Claude Code richtig einrichten — in 5 Minuten

Für alle, die Claude Code gerade installiert haben und nicht wissen, wo man anfängt.

UPRO Capital · white-label Kurs

Inhalt

1. Schritt 1: Claude Code sagen, wie du arbeitest

1.1 Die CLAUDE.md — dein erstes Setup

Claude Code vergisst nach jeder Session alles. Ohne eine CLAUDE.md startet es jedes Mal neu — ohne zu wissen, welchen Tech-Stack du nutzt, welchen Stil du bevorzugst oder wie du deployst. Das Ergebnis: generische Antworten, falsche Annahmen, viel Nacharbeit.

Die CLAUDE.md ist eine einfache Textdatei, die Claude Code automatisch liest bevor es dir antwortet. Du schreibst einmal rein: dein Stack, dein Paket-Manager, dein Deploy-Befehl, deine Regeln. Ab da kennt Claude Code deinen Kontext — in jeder Session.

Es gibt zwei Versionen: Eine globale unter `~/.claude/CLAUDE.md` (gilt für alle Projekte) und eine projektspezifische direkt im Projektordner (gilt nur für dieses Projekt). Beide werden automatisch zusammengeführt.

1. 1. Öffne dein Terminal.
2. 2. Prüfe ob die globale Datei existiert: ls ~/.claude/CLAUDE.md
3. 3. Falls nicht vorhanden, erstelle sie: touch ~/.claude/CLAUDE.md
4. 4. Öffne sie in einem Editor und trage deine Grundregeln ein (z.B. Sprache, Coding-Stil, was du nie willst).
5. 5. Gehe dann in deinen Projektordner und erstelle dort ebenfalls eine CLAUDE.md: touch CLAUDE.md
6. 6. Trage dort ein: Stack, Package-Manager, Dev-Befehl, Build-Befehl, Deploy-Befehl.
7. 7. Starte Claude Code und frage im Chat: 'Was weißt du über dieses Projekt?' — Claude soll die Infos aus deiner Datei nennen.

PROMPT

```
Was weißt du über dieses Projekt laut deiner Konfiguration? Nenne Stack,
Startbefehl und Deploy-Befehl.
```

PROMPT

```
Prüfe meine CLAUDE.md. Was fehlt damit du das Projekt vollständig kennst?
```

ÜBUNG

Öffne dein Projekt und erstelle eine CLAUDE.md im Projektordner. Trage mindestens 4 Dinge ein: (1) deinen Stack, (2) den Befehl um das Projekt zu starten, (3) den Deploy-Befehl, (4) eine Coding-Regel. Frage danach Claude Code: 'Wie starte ich das Projekt?' — die Antwort soll aus deiner Datei kommen.

▶ Lösung

1.2 CLAUDE.md mit /init automatisch erstellen lassen

Du musst die CLAUDE.md nicht komplett selbst schreiben. Claude Code kann das für dich übernehmen — mit dem Befehl `/init` scannt es dein Projekt und schreibt eine erste Version automatisch.

Das Ergebnis ist ein guter Startpunkt, aber oft noch zu allgemein. Claude erkennt den Stack, aber nicht deine persönlichen Regeln — die musst du danach manuell ergänzen. Das dauert 2–3 Minuten.

So geht's: Tippe `/init` im Claude Code Chat. Claude scannt den Projektordner, liest vorhandene Config-Dateien (package.json, tsconfig.json, wrangler.toml etc.) und schreibt die CLAUDE.md. Danach öffnest du die Datei und fügst deine eigenen Regeln ein.

1. 1. Öffne Claude Code in deinem Projektordner.
2. 2. Tippe im Chat: /init
3. 3. Warte bis Claude Code die CLAUDE.md geschrieben hat.
4. 4. Öffne die erzeugte CLAUDE.md und lies sie durch.
5. 5. Ergänze fehlende Dinge: persönliche Coding-Regeln, spezifische Deploy-Befehle, Dinge die Claude nie tun soll.
6. 6. Speichere die Datei und teste mit: 'Was sind die Coding-Regeln für dieses Projekt?'

PROMPT

```
/init
```

PROMPT

```
Lies die CLAUDE.md und ergänze fehlende Punkte: Was brauchst du noch um
wirklich produktiv in diesem Projekt zu sein?
```

ÜBUNG

Tippe /init in Claude Code und lass die CLAUDE.md automatisch generieren. Ergänze danach zwei eigene Regeln die nicht drin stehen — zum Beispiel eine Stilregel und eine Regel was Claude nie tun soll. Frage danach: 'Welche Coding-Regeln gelten?' — beide eigenen Regeln sollen in der Antwort auftauchen.

▶ Lösung

2. Schritt 2: Das richtige Modell wählen

2.1 Modell wechseln — wann und wie

Claude Code startet standardmäßig mit einem mittleren Modell. Das reicht für einfache Aufgaben, aber für komplexere Probleme — Architektur planen, schwierige Bugs finden, Code komplett refactorn — gibt es stärkere Optionen.

Das Modell lässt sich direkt im Chat wechseln, ohne Claude Code neu zu starten. Du tippst einfach einen Slash-Command. Der Wechsel gilt sofort für die nächste Antwort.

Faustregel: Kleines Modell für schnelle Fragen und einfache Änderungen. Großes Modell für alles was wirklich denken erfordert. Wenn du auf API-Billing bist (nicht Flatrate), kostet das große Modell deutlich mehr — nutze es gezielt, nicht dauerhaft.

1. 1. Öffne Claude Code und starte einen Chat.
2. 2. Tippe /model um die Liste verfügbarer Modelle zu sehen.
3. 3. Wähle das gewünschte Modell aus der Liste.
4. 4. Stelle eine komplexe Frage und vergleiche die Antwortqualität mit dem Standard-Modell.
5. 5. Wechsle für einfache Folge-Fragen wieder zurück auf das kleinere Modell.

PROMPT

```
/model
```

PROMPT

```
Erkläre mir die Architektur dieses Projekts in 5 Sätzen – was macht was, wie hängt es zusammen?
```

ÜBUNG

Tippe /model im Chat und schau dir die Liste an. Wechsle auf das stärkste verfügbare Modell. Stelle dann eine konkrete Frage zu deinem Projekt — zum Beispiel: 'Erkläre mir die Architektur dieser Codebasis in 5 Sätzen.' Vergleiche die Antwort mit der, die du vom Standard-Modell bekämst.

▶ Lösung

2.2 Modell-Wechsel automatisch machen

Wenn du weißt, dass du für bestimmte Aufgaben immer das stärkste Modell willst, kannst du das in der CLAUDE.md festhalten. Du schreibst dort eine Regel rein — zum Beispiel: 'Wenn ich /ultrareview sage, wechsle auf das stärkste Modell.' Claude Code liest diese Regel und handelt entsprechend.

Das spart Zeit, weil du nicht jedes Mal manuell wechseln musst. Gleichzeitig bleibt der Wechsel kontrolliert — er passiert nur wenn du ein bestimmtes Wort oder einen Befehl nutzt, nicht automatisch bei jeder Anfrage.

Wichtig falls du API-Billing nutzt: Das stärkste Modell kostet ca. 5× mehr als das Standard-Modell. Auf einem Flatrate-Plan ist das egal — auf API-Billing solltest du den Auto-Wechsel nur für wirklich wichtige Aufgaben konfigurieren.

1. 1. Öffne deine CLAUDE.md im Projektordner.
2. 2. Füge einen Abschnitt 'Modell-Regeln' ein.
3. 3. Schreibe rein: 'Wenn ich /ultrareview sage, nutze das stärkste verfügbare Modell mit maximaler Denktiefe.'
4. 4. Speichere die Datei.
5. 5. Starte einen neuen Chat in Claude Code.
6. 6. Tippe /ultrareview gefolgt von einer komplexen Frage — Claude soll automatisch auf das stärkste Modell wechseln.

PROMPT

```
Welche Modell-Regeln gelten laut meiner CLAUDE.md?
```

PROMPT

```
deep: Was sind die 3 größten Probleme in meiner aktuellen Codebase – sei ehrlich.
```

ÜBUNG

Füge in deine CLAUDE.md eine Modell-Regel ein. Wähle ein eigenes Trigger-Wort (z.B. 'deep:' oder 'think:'). Schreibe die Regel so: 'Wenn ich [dein Wort] vor einer Frage schreibe, nutze das stärkste Modell.' Teste es mit: 'deep: Was sind die 3 größten Risiken in meiner aktuellen Codebase?'

▶ Lösung

3. Schritt 3: Die 4 wichtigsten Skills installieren

3.1 Was Skills sind und wo sie liegen

Skills sind einfache Textdateien — keine Programme, keine Installationen, keine Abhängigkeiten. Du legst eine Datei namens SKILL.md in einen bestimmten Ordner, und Claude Code liest sie automatisch bei jedem Start.

Der Ordner für globale Skills (gelten für alle Projekte): `~/.claude/skills/dein-skill-name/SKILL.md`. Für projektspezifische Skills: `./.claude/skills/dein-skill-name/SKILL.md` im Projektordner.

So einfach ist es: Ordner erstellen, SKILL.md reinlegen, fertig. Claude Code erkennt neue Skills ohne Neustart oder Konfiguration. Du kannst den Inhalt jederzeit direkt in der Datei bearbeiten — der Skill ändert sich sofort.

1. 1. Öffne dein Terminal.
2. 2. Erstelle den Skills-Ordner falls er nicht existiert: mkdir -p ~/.claude/skills/
3. 3. Erstelle einen Testskill-Ordner: mkdir -p ~/.claude/skills/mein-test-skill/
4. 4. Erstelle die SKILL.md: touch ~/.claude/skills/mein-test-skill/SKILL.md
5. 5. Schreibe in die Datei: 'Wenn ich /test sage, antworte mit: Skill funktioniert!'
6. 6. Starte Claude Code neu und tippe /test — Claude soll mit 'Skill funktioniert!' antworten.
7. 7. Prüfe installierte Skills: ls ~/.claude/skills/

PROMPT

```
Welche Skills hast du aktuell geladen? Liste alle auf.
```

PROMPT

```
/check
```

ÜBUNG

Erstelle deinen ersten eigenen Skill. Der Skill soll auf das Wort /check reagieren und dir dann 5 Fragen stellen, die du dir stellen solltest bevor du Code deployst. Erstelle die SKILL.md, lege sie ab, und teste /check in Claude Code.

▶ Lösung

3.2 Superpowers: Erst planen, dann bauen

Das größte Problem mit KI-generierten Code: Man tippt 'Bau mir Feature X' und bekommt sofort Code — ohne Plan, ohne Rückfragen, ohne zu wissen ob das überhaupt der richtige Weg ist. Dann hat man 200 Zeilen Code die zwar laufen, aber das falsche Problem lösen.

Der Superpowers-Skill dreht das um. Du tippst `/brainstorm` und beschreibst was du bauen willst. Statt sofort Code zu schreiben, stellt Claude Rückfragen, schreibt einen Plan (PLAN.md) und wartet auf deine Freigabe. Erst wenn du sagst 'Ja, bau das so' — wird Code geschrieben.

Das klingt langsamer, ist es aber nicht. Ein schlechter Plan kostet dich Stunden an Reparatur-Code. Ein guter Plan bedeutet: erster Code läuft oft direkt.

1. 1. Installiere den Superpowers-Skill — tippe in Claude Code: 'Installiere den Superpowers Skill von GitHub. Suche den offiziellen Install-Command und führe ihn aus.'
2. 2. Warte auf die Bestätigung der Installation.
3. 3. Teste die Installation: ls ~/.claude/skills/superpowers/
4. 4. Starte einen neuen Chat und tippe: /brainstorm Ich will eine Login-Seite mit Email und Passwort bauen.
5. 5. Lies den generierten Plan durch — stimmt er? Fehlt etwas?
6. 6. Antworte mit 'Ja, bau das so' oder 'Ändere Punkt 2 zu...'
7. 7. Warte bis Claude Code den Code schreibt und committet.

PROMPT

```
/brainstorm
```

PROMPT

```
Führe den Plan aus. Schreibe Tests zuerst, dann die Implementierung.
Committe jeden Schritt einzeln.
```

ÜBUNG

Nutze /brainstorm für ein kleines Feature in deinem Projekt — zum Beispiel eine einfache Such-Funktion oder ein Formular. Lass Claude den Plan schreiben. Lies den Plan durch und ändere mindestens einen Punkt bevor du sagst 'Bau das so.' Das zeigt dir, ob der Plan wirklich deinen Vorstellungen entspricht.

▶ Lösung

3.3 Caveman: Kürzere Antworten, bessere Sessions

Je länger eine Session läuft, desto mehr Kontext muss Claude Code im Blick behalten. Irgendwann ist der Kontext so voll, dass die Antworten schlechter werden — Claude wiederholt sich, verliert den Faden oder schreibt zu allgemein.

Der Caveman-Skill löst das: Er reduziert die Ausgabe-Tokens um 65–75%, ohne dass die Antwort inhaltlich schlechter wird. Claude antwortet knapper, aber präziser. Keine ausgeschriebenen Erklärungen wenn du schon Entwickler bist. Kein 'Hier ist eine mögliche Lösung...' — direkt die Lösung.

Wann du ihn brauchst: Bei langen Sessions (mehr als 30 Nachrichten), wenn Claude anfängt zu labern, oder wenn du einfach schnellere Antworten willst.

1. 1. Installiere Caveman: Tippe in Claude Code: 'Installiere den Caveman Skill. Suche den offiziellen Install-Command.'
2. 2. Nach der Installation: Starte einen neuen Chat.
3. 3. Aktiviere Caveman: Tippe /caveman (oder das festgelegte Trigger-Wort).
4. 4. Stelle eine normale Frage und beobachte die Antwortlänge.
5. 5. Deaktiviere Caveman und stelle dieselbe Frage — vergleiche.
6. 6. Nutze Caveman standardmäßig bei allen Sessions die länger als 20 Nachrichten gehen.

PROMPT

```
/caveman
```

PROMPT

```
Erkläre mir diese Funktion in maximal 3 Sätzen:
```

ÜBUNG

Aktiviere Caveman und stelle Claude Code 5 typische Entwickler-Fragen — zum Beispiel 'Wie baue ich ein debounce in React?'. Notiere die Antwortlänge. Deaktiviere dann Caveman und stelle dieselben Fragen. Vergleiche: Fehlt inhaltlich etwas in den kürzeren Antworten?

▶ Lösung

3.4 LLM Council: Mehrere Perspektiven für schwierige Entscheidungen

Wenn du Claude Code eine schwierige Frage stellst — 'Soll ich PostgreSQL oder SQLite nehmen?' oder 'Welche Architektur für dieses Feature?' — bekommst du eine Antwort aus einem Blickwinkel. Die Antwort klingt überzeugend, aber du weißt nicht ob sie alle Seiten beleuchtet hat.

Der LLM Council-Skill startet 5 unabhängige Analysen parallel. Jede Analyse kommt aus einer anderen Perspektive: einer sucht aktiv den Fehler im Plan, einer denkt von den Grundannahmen zurück, einer findet übersehene Vorteile, einer schaut von außen drauf, einer fragt nur 'Was machst du konkret Montag?' — danach fasst Claude alles in einem Fazit zusammen.

Wann sinnvoll: Bei echten Abwägungen wo ein Fehler teuer ist — Technologie-Wahl, Pricing, Produkt-Entscheidungen. Nicht bei einfachen Coding-Fragen.

1. 1. Installiere den LLM Council Skill: Tippe in Claude Code: 'Installiere den LLM Council Skill. Suche den offiziellen Install-Command.'
2. 2. Starte einen neuen Chat nach der Installation.
3. 3. Stelle eine schwierige Entscheidungs-Frage mit dem Trigger: 'council this: Soll ich für mein Projekt [Option A] oder [Option B] wählen?'
4. 4. Warte auf das Fazit — es zeigt wo alle 5 Perspektiven übereinstimmen und wo sie sich widersprechen.
5. 5. Lies den Teil 'Wo sich der Council widerspricht' besonders aufmerksam — das sind die echten Abwägungen.
6. 6. Nutze den konkreten nächsten Schritt aus dem Fazit.

PROMPT

```
council this: [Deine schwierige Entscheidung hier]
```

PROMPT

```
pressure-test this: [Dein Plan hier] – was kann daran schiefgehen?
```

ÜBUNG

Nutze den Council für eine echte Entscheidung aus deinem Projekt. Zum Beispiel: 'council this: Soll ich meine Authentifizierung selbst bauen oder eine fertige Lösung wie Clerk oder Auth.js nutzen?' Lies das Fazit und schreibe auf: Was hat der Council gesagt das du vorher nicht bedacht hattest?

▶ Lösung

4. Bonus: 4 Extra-Skills für besseren Code

4.1 UI UX Pro Max: Kein generisches KI-Design mehr

Wenn du Claude Code ohne diesen Skill bittest eine UI zu bauen, bekommst du Generic-KI-Design: blau-weiß, Times New Roman, Standard-Buttons. Es funktioniert, sieht aber nicht gut aus.

Der UI UX Pro Max Skill gibt Claude Code über 50 Styles, 161 Farbpaletten, 57 Font-Kombinationen und 99 UX-Regeln. Der Skill aktiviert sich automatisch wenn du etwas UI-bezogenes baust — du musst nichts extra tippen. Claude fragt dich nach deinem gewünschten Stil, dann baut es entsprechend.

Das Ergebnis ist kein perfektes Design, aber deutlich besser als der Standard-Output. Gut genug für Prototypen und MVPs.

1. 1. Installiere UI UX Pro Max: Tippe in Claude Code: 'Installiere den UI UX Pro Max Skill. Suche den offiziellen Install-Command.'
2. 2. Warte auf die Bestätigung.
3. 3. Teste sofort: 'Baue mir eine Login-Seite. Modern, dark theme.'
4. 4. Der Skill fragt nach deinem bevorzugten Stil — wähle einen aus.
5. 5. Schau dir das Ergebnis an: Deutlich strukturierter als ohne den Skill.

PROMPT

```
Baue mir eine Card-Komponente für mein Projekt. Nutze den UI UX Pro Max
Skill für das Design.
```

PROMPT

```
Welche Design-Stile stehen zur Verfügung?
```

ÜBUNG

Installiere UI UX Pro Max und lass eine einfache Karte (Card-Komponente) bauen — mit Titel, Text und einem Button. Vergleiche das Ergebnis mit dem was Claude Code ohne den Skill gebaut hätte. Beschreibe den Unterschied in einem Satz.

▶ Lösung

4.2 Context7: Aktuelle Doku statt veraltetes Wissen

Claude Code wurde mit Trainingsdaten aus der Vergangenheit trainiert. Das bedeutet: Wenn du nach Next.js 15 fragst, könnte Claude Code dir Next.js 14 Syntax geben ohne es zu merken. Oder veraltete Tailwind-Klassen. Das findest du erst beim Build heraus — wenn nichts mehr funktioniert.

Context7 ist ein sogenannter MCP-Server — ein kleines Programm das Claude Code in Echtzeit mit aktueller Framework-Dokumentation verbindet. Wenn du mit Context7 nach 'Wie baue ich Server Actions in Next.js?' fragst, zieht Claude Code die aktuelle offizielle Doku und gibt dir die korrekte Antwort — nicht die von vor 2 Jahren.

Pflicht für jeden der mit React, Next.js, Tailwind, Supabase oder anderen schnell-ändernden Frameworks arbeitet.

1. 1. Installiere Context7 via npm: npx -y @upstash/context7-mcp (oder schau die aktuelle Install-Anleitung auf context7.com nach).
2. 2. Füge den MCP-Server in Claude Code ein — öffne die Claude Code Einstellungen (Settings → MCP Servers) und trage Context7 ein.
3. 3. Starte Claude Code neu.
4. 4. Teste mit: 'Wie baue ich eine Server Action in Next.js 15? Nutze use context7.'
5. 5. Claude soll jetzt mit einem Hinweis auf die aktuelle Doku-Version antworten.

PROMPT

```
Wie baue ich [dein Feature] in [dein Framework]? use context7
```

PROMPT

```
Was hat sich in [Framework] zwischen Version X und Y geändert? use context7
```

ÜBUNG

Installiere Context7 und stelle danach eine Framework-Frage zu dem Tool das du am häufigsten nutzt — z.B. 'Wie richte ich Middleware in Next.js 15 ein? use context7.' Prüfe ob Claude die aktuelle API-Syntax nutzt indem du die Antwort mit der offiziellen Doku vergleichst.

▶ Lösung

4.3 /security-review — immer vor dem Deploy

Jeder Code hat Lücken. Hardcodierte API-Keys, fehlende Input-Validierung, SQL-Queries die man manipulieren kann — das sind keine seltenen Fehler, das sind typische KI-generierte Fehler weil Claude Code auf 'es funktioniert' optimiert, nicht auf 'es ist sicher'.

Der Befehl `/security-review` ist in Claude Code eingebaut — du brauchst nichts installieren. Du tippst ihn, Claude Code scannt deinen letzten Code-Block oder das aktuelle File und listet konkrete Probleme mit konkreten Fixes auf. Kein allgemeines Sicherheits-Gelaber — spezifische Zeile, spezifisches Problem, spezifischer Fix.

Mach das zur Gewohnheit: Kein Deploy ohne `/security-review`. Besonders wichtig wenn du Code hast der Nutzer-Input verarbeitet oder auf eine Datenbank zugreift.

1. 1. Öffne Claude Code mit einem fertigen Feature das du gleich deployen würdest.
2. 2. Tippe: /security-review
3. 3. Lies den Output — Claude listet Probleme nach Schwere: CRITICAL, HIGH, MEDIUM, LOW.
4. 4. Fixe zuerst alle CRITICAL und HIGH Probleme.
5. 5. MEDIUM und LOW: Entscheide selbst ob relevant.
6. 6. Führe /security-review nach den Fixes nochmal aus — CRITICAL und HIGH sollten weg sein.
7. 7. Jetzt deployen.

PROMPT

```
/security-review
```

PROMPT

```
Fixe alle CRITICAL und HIGH Probleme aus dem security-review und erkläre
kurz was du geändert hast.
```

ÜBUNG

Nimm einen beliebigen Code-Block aus deinem Projekt der Nutzer-Input verarbeitet (Formular, API-Endpoint, Datenbank-Query). Kopiere ihn in den Chat und tippe /security-review. Notiere das erste gefundene Problem und setze den vorgeschlagenen Fix direkt um.

▶ Lösung

5. Alles zusammen: So arbeitest du ab jetzt

5.1 Alle Skills auf einmal installieren

Du kannst alle Skills einzeln installieren — oder du gibst Claude Code einen Prompt und es erledigt alles in einem Rutsch. Claude sucht für jeden Skill den aktuellen offiziellen Install-Command, führt ihn aus und meldet Fehler wenn etwas nicht klappt.

Das dauert ca. 3–5 Minuten. Danach hast du alle Skills und Context7 fertig eingerichtet. Falls ein einzelner Skill fehlschlägt, zeigt dir Claude Code den Fehler und den Fix.

Kopiere einfach den Prompt aus der Übung — fertig.

1. 1. Öffne Claude Code.
2. 2. Kopiere den Massen-Install-Prompt aus der Übung unten.
3. 3. Schicke ihn ab.
4. 4. Warte — Claude Code arbeitet sich durch alle Skills durch.
5. 5. Am Ende: Prüfe die Zusammenfassung. Welche Skills sind installiert, welche haben Fehler gemeldet?
6. 6. Fehlgeschlagene Skills: Tippe 'Installiere [Skill-Name] nochmal und zeig mir den Fehler.'
7. 7. Verifiziere: ls ~/.claude/skills/

PROMPT

```
Installiere diese Skills der Reihe nach. Suche für jeden den offiziellen
aktuellen Install-Command, führe ihn aus und berichte ob es funktioniert
hat:
1. Superpowers Skill
2. Caveman Skill
3. LLM Council Skill
4. UI UX Pro Max Skill
5. Context7 MCP Server

Am Ende: Liste alle erfolgreich installierten Skills auf.
```

PROMPT

```
Welche Skills sind aktuell installiert und geladen? Zeige mir ls
~/.claude/skills/
```

ÜBUNG

Kopiere diesen Prompt und schicke ihn in Claude Code: 'Installiere diese Skills der Reihe nach. Suche für jeden den offiziellen aktuellen Install-Command, führe ihn aus und berichte ob es funktioniert hat: 1. Superpowers Skill 2. Caveman Skill 3. LLM Council Skill 4. UI UX Pro Max Skill 5. Context7 MCP Server Am Ende: Liste alle erfolgreich installierten Skills auf und zeige mir eventuelle Fehler.'

▶ Lösung

5.2 Dein neuer Workflow für jedes Feature

Alles zusammen ergibt einen klaren Ablauf für jedes neue Feature — von der Idee bis zum Deploy. Nicht jeder Schritt ist immer nötig, aber das ist die Reihenfolge die funktioniert.

Plannen mit /brainstorm, bauen mit TDD, Design prüfen mit UI UX Pro Max (automatisch), Code reviewen mit /ultrareview, Sicherheit prüfen mit /security-review, deployen. Bei schwierigen Entscheidungen zwischendurch: 'council this:' einwerfen.

Der entscheidende Unterschied zum alten Workflow: Du baust nicht mehr Code und hoffst dass er gut ist. Du planst, prüfst und verbesserst in klaren Schritten. Das klingt langsamer, spart dir aber die stundenlange Debugging-Session am nächsten Tag.

1. 1. PLANEN: Tippe /brainstorm und beschreibe dein Feature. Beantworte die Rückfragen von Claude Code. Lies den fertigen Plan.
2. 2. ENTSCHEIDEN (falls unsicher): Tippe 'council this:' gefolgt von der größten offenen Frage aus dem Plan.
3. 3. BAUEN: Sage 'Führe den Plan aus. Schreibe Tests zuerst, dann den Code. Committe jeden Schritt einzeln.'
4. 4. REVIEWEN: Tippe /ultrareview sobald der Code fertig ist.
5. 5. SICHERHEIT: Tippe /security-review und fixe alle CRITICAL und HIGH Probleme.
6. 6. DEPLOYEN: Deploye mit dem Befehl aus deiner CLAUDE.md.

PROMPT

```
/brainstorm Ich will folgendes Feature bauen: [beschreibe dein Feature]
```

PROMPT

```
Führe den Plan aus. Schreibe Tests zuerst, dann die Implementierung.
Committe jeden Schritt einzeln mit einer aussagekräftigen Commit-Message.
```

PROMPT

```
/ultrareview
```

PROMPT

```
/security-review
```

PROMPT

```
council this: [deine schwierigste Entscheidung im Projekt gerade]
```

ÜBUNG

Nimm ein kleines Feature das du in deinem Projekt bauen willst — nichts Riesiges, zum Beispiel ein neues Formular oder eine neue Seite. Führe den kompletten Workflow durch: /brainstorm → Plan reviewen → 'Führe den Plan aus' → /security-review. Notiere wieviel Zeit du für jeden Schritt brauchst.

▶ Lösung