Worktrees sind weg, ihre Branches gemergt und über das Kontextmenü des Graphen gelöscht. Die Pull-Request-Pills in der Toolbar und in den Commit-Details zeigen, welche der heutigen Branches bereits gemergt sind, was mir die Frage "Ist der hier fertig?" erspart, die mich früher einen Ausflug zu GitHub gekostet hat. Seit 12.6 zeigt die Auswahl einer Folge aufeinanderfolgender Commits ihren kombinierten Diff im rechten Panel, sodass die vier Commits, die Commit Composer aus dem Bestellungs-Refactoring gemacht hat, als eine Änderung gelesen werden können, wenn das die bessere Sicht ist, und als vier, wenn nicht.

Und wenn etwas falsch aussieht, ist der Graph immer noch der Ort, an dem ich anfange. Der Forensik-Beitrag

ging vier klassische Fehler durch, die ein neues Teammitglied auf einem Remote hinterlassen hatte, und jeder davon war auf dem Graphen sichtbar, bevor er in einem Log verständlich war. Der Unterschied 2026 ist, dass der Verdächtige ein Agent sein kann. Ein Agent, der von der falschen Basis abgezweigt oder in einen bereits gemergten Branch committet hat, hinterlässt genau dieselben Formen wie ein Mensch. Dem Graphen ist egal, wer den Commit gemacht hat. Das ist, mehr oder weniger, der ganze Punkt.

Das Fazit

Die Aufgabe eines Git-Clients hat sich geändert, als das Tippen billig wurde. Aus meiner Sicht ist GitKraken Desktops beste Funktion 2026 kein einzelner KI-Knopf; es ist, dass der Commit-Graph zu dem einen Ort geworden ist, an dem der gegenwärtige Zustand von drei Agenten, zwei Teammitgliedern und meinem eigenen Branch auf einmal sichtbar ist, und dass die KI-Funktionen an diesem Bild hängen, statt es zu ersetzen. Agent-Sessions bringen jeden Worktree auf den Graphen. Commit Composer macht aus einem Haufen Agentenausgabe Commits, die ein Reviewer lesen kann, mit intakten Autoren. Die KI-Konfliktauflösung schreibt einen ersten Entwurf und sagt Ihnen, wie sicher sie ist, was genau so lange nützlich ist, wie Sie sich erinnern,

worüber sie sicher ist. Die Konflikterkennung verlegt die Warnung in den einen Moment, in dem es billig ist zu handeln.

Jede dieser Funktionen ist Preview, hinter einem bezahlten Plan oder davon abhängig, dass das Team dasselbe Werkzeug nutzt, und ich habe das an jeder Station gesagt. Der Open-Source-Weg über Worktrees, lazygit, den Merge-Editor von VS Code und `rerere` deckt dasselbe Terrain mit mehr Schritten ab. Sehen Ihre Repositories einen Agenten pro Woche, sind diese Schritte in Ordnung. Sehen sie drei pro Tag, ist die Integration das, wofür ich zahlen würde.

> 2024 zeigte mir der Commit-Graph die Vergangenheit. 2026 zeigt er mir die Gegenwart, und die Gegenwart hat mehr Autoren als früher.

Ein Beitrag zur Agentenseite dieses Bildes steht auf meiner Liste: was der GitKraken-MCP-Server einen Agenten über ein Repository fragen lässt, und wo ein Agent meiner Meinung nach weiterhin mich fragen sollte.

INDEPENDENT · AD-FREE · READER-SUPPORTED

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

Support on Ko-fi —