Claude Code als täglicher Assistent: Setup Guide

Despot Bitschnau
arcneo GmbH, Berlin

Stand: September 2026

Zusammenfassung

Dieser Guide zeigt, wie ich Claude Code im Terminal als persönlichen Alltags-Agenten einsetze. Nicht zum Programmieren, sondern für Mails, Recherche, Notizen, Branding und Organisation. Er führt chronologisch durch das komplette Setup auf dem Mac, von der Terminal-Installation bis zu konkreten Use Cases, die du direkt übernehmen kannst. Wichtig zur Einordnung: Das hier ist bewusst **nicht** mein Setup für Coding-Agenten. Meine Coding-Setups enthalten zwar alles, was hier beschrieben ist, aber darüber hinaus noch deutlich mehr (eigene Projekt-Regeln, Test- und Verifikations-Pipelines, CI-Anbindung und weitere Werkzeuge). Dieser Guide beschreibt ausschließlich das Setup für Nicht-Coding-Aufgaben.

So nutzt du diesen Guide am besten: Sobald die Grundinstallation steht (Kapitel 2 und 3), gib dieses Dokument deinem eigenen Claude-Agenten im Terminal. Er soll es bei sich ablegen, Schritt für Schritt durcharbeiten und alles installieren und einrichten, was bei dir noch fehlt: Ordnerstruktur, CLAUDE.md, Vault, Connectors, Plugins. „Wie richte ich X ein?" funktioniert ab diesem Moment, du bestätigst nur noch. Aber: Lies den Guide trotzdem unbedingt selbst durch. Vieles hier ist keine Installations-, sondern Anwendungssache (Workflows, Gewohnheiten, Use Cases), und du solltest verstehen, was auf deinem Rechner passiert und was es Neues gibt. Das eine schließt das andere nicht aus: Während du liest, kann dein Agent parallel schon die ersten Sachen installieren.

---

1. Warum Claude Code im Terminal statt Chat im Browser?

Claude Code ist Claude mit Händen. Der Chat auf claude.ai kann dir Texte schreiben, aber Claude Code kann zusätzlich:

- Dateien auf deinem Rechner lesen und schreiben (z. B. deine Obsidian-Notizen)
- Programme ausführen, Webseiten durchsuchen, PDFs bauen
- Über Connectors direkt in Gmail, Google Kalender, Notion, Figma usw. arbeiten
- Sich per CLAUDE.md-Datei dauerhaft merken, wer du bist und wie du arbeitest

Der Effekt: Statt Antworten zu kopieren, erledigt der Agent Dinge. "Fasse meine Mails von heute zusammen und leg die Notiz in Obsidian ab" ist ein einziger Satz.

Und warum nicht Claude Cowork?

Anthropic bietet mit **Claude Cowork** eine Agenten-Variante direkt in der Claude-Desktop-App an, ohne Terminal, ohne Installation. Für den absoluten Einstieg ist das okay, aber für den täglichen Einsatz, wie ich ihn hier beschreibe, ist Claude Code die bessere Wahl:

- **Arbeitet direkt auf deinen echten Dateien.** Cowork läuft in einer abgeschotteten virtuellen Maschine, deine Dateien müssen erst in diese Sandbox hinein und Ergebnisse wieder heraus. Claude Code arbeitet direkt in deinem Dateisystem, also z. B. live in deinem Obsidian-Vault. Genau darauf baut dieses ganze Setup auf.
- **Erledigt Sachen im Hintergrund, ohne dich zu blockieren.** Ein Claude-Code-Auftrag läuft in seinem Terminal-Tab weiter, während du am Rechner ganz normal weiterarbeitest. Claude muss dafür nie deinen Bildschirm oder deine Programme übernehmen, es meldet sich einfach, wenn es fertig ist.
- **Plugins, Skills und eigene Befehle.** Claude Code lässt sich frei erweitern: Plugins installieren, eigene Slash-Commands und Hooks bauen, Berechtigungen fein einstellen. Cowork bringt nur einen festen, vorinstallierten Funktionsumfang mit.
- **Mehrere Instanzen parallel im Griff.** Mein Kern-Workflow, ein benannter und eingefärbter iTerm2-Tab pro Thema mit jeweils eigener Session, funktioniert nur im Terminal. So laufen problemlos drei, vier Agenten gleichzeitig, ohne dass man durcheinanderkommt.
- **Volle Konfigurierbarkeit.** CLAUDE.md-Dateien pro Ordner, eigene Einstellungen, Sprach- und Voice-Konfiguration, Statuszeile: alles anpassbar.

Kurz: Cowork ist der Mietwagen mit Automatik, Claude Code das eigene Auto. Wer nur einmal schnappen will, was der Agent kann, startet mit Cowork. Wer den hier beschriebenen Alltags-Workflow will, installiert Claude Code, und der Rest dieses Guides zeigt, dass das keine 15 Minuten dauert.

---

2. Schritt 1: Das richtige Terminal

Am Mac nimmst du **iTerm2**, nicht das vorinstallierte Terminal.app. Download: iterm2.com (kostenlos). Einfach die App in den Ordner "Programme" ziehen.

Warum iTerm2:

- **Tabs benennen, und zwar automatisch.** Ich habe meist mehrere Tabs parallel offen, und jeder trägt den Namen seines Themas. Den Namen setzt der Agent selbst per `/rename` (wie genau, steht in Kapitel 9). Damit der Name auch im Tab ankommt, braucht iTerm2 zwei Handgriffe. Erstens: keine Tab-Titel mehr von Hand vergeben. Ein manuell gesetzter Titel hat in iTerm2 immer Vorrang und blockiert jede Umbenennung aus Claude Code heraus; die Session heißt dann intern richtig, der Tab zeigt aber weiter den alten Namen. Wer noch Handtitel hat, löscht sie einmal: Rechtsklick auf den Tab, "Edit Tab Title", Feld leeren, bestätigen. Zweitens: in Settings, Profiles, General beim Feld "Title" nur "Session Name" ankreuzen und "Job" abwählen, sonst hängt hinter jedem Namen ein "(node)". Farben vergebe ich von Hand: Rechtsklick, "Edit Tab Color".

- **Native Benachrichtigungen.** Claude Code kann dich über iTerm2 benachrichtigen, wenn eine lange Aufgabe fertig ist und du gerade in einem anderen Fenster bist. Das passiert doppelt: ein kleines Pop-up rechts oben am Bildschirmrand und dazu ein kurzer Benachrichtigungston. Man verpasst also nichts, egal wo man gerade hinschaut.
- **Split Panes.** Cmd+D teilt das Fenster, praktisch wenn ein Agent arbeitet und du parallel etwas nachschauen willst.
- **Bilder per Ctrl+V einfügen.** Screenshots oder kopierte Bilder landen direkt im Prompt, wenn du im leeren Eingabefeld Ctrl+V drückst (nicht Cmd+V, das fügt nur Text ein). Damit das in iTerm2 funktioniert, muss iTerm2 Programmen den Zugriff auf die Zwischenablage erlauben, und das ist ab Werk aus. Einmalig in Settings, General, Selection den Haken bei "Applications in terminal may access clipboard" setzen, danach iTerm2 komplett beenden und neu öffnen (Cmd+Q, nicht nur das Fenster schließen). Ohne diesen Haken passiert bei Ctrl+V schlicht nichts, und man sucht den Fehler an der falschen Stelle.

Windows und Linux: Tabby. iTerm2 gibt es nur am Mac. Wer Windows oder Linux nutzt, nimmt **Tabby** (tabby.sh, kostenlos und quelloffen). Tabby kann genau das, worauf es hier ankommt: benannte und gefärbte Tabs, Split Panes, Bilder aus der Zwischenablage. Unter Windows öffnest du darin eine PowerShell-Session, dort läuft Claude Code nativ. Das vorinstallierte Windows Terminal tut es auch, aber mit Tabby bist du am nächsten an dem, was in diesem Guide beschrieben ist.

Typischer Anfängerfehler: Leute bleiben in Terminal.app und wundern sich, warum Benachrichtigungen und Farben nicht funktionieren.

---

3. Schritt 2: Claude Code installieren

Mac und Linux. Öffne iTerm2 und füge diese Zeile ein (Enter drücken):

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows. Öffne in Tabby eine PowerShell-Session und füge diese Zeile ein:

```
irm https://claude.ai/install.ps1 | iex
```

Unter Windows ist es also nicht der curl-Befehl, sondern `irm`. Wer die curl-Zeile aus einem Mac-Tutorial kopiert, bekommt eine Fehlermeldung und sucht danach an der falschen Stelle.

Beides installiert Claude Code nativ, ohne dass du vorher Node.js oder Homebrew brauchst. Danach das Terminal einmal komplett schließen und neu öffnen.

Wenn später "command not found" kommt (unter Windows: "wird nicht als Name eines Cmdlets erkannt"), wurde das Terminal nicht komplett neu gestartet. Nicht nur ein neues Tab öffnen, sondern das Programm beenden und neu starten; der Installer trägt den Pfad erst für neue Sitzungen ein.

Zuerst der Ordner, dann Claude

Bevor du Claude das erste Mal startest, legst du den Ordner an, in dem dein Assistent wohnen soll. Das wirkt wie eine Kleinigkeit und ist doch die wichtigste Weiche im ganzen Setup: Claude liest Identität, Regeln und Gedächtnis immer aus dem Ordner, in dem du ihn gestartet hast. Startest du ihn woanders, ist er ein leeres Blatt.

Gib dem Ordner den Namen deines Assistenten oder schlicht `Assistant`. Bei mir heißt der Ordner `Assistant` und der Agent darin Falco.

Mac und Linux:

```
mkdir ~/Assistant
cd ~/Assistant
claude
```

Windows (PowerShell):

```
mkdir $HOME\Assistant
cd $HOME\Assistant
claude
```

Ab jetzt gilt ohne Ausnahme: Claude wird in diesem Ordner gestartet, nirgends sonst.

Anmelden und Team wählen

Beim ersten Start von `claude` öffnet sich der Login im Browser:

1. Mit deinem **Claude-Konto** anmelden (das gleiche wie auf claude.ai). Du brauchst einen Pro-, Max- oder Team-Plan.
2. Achtung, zweiter klassischer Fehler: Beim Login gibt es zwei Wege, "Claude account" (Abo) und "Console account" (API mit Guthaben). Für den Alltag willst du **Claude account**, sonst zahlst du pro Anfrage statt über dein Abo.
3. Wenn du in mehreren Organisationen bist (privat + Firma): Nach dem Login das richtige **Team auswählen**. Falls du im falschen gelandet bist, in Claude `/login` eingeben und neu anmelden, dort lässt sich die Organisation wechseln.

Kurzer Funktionstest: `claude` starten und etwas Einfaches fragen ("Welche Dateien liegen auf meinem Desktop?"). Wenn Claude antwortet und um Erlaubnis für Aktionen fragt, läuft alles.

Ein eigener Startbefehl

`cd ~/Assistant && claude` tippt man ein paar Tage lang brav und startet Claude dann aus Bequemlichkeit doch irgendwo anders. Deshalb legst du dir gleich einen eigenen Befehl an, der beides in einem erledigt: in den Ordner wechseln und Claude starten. Er heißt am besten wie dein Assistent, bei mir `falco`.

Mac und Linux (zsh):

```
echo 'falco() { ( cd "$HOME/Assistant" && claude "$@" ) }' >> ~/.zshrc
source ~/.zshrc
```

Windows (PowerShell): einmalig das Profil anlegen und öffnen,

```
if (!(Test-Path $PROFILE)) { New-Item -ItemType File -Path $PROFILE -Force }
notepad $PROFILE
```

dann diese Zeile hineinschreiben, speichern und das Terminal neu starten:

```
function falco { Push-Location $HOME\Assistant; claude @args; Pop-Location }
```

Ab jetzt reicht `falco`, egal in welchem Ordner du gerade stehst. Der Befehl wächst später mit: Bei mir hängt er zusätzlich zwei Google-Drive-Ordner an, auf die Claude zugreifen darf, über `--add-dir`.

```
falco() {
  ( cd "$HOME/Assistant" && claude --add-dir "$HOME/Pfad/zu/Ordner1" --add-d
}
```

---

4. Schritt 3: Grundkonfiguration

In Claude Code tippst du Befehle, die mit `/` beginnen. Die wichtigsten fürs Setup:

/config: die Zentrale

`/config` öffnet die Einstellungen. Dort stellst du ein:

- **Theme:** Dunkles oder helles Farbschema, je nach Terminal-Optik. Einfach durchprobieren.
- **Sprache:** Auf **Deutsch** stellen, dann antwortet Claude immer auf Deutsch, egal wie du fragst. (In der Konfigdatei `~/.claude/settings.json` steht dann `"language": "German"`.)
- **Benachrichtigungen:** Auf iTerm2 stellen, damit du gepingt wirst, wenn eine Aufgabe fertig ist, als kleines Pop-up rechts oben plus kurzem Ton (siehe Kapitel 2).

Modell und Effort: welche Stufe für was

Zwei Regler bestimmen, wie stark der Agent arbeitet, und die meisten kennen nur einen davon.

Regler 1: das Modell. Mit `/model` (oder in `/config`) wählst du zwischen den Claude-Modellen. Grob: **Sonnet** ist der schnelle Arbeiter, **Opus** der starke Allrounder, **Fable** die höchste Stufe für Aufgaben, bei denen ein Fehler richtig weh tut. Dazu gibt es `/fast`, das schaltet Opus in einen Modus mit schnellerer Ausgabe, ohne auf ein kleineres Modell herunterzuschalten.

Regler 2: der Effort. Unabhängig vom Modell stellst du ein, wie viel Denk-Budget der Agent pro Antwort bekommt. Das ist im Alltag oft der größere Hebel als der Modellwechsel: Ein mittleres Modell mit mehr Effort schlägt regelmäßig ein starkes Modell, das im Schnelldurchlauf antwortet.

So fahre ich das täglich, und zwar praktisch immer auf **Medium Effort**:

- **Sonnet, Medium:** Routine. Mails formulieren, Listen bauen, Dateien einsortieren, Termine, Kleinkram. Da braucht niemand ein Schwergewicht.
- **Opus, Medium:** mein Standard für alles, was ein bisschen Denken braucht. Recherche zu einem Wettbewerber und danach direkt umsetzen, Analysen, mehrstufige Aufträge. 90 Prozent meines Tages läuft hier.
- **Fable, Medium:** wenn es zählt. Sicherheitslücken suchen, ein Dokument final durchsehen, bevor es rausgeht, ein Vertragstext, ein heikler Investoren-Text. Immer dann, wenn ein übersehener Fehler teuer wird.
- **Fable plus ultrathink oder ultracode:** Planung und Architektur. Wie bauen wir diese neue App auf, welche Struktur, welche Aufgaben, welches Security-Konzept. Das sind die Momente, in denen sich die höchste Eskalationsstufe wirklich rechnet (mehr dazu in Kapitel 9).

Die Faustregel, die ich mir daraus gebaut habe: **Erst den Effort hochdrehen, dann das Modell wechseln.** Und die Stufe nach den Fehlerkosten wählen, nicht nach dem Bauchgefühl. Eine Liste, die falsch ist, korrigiert man in zehn Sekunden. Eine Mail an einen Investor oder eine übersehene Sicherheitslücke nicht.

Was man dabei im Blick behalten muss: Die hohen Stufen kosten Kontingent, `ultracode` mit seinem Schwarm an Subagenten deutlich mehr als alles andere. Wer den ganzen Tag auf Maximum fährt, steht am Nachmittag ohne Budget da.

Und ein Punkt fürs Team: Wenn ihr euch eigene Skills baut (Kapitel 6), schreibt dazu, **für welche Modellstufe der Skill gebaut und getestet ist.** Alles mit Außenwirkung, also Mails, LinkedIn-Posts, Kundendokumente, ist sonst schwer reproduzierbar, weil dasselbe Skript auf Sonnet anders herauskommt als auf Fable. Bei uns steht das in jedem Skill im Kopf: Unsere Team-Skills sind auf **Fable 5.0 mit Medium Effort** getestet und laufen ebenso sauber unter **Opus 4.8**. Wer sie auf einer kleineren Stufe fährt, bekommt Ergebnisse, aber nicht zwingend die, auf die der Skill geeicht wurde.

Voice: mit Claude sprechen

Das hat meinen Workflow am stärksten verändert: Ich diktiere fast alles, statt zu tippen. In `/config` den **Voice Mode** aktivieren. Ich nutze den "Hold"-Modus: Taste gedrückt halten, sprechen, loslassen, fertig. Diktieren ist gerade bei längeren Aufträgen ("Schreib eine Mail an X, erwähne Y, Ton eher locker...") viel schneller als Tippen. In Kombination mit Sprache = Deutsch fühlt sich das an wie ein Gespräch mit einem Assistenten.

Wer noch einen Schritt weiter will, nimmt **SuperWhisper** (superwhisper.com): ein eigenständiges Diktier-Tool für den Mac auf Basis der Whisper-Modelle. Die Erkennung ist noch mal einen Tick besser als der eingebaute Voice Mode, gerade bei Fachbegriffen und gemischtem Deutsch/Englisch. Zwei Dinge dazu: Die **Free-Version ist limitiert** (die besseren Modelle und der volle Funktionsumfang stecken im Abo), und der eigentliche Trumpf ist, dass es **systemweit** funktioniert, nicht nur in Claude. Einmal eingerichtet, diktierst du damit überall: Mails, Notizen, Nachrichten, jedes Textfeld. Wer sich ans Diktieren mit KI gewöhnt hat, will das ohnehin nicht mehr nur im Terminal.

Berechtigungen

Claude fragt anfangs bei jeder Aktion nach Erlaubnis. Das ist am Anfang gut, um ein Gefühl zu bekommen. Später kannst du im Berechtigungsdialog "immer erlauben" wählen oder in `/config` einen entspannteren Modus setzen. Empfehlung: Die erste Woche alles bestätigen, danach lockern.

Der wichtigste Handgriff dabei: **Shift+Tab**. Damit schaltest du in der laufenden Session den Modus um, auf "auto-accept" führt Claude Aktionen direkt aus, ohne jedes Mal nachzufragen. Das mache ich in **jeder Session direkt als Erstes**, sonst sitzt du nur da und drückst Bestätigungen weg. Der Modus gilt immer nur für die aktuelle Session, also gewöhn dir Shift+Tab als ersten Tastendruck nach dem Start an. Was das sicher macht: Unter dem Modus liegt bei mir eine feste Sperrliste, die auch im auto-Modus greift, siehe Kapitel 12.

Die CLAUDE.md: dem Agenten eine Identität geben

Im Arbeitsordner (z. B. `~/Assistant/`) legst du eine Datei `CLAUDE.md` an. Die liest Claude bei jedem Start automatisch und richtet sich danach. Meine ist über die Monate auf neun Abschnitte gewachsen:

1. **Identität und Persona.** Wer der Agent ist und wie er klingt. Bei mir heißt er Falco und antwortet mit trockenem Wiener Einschlag; ein eigener Abschnitt am Ende sammelt Phrasen und No-Gos, den ich laufend erweitere. Das ist Geschmackssache, macht die tägliche Arbeit aber angenehmer.
2. **Über mich.** Name, E-Mail-Adressen, Firma, Sprachen. Die Basisdaten, die der Agent sonst in jeder Session neu erfragen müsste.
3. **Zusammenarbeit.** Wie geantwortet wird: kurz und direkt, keine Floskeln, Annahmen treffen statt nachfragen, am Ende ein bis zwei Sätze, was passiert ist.
4. **Kontextordner.** Ein kuratierter Wissensordner für den Agenten (über mich, Firma, Referenzmaterial) mit einer README als Index, die er bei passenden Fragen zuerst liest.
5. **Inbox-Workflow.** Was passiert, wenn ich eine Datei in den `inbox/`-Ordner werfe: lesen, in den richtigen Kontextordner einsortieren, Index aktualisieren, in einem Satz berichten.
6. **Ablage-Routing.** Welche Inhalte in meinen privaten Vault gehören, welche in den Team-Vault der Firma, und was nur Arbeitsmaterial bleibt (mehr zu den Vaults im Obsidian-Kapitel).
7. **Wiki-Pflege.** Die Regeln, nach denen der Agent den Vault als gepflegtes Wiki behandelt: Querverweise setzen, Widersprüche markieren, Chronik führen (auch dazu mehr im Obsidian-Kapitel).
8. **Externe Ressourcen.** Wo was liegt und wie der Agent drankommt: die beiden Google Drives, der Vault, die Connectors.

9. **Harte Grenzen.** Was nie ohne Rückfrage passiert: keine Mails senden, Firmen-Drive nur lesen, nichts endgültig löschen.

Du musst nicht mit neun Abschnitten starten. Meine erste Version hatte vier Zeilen; die Datei wächst mit jeder Regel, die du sonst zweimal erklären müsstest. Je präziser die CLAUDE.md, desto weniger musst du in jeder Session neu erklären.

Startbefehl im Alltag: `falco` (der eigene Startbefehl aus Kapitel 3), oder ein Tab bleibt dauerhaft im Assistant-Ordner offen.

Das Memory: was sich der Agent selbst merkt

Neben der CLAUDE.md, die ich selbst schreibe und kuratiere, führt Claude Code ein eigenes Memory-Verzeichnis: kleine Markdown-Dateien, eine pro Fakt, die der Agent von sich aus anlegt, wenn er etwas über mich oder meine Arbeitsweise lernt. Ich muss dafür nichts tun, außer gelegentlich zu korrigieren. Bei mir liegen dort inzwischen rund 40 Einträge in vier Gruppen. Ein Auszug:

Über mich

1. **Diktier-Fakten.** Bestimmte Verhörer meiner Spracheingabe meinen immer dasselbe Wort; der Agent weiß das und fragt nicht mehr nach.
2. **Unterschrift.** Liegt als Bilddatei bereit und wird auf Zuruf in Dokumente eingefügt.
3. **Dritte Mail-Adresse.** Neben den zwei Haupt-Accounts gibt es eine dritte Adresse, samt Kontext, wofür sie da ist.
4. **F1-Interesse und Space-Interesse.** Privates, damit Recherchen dazu gleich im richtigen Vault-Ordner landen.
5. …

Meine Feedbackregeln

1. **Nie Mails senden.** Mails werden immer nur als Entwurf vorbereitet; gesendet wird erst nach meinem OK.
2. **Keine Gedankenstriche.** In Texten, die für mich entstehen, kommt das Zeichen nicht vor.
3. **Sheets nur zellgenau.** Google Sheets werden Zelle für Zelle bearbeitet, nie geleert und neu geschrieben.
4. **Dev auf Englisch.** Alles Entwickler- und GitHub-Bezogene (Commits, READMEs, Doku) auf Englisch.
5. **LinkedIn über den Team-Skill.** Posts entstehen immer über unseren gemeinsamen Skill, nie freihändig.
6. …

Firma

1. Fachliche Fakten und Sprachregelungen rund um arcneo, die der Agent in Texten konsistent einhalten soll. Die Inhalte selbst gehören nicht in diesen Guide.
2. …

Projekte und Referenzen

1. **Kalender.** Welcher Kalender-Connector über welches Konto läuft und welcher nur lesbar ist.
2. **Personal Hub.** Mein privates Dashboard-Projekt, dessen Daten der Agent pflegt.
3. …

Der Unterschied zur CLAUDE.md: die CLAUDE.md pflege ich, das Memory wächst von selbst. Der Effekt ist derselbe wie bei einem guten Mitarbeiter, jede Korrektur sitzt beim ersten Mal, und nach ein paar Wochen fühlt sich der Agent an, als würde er dich kennen. Falsche Einträge lasse ich einfach löschen, das ist ein Satz im Chat.

Ein Detail, das man wissen sollte: Das Memory-Verzeichnis liegt nicht im Vault, sondern global unter `~/.claude/`. Auf einen zweiten Rechner kommt es deshalb nicht von selbst mit. Bei mir ist es per Symlink ins Repo eingehängt, wie das geht, steht in Kapitel 7.

---

5. Schritt 4: Connectors (MCP), Claude an deine Tools anschließen

Connectors verbinden Claude mit externen Diensten. Die meisten richtest du einmal auf **claude.ai** ein (Einstellungen, Connectors), danach stehen sie automatisch auch in Claude Code im Terminal zur Verfügung. Mit `/mcp` siehst du in Claude Code, was verbunden ist.

Meine Connectors und wofür ich sie nutze:

Connector	Wofür
Gmail	Mails suchen, Threads zusammenfassen, Entwürfe im eigenen Stil anlegen
Google Kalender	Termine suchen, anlegen, Zeitvorschläge ("Wann habe ich Freitag Zeit?")
Google Drive	Dokumente suchen und lesen (bei mir: Firmen-Drive nur lesend)
Notion	Seiten lesen, anlegen, Datenbanken abfragen
Figma	Designs lesen UND erstellen (meine LinkedIn-Posts und Pitch-Decks entstehen so)
HubSpot	CRM-Abfragen, Kontakte, Kampagnen
GitHub	Repos, Issues, Pull Requests (bei mir über die `gh`-CLI, die Claude direkt bedient)
Vercel	Websites deployen und verwalten, Änderungen direkt aus dem Terminal live stellen
Supabase	Datenbanken und Backends abfragen und verwalten
Brave Search	Websuche direkt aus dem Terminal

Connector	Wofür
Context7	Aktuelle Dokumentation von Tools und Bibliotheken
Namecheap	DNS-Einträge meiner Domain verwalten (lokal eingerichtet)

Praxis-Tipp: Nur verbinden, was du wirklich nutzt. Jeder Connector ist ein Zugriff, den der Agent hat. Bei sensiblen Systemen (z. B. Firmen-Drive) in der CLAUDE.md eine Nur-Lesen-Regel festhalten.

---

6. Schritt 5: Plugins und Skills

Plugins erweitern Claude Code um fertige Fähigkeiten ("Skills"). Installation in Claude Code über `/plugin`, dort den Marketplace durchsuchen. Was bei mir läuft:

- **Superpowers** (offiziell): **das absolute Must-have, als Erstes installieren.** Superpowers verändert, *wie* Claude arbeitet, um Klassen: Statt draufloszulegen klärt Claude erst die Anforderungen, plant, setzt dann um und verifiziert am Ende sein eigenes Ergebnis. Der Unterschied ist bei jeder nicht-trivialen Aufgabe spürbar, weniger Nachbessern, weniger "das habe ich so nicht gemeint". Wenn du nur ein einziges Plugin installierst, dann dieses.
- **Firecrawl** (CLI + Skills): **die zweite klare Empfehlung, speziell für Research.** Mein Web-Werkzeugkasten: Suchen, Webseiten sauber auslesen, ganze Doku-Seiten herunterladen, Preisseiten von Wettbewerbern beobachten. Das Highlight sind die Deep-Research-Skills, die liefern Berichte mit echten Quellenangaben statt Halbwissen aus dem Modellgedächtnis. Für alle Recherche-Use-Cases weiter unten ist Firecrawl die Grundlage. Installation: auf firecrawl.dev registrieren, dann die CLI mit den Skills installieren; danach kann Claude jederzeit "das Web lesen".
- **Frontend-Design** (offiziell): Sorgt für hochwertige, nicht nach Vorlage aussehende Gestaltung, wenn Claude Dokumente, Seiten oder Grafiken baut.
- **arcneo-branding** (eigenes Plugin): Meine Marken-Skills, die LinkedIn-Cards, OnePager und Pitch-Deck-Folien direkt in Figma im Firmen-CI erzeugen. Der Punkt dahinter: Man kann sich **eigene Plugins** für die eigenen wiederkehrenden Aufgaben bauen und im Team teilen. Genau das lohnt sich für Branding-Aufgaben enorm.
- **Game-Sounds** (Spielerei): Soundeffekte bei bestimmten Ereignissen. Muss nicht sein, macht aber Laune.
- **handelsregister** (eigenes Plugin, Open Source): Zieht Handelsregister-Dokumente für beliebige deutsche Firmen direkt vom offiziellen Portal, also Gesellschafterlisten, Gesellschaftsverträge und Registerauszüge (AD/CD/SI). Praktisch für KYC und Due Diligence. Entstanden, weil das Portal zwar seit 2022 kostenlos ist, aber keine API hat. Der Code liegt öffentlich auf GitHub (DespotB/handelsregister-cli). Jeder kann es direkt von dort installieren, ganz ohne unseren Hub: `/plugin marketplace add DespotB/handelsregister-cli`, dann `/plugin install handelsregister@handelsregister-cli`. Danach reicht ein `/handelsregister Beispiel GmbH` und die Dokumente liegen sauber benannt und dedupliziert in einem Ordner.

Dazu kommen **Slash-Commands**: eigene Kurzbefehle wie `/note` (Tagesnotiz in den Vault schreiben) oder `/dns` (DNS-Änderung mit Vorher/Nachher-Prüfung). Das sind einfache Textdateien im Ordner `.claude/commands/`, jeder kann sich seine eigenen bauen, einfach Claude darum bitten.

Vom Einzel-Setup zum Team

Wie man dieses Plugin-Prinzip auf die ganze Firma ausweitet (geprüfte Skills zentral bauen und pflegen, statt dass jeder auf seinem Rechner eigene Varianten bastelt), zeigt Kapitel 8.

---

7. Schritt 6: Obsidian als Gedächtnis (Second Brain)

Claude Code kann Dateien lesen und schreiben. Obsidian speichert Notizen als einfache Markdown-Dateien in einem Ordner ("Vault"). Diese Kombination ist der Kern meines Setups: **Claude arbeitet direkt in deinem Obsidian-Vault**, ohne Plugin, ohne Schnittstelle, einfach über das Dateisystem.

1. Obsidian installieren: obsidian.md (kostenlos)
2. Einen Vault anlegen, z. B. unter `~/Assistant/vault/`
3. Eine sinnvolle Ordnerstruktur wählen. Meine:

```
vault/
├── Daily/                    Tagesnotizen und Journal
├── Mail/                     Mail-Stilanalyse, Kontakte, Archiv
├── Work/
│   ├── arcneo/               Aktuelle Firma, Unterordner spiegeln das Share
│   │   ├── 04 HR
│   │   ├── 06 Legal
│   │   ├── 07 Partners
│   │   ├── 08 Marketing & Sales
│   │   ├── 10 Events
│   │   ├── 11 Funding
│   │   └── 14 Agents-Workspace
│   └── Karriere & Zukunft/   Arbeit, aber nicht aktuell
└── Privat/
    ├── Freunde & Familie/
    ├── Wohnung & Verträge/
    ├── Hobbys & Freizeit/    z. B. DJ/, Gaming/
    ├── Bildung & Wissen/
    └── Organisation/         Anlässe, Planungen, Reden
```

Tipp aus der Praxis: Den Work-Ordner genauso benennen wie das Shared Drive der Firma. Dann findet Claude Dinge an beiden Orten unter demselben Pfadnamen.

Zur Einordnung: Das hier ist mein **privater** Vault: meine Struktur, mein Rechner, mein Backup. Daneben gibt es bei uns noch ein zweites Obsidian: den **geteilten Team-Vault** der Firma, ein gemeinsames Git-Repo als Teil des arcneo Hubs (dazu ausführlich Kapitel 8). Der ist bewusst anders geschnitten, nicht nach Lebensbereichen, sondern nach Wissens-Typen: `Meetings/` (alle Protokolle, benannt `YYYY-MM-DD Thema.md`), `Wissen/` (abgeschlossene Recherchen, Konzepte, Entscheidungen), `Projekte/`, `Vorlagen/`, `Archiv/` (Veraltetes wird markiert statt gelöscht) und eine `00 Inbox` als Drop-Zone. Die Trennlinie ist hart: Privates bleibt im privaten Vault, Firmenthemen gehören in den Team-Vault, und keiner der beiden verlinkt in den anderen.

Wichtig: Obsidian braucht bei mir **keine Community-Plugins**. Alles, was sonst Plugins erledigen (Templates, Auto-Verlinkung, Importe), macht Claude. Obsidian ist nur der schöne Viewer mit Graph-Ansicht und Verlinkung, Claude ist der Autor.

Drei Ausbaustufen für den Vault (GitHub-Backup, zweiter Rechner, der Vault als gepflegtes Wiki) stehen gesammelt in Kapitel 12.

---

8. Schritt 7: Der arcneo Hub, ein Notion für die Agenten

Seit August 2026 gibt es bei uns ein zentrales, privates Git-Repo namens **arcneo Hub** (`arcneo-gmbh/arcneo-hub`). Die kürzeste Definition: **Was Notion für die Menschen im Team ist, ist der Hub für die Agenten**: der eine Ort, an dem alles liegt, was jeder Claude-Agent in der Firma wissen und können muss. Das Prinzip dahinter: einmal bauen, im ganzen Team nutzen. Das Repo ist die einzige Wahrheitsquelle; Skills, Prompts und Team-Wissen werden dort gepflegt und versioniert, nicht auf Einzel-Rechnern.

Was drin liegt

Ordner	Inhalt
`skills/`	Geprüfte Team-Skills, als Plugin `arcneo-skills` installierbar (LinkedIn-Posts, Meeting-Protokolle, Investoren-FAQ, …)
`prompts/`	Prompt-Library für alle, die nur mit dem Chat auf claude.ai arbeiten (kein Claude Code nötig)
`vault/`	Der gemeinsame Obsidian-Vault: `Meetings/`, `Wissen/`, `Projekte/`, `Vorlagen/`, `Archiv/` (siehe Kapitel 7)
`onboarding/`	Der Agent Workspace: Vorlage + geführtes Setup für den persönlichen Assistenten
`docs/`	Anleitungen: Wie schreibe ich einen Skill? Was gehört wohin?

Ordner	Inhalt
`.claude-plugin/`	Die Marketplace-Definition, macht das Repo als Plugin-Quelle installierbar

Installation pro Person: `/plugin marketplace add arcneo-gmbh/arcneo-hub`, dann `/plugin install arcneo-skills@arcneo-hub`. Der Marketplace liefert auch das Branding-Plugin und das Handelsregister-Plugin gleich mit aus. Updates holt `/plugin marketplace update arcneo-hub`. Neue und verbesserte Skills kommen so automatisch bei allen an.

Warum wir das haben

- **Ein geprüfter Stand statt zehn Bastellösungen.** Ein Skill wird einmal gebaut, reviewt und dann von allen genutzt. Verbessert jemand einen Skill, bekommt es das ganze Team beim nächsten Update.
- **Gemeinsames Gedächtnis.** Der Team-Vault (alle Meeting-Protokolle, alles abgeschlossene Wissen) ist für jeden Agenten im Team automatisch verfügbarer Kontext. Der Agent eines Kollegen weiß, was in meinem Meeting letzte Woche entschieden wurde.
- **Onboarding in Stunden statt Tagen.** Neue Teammitglieder klonen das Repo, starten `claude` darin und sagen „Führe das Onboarding durch". Der Agent richtet Plugins, Vaults und Connectors selbst ein.
- **Auch ohne Terminal nutzbar.** Die Prompt-Library deckt die ab, die nur mit dem Chat arbeiten. Der Hub ist für das ganze Team da, nicht nur für die Power-User.

Was wohin gehört (die Ablage-Matrix)

Der Hub ersetzt weder Notion noch das Google Drive. Die Systeme haben klare Rollen, und eine der wichtigsten Seiten im Hub ist genau diese Entscheidungsmatrix (`docs/ablage-regeln.md`), an die sich Menschen **und** Agenten halten. Die Faustregel: **Markdown-Wissen ins Repo, Lebendes nach Notion, Binäres ins Drive.**

Warum kommt Wissen nicht einfach alles nach Notion? Weil die beiden Orte für unterschiedliche Leser gebaut sind. Agenten finden sich in einem Vault aus Markdown-Dateien schlicht besser zurecht als in Notion: Sie lesen das Dateisystem direkt, durchsuchen hunderte Seiten in Sekunden, folgen Verlinkungen ohne API-Umweg und sehen über Git sogar, wann sich welches Wissen geändert hat. Das heißt schnellere Antworten auf besserer Informationsbasis. Und der Nebeneffekt ist genauso wichtig: Notion bleibt sauber. Wissen, das primär für Agenten da ist (Recherchen, Prozess-Doku, Hintergrund-Kontext), würde Notion nur zumüllen; dort sollen Menschen ihre lebenden Boards und Datenbanken finden, nicht hundert Referenzseiten, die nie ein Mensch aufschlägt.

Inhalt	Ort
Meeting-Protokolle, abgeschlossenes Wissen (Recherchen, Konzepte, Entscheidungen)	**Hub-Vault**

Inhalt	Ort
Laufende Aufgaben, Task-Boards, lebende Datenbanken (z. B. Rechtsfragen-Tracker, Pipelines)	**Notion**
Lead-Listen, Kontaktdaten	**Notion** / **HubSpot** (Personenbezogenes gehört nicht in ein Git-Repo)
Binärdateien: PDFs, Verträge, Renderings, Präsentationen	**Drive**
Zugangsdaten, API-Keys	**Passwort-Manager** (niemals in Repo, Notion oder Drive)
Privates	**Eigener lokaler Vault**, nie in Team-Systeme

Manches gehört bewusst an zwei Orte, aber nur nach dem **Master-Kopie-Muster**: Eine Stelle ist immer das Master, die andere eine datierte Kopie. Beispiel: Eine lebende Notion-Datenbank, die Agenten als Kontext brauchen, bekommt einen datierten Markdown-Export im Vault; bei Widerspruch gilt Notion. Oder: Verträge bleiben als PDF im Drive, ein Index im Vault verlinkt sie, und nur was Agenten ständig als Text brauchen, wird zusätzlich als Markdown konvertiert, mit Quellenverweis. Was nicht erlaubt ist: dieselbe Information an zwei Stellen parallel pflegen. Das erzeugt zwei Wahrheiten, und irgendwann stimmt keine mehr.

So baust du das für deine eigene Firma

Der Hub ist kein Produkt, sondern ein Muster. Und der Nachbau ist selbst ein Claude-Auftrag, kein Wochenendprojekt:

1. **Privates GitHub-Repo anlegen** (z. B. `deinefirma/deinefirma-hub`), Team-Mitglieder einladen.
2. **Drei Kern-Ordner:** `skills/`, `prompts/`, `vault/`.
3. **Claude die Marketplace-Definition bauen lassen:** „Mach aus diesem Repo einen Claude-Code-Plugin-Marketplace, der die Skills aus `skills/` als Plugin ausliefert." Ab dann ist das Repo per `/plugin marketplace add` installierbar.
4. **Die ersten zwei, drei Skills destillieren**: aus den Aufgaben, die im Team am häufigsten wiederkehren (bei uns: Meeting-Protokolle, LinkedIn-Posts, Investoren-FAQ).
5. **Eine Ablage-Regeln-Seite schreiben**, damit von Anfang an klar ist, was ins Repo, was nach Notion, was ins Drive gehört.
6. **Team onboarden:** Repo klonen, Marketplace hinzufügen, `vault/` in Obsidian öffnen (mit dem Community-Plugin Git für Auto-Sync).

Eine Regel gilt ab Tag eins und ohne Ausnahme: keine Passwörter, keine privaten Inhalte, keine sensiblen Personaldaten ins Repo. Der Rest wächst von selbst. Jedes Meeting-Protokoll und jeder neue Skill macht den Hub wertvoller.

9. Mein täglicher Workflow

So sieht ein normaler Tag bei mir aus:

1. **iTerm2 mit benannten, gefärbten Tabs.** Ein Tab pro Kontext, z. B. blau = "Falco" (persönlicher Assistent), grün = "arcneo" (Firma), orange = "Recherche". Jeder Tab ist eine eigene Claude-Session mit eigenem Gesprächsverlauf. Die Namen vergibt der Agent: In Claude Code benennt der Befehl `/rename <Wort>` die Session um, und zwar an beiden Stellen gleichzeitig, im Terminal-Tab oben und in der Session-Liste, die man in Remote Control auf dem Handy sieht. Ohne Umbenennen vergibt Claude dort Zufallsnamen wie "Polish Blossom", mit denen man nichts anfangen kann. Ich habe meinem Agenten deshalb in die CLAUDE.md die Regel geschrieben: Sobald in einer neuen Session das erste Thema klar ist, benennt er die Session automatisch mit genau einem prägnanten Wort (technisch über ein kleines Skript, das `/rename` per AppleScript ins eigene iTerm2-Fenster tippt). Der Effekt ist simpel, aber Gold wert: Der Tab oben heißt immer exakt gleich wie die Session, und wenn ich unterwegs per Remote Control vom Handy reinschaue, erkenne ich sofort, welche Session zu welchem Thema und welchem Tab gehört.
2. **Voice statt Tippen.** Taste halten, Auftrag diktieren, loslassen. Gerade unterwegs zwischen Meetings der schnellste Weg.
3. **Alles Wichtige landet im Vault.** Ergebnisse (Recherchen, Briefings, Analysen) schreibt Claude als Markdown-Notiz mit Verlinkungen in den passenden Obsidian-Ordner. Abends in Obsidian durchblättern, alles ist da und verlinkt.
4. **Lange Aufgaben laufen im Hintergrund.** Claude arbeitet weiter, iTerm2 meldet sich, wenn es fertig ist. Währenddessen im nächsten Tab das nächste Thema.
5. **Kontext-Fenster sauber halten, immer.** Das ist wichtiger, als es klingt: Claude hat pro Session ein begrenztes "Gedächtnis" (das Kontext-Fenster). Je länger eine Session läuft und je mehr Themen sich darin vermischen, desto langsamer, teurer und unkonzentrierter wird der Agent. Deshalb die eiserne Regel: **Neues Thema = neuer Tab oder `/clear` im bestehenden Tab.** `/clear` leert den Gesprächsverlauf und startet frisch (die CLAUDE.md und alle Einstellungen bleiben natürlich erhalten). Niemals ein neues Thema in eine volle Session hineinquetschen.
6. **Bei sehr anspruchsvollen Aufgaben: ultrathink.** Ein magisches Keyword: Schreibst du das Wort "ultrathink" irgendwo in deinen Auftrag, schaltet Claude in den tiefsten Denkmodus. Claude bekommt dann das maximale Budget an "Nachdenk-Zeit" und überlegt ausführlich, bevor es antwortet oder loslegt: Es wägt Varianten ab, prüft die eigene Logik und plant mehrere Schritte voraus. Das Wort wird im Terminal sogar bunt eingefärbt, damit du siehst, dass es erkannt wurde. Für Alltagsaufgaben unnötig (dauert länger), aber bei komplexen Sachen, etwa einer heiklen Verhandlungs-Mail, einem strategischen Plan oder einer kniffligen Analyse, macht es einen spürbaren Qualitätsunterschied. Beispiel: "ultrathink: Entwirf die Antwort an den Investor, hier ist der Verlauf ..."
7. **Und die höchste Eskalationsstufe: ultracode.** Das zweite magische Keyword. Steht "**ultracode**" im Auftrag, erledigt Claude die Aufgabe nicht mehr allein, sondern orchestriert einen ganzen Schwarm von Subagenten: Die Aufgabe wird zerlegt, parallel bearbeitet, und die Ergebnisse prüfen sich gegenseitig, bevor etwas zurückkommt. Zur Einordnung: Superpowers macht genau das in kleinem Maß-

stab sowieso schon (einzelne Subagenten für Recherche oder Review). **ultracode** dreht es auf Dutzende Agenten hoch, entsprechend teurer und langsamer. Deshalb nur bei extrem aufwendigen Aufgaben einsetzen, etwa einem kompletten Audit, einer großen Recherche über viele Quellen oder einer Analyse, die wirklich in die Tiefe muss.

8. **Der Agent verbessert sich selbst.** Wenn etwas schiefläuft oder ich eine Vorliebe korrigiere, lasse ich Claude die CLAUDE.md ergänzen. Jeder Fehler passiert nur einmal.

---

10. Use Cases aus der Praxis (zum Nachbauen)

Der persönliche E-Mail-Stil-Guide

Mein Lieblingsbeispiel. Ich habe mein komplettes Mail-Archiv (rund 2.500 Mails aus drei Jahren, per Google-Takeout-Export) an Claude gegeben. Claude hat sie komplett gelesen und daraus eine **Stil-Bibel** destilliert: Wie ich Leute anspreche (wann "Hallo", wann "Sehr geehrte"), wie ich grüße, Satzbau, Lieblingswörter, was ich nie tue (Emojis, Floskeln), sogar Unterschiede zwischen deutschen und englischen Mails. Ergebnis ist eine Notiz `Mail/Stil.md` im Vault, plus eine Kontaktliste und Few-Shot-Beispiele.

Seitdem gilt: Wenn Claude eine Mail in meinem Namen entwirft, liest er erst den Stil-Guide. Die Entwürfe klingen nach mir, nicht nach KI. Aufwand einmalig ein Nachmittag, Nutzen täglich.

Competitor-Research: Charts für die Fintech-Plattform

Für unsere Fintech-Plattform wollten wir wissen, wie Wettbewerber festverzinste Anleihen visualisieren: welche Chart-Typen sie einsetzen, wie Zinsverlauf und Rückzahlung dargestellt werden, was davon Standard ist und was nicht. Claude hat die Plattformen der Wettbewerber recherchiert und die Ergebnisse direkt in Notion aufgearbeitet, als komplette Page mit Übersicht pro Wettbewerber, den verwendeten Chart-Typen und einer Empfehlung für unsere Plattform. Im Anschluss hat Claude in Figma erste Beispiel-Charts in unserem Branding angelegt, als Diskussionsgrundlage fürs Design. Von der Frage bis zur präsentierbaren Seite eine Session, ohne dass ich selbst recherchieren musste.

Branding und Content direkt in Figma

Branding und Content sind bei uns ein **eigenes Plugin** (`arcneo-branding`, ausgeliefert über den Hub-Marketplace) mit drei Skills, die direkt in Figma arbeiten: **LinkedIn-Cards** (Posts, Karussells, Speaker-Cards, Banner und verwandte Formate), **Pitch-Deck-Folien** (16:9) und der **OnePager** (A4-Factsheet für Investoren). Das Fundament darunter: ein selbst geschriebenes **Branding-Guidebook** (Farben, Typografie-Skala, Layout-Patterns, Format-Specs), das jeder der drei Skills vor der Arbeit liest, plus eine Sammlung echter **Beispiel-Folien und -Designs**, die die Skills recyceln oder als Orientierung nutzen. Aus "Mach einen Ankündigungspost für die Messe" wird so ein fertiges Design im richtigen CI (Farben, Schriften, Logo-Platzierung), zum Feinschliff statt zum Selberbauen. Für die **Texte** gibt es daneben den Team-Skill `linkedin-post` im Hub, der verschiedene Post-Typen schreiben kann (Messe-Ankündigun-

gen, Produkt-Updates, Meilensteine, Gründer-Posts), auf Basis einer Stil-Bibel mit echten Referenz-Posts, analog zum Mail-Stil.

Guides aus den eigenen Learnings

Der Meta-Use-Case: Dieser Guide hier ist selbst so entstanden. Claude hat ihn geschrieben, aus allem, was sich über die Monate im Setup angesammelt hat (Vault, CLAUDE.md, Workflows, gelöste Probleme). Ein Prompt genügt ("Schreib aus meinem Setup einen Guide für Einsteiger"), denn das Wissen liegt ja schon bei Claude: Es muss nichts recherchieren, nur verdichten und in eine lesbare Reihenfolge bringen. Das Muster funktioniert für alles, was man weitergeben will: intern im Team als How-we-work-Dokumente (z. B. wie wir mit Sales arbeiten, wie wir Meetings nachbereiten), extern als Guide wie dieser. Einmal aufgeschrieben, lebt so ein Guide als normale Markdown-Datei im Vault oder Hub und wird bei Änderungen einfach mitgepflegt.

Verwaltung und Kleinkram

- **DNS- und Domain-Verwaltung:** Über zwei Connectors, den Vercel MCP und den Namecheap MCP, ist Claude direkt an meine Domains angeschlossen (privat und Firma). Das heißt konkret: Ich kann Inhalte auf meinen Webseiten direkt aus dem Terminal bearbeiten und live stellen, und ich kann DNS-Einträge per Zuruf ändern, mit eingebauter Vorher/Nachher-Kontrolle. Dazu gehört auch der Security-Teil für E-Mail: SPF, DKIM und DMARC richtet Claude auf den Domains sauber ein und prüft sie, damit meine Mails nicht im Spam landen und niemand in meinem Namen senden kann.
- **Event-Radar:** Claude beobachtet Branchen-Events und pflegt eine Liste im Vault.
- **LinkedIn-Netzwerkpflege:** Ein kleiner Startup-Check erinnert Claude einmal am Tag daran, über die Chrome-Erweiterung mein LinkedIn-Netzwerk zu erweitern (eine Handvoll Kontaktanfragen an passende Profile, mit hartem Tageslimit und Log im Vault). Das Muster dahinter ist wiederverwendbar: ein Hook prüft beim Start, ob eine tägliche Routine schon gelaufen ist, und stupst Claude sonst an.
- **Meeting-Nachbereitung:** Notizen diktieren, Claude strukturiert sie und legt sie im richtigen Firmenordner ab.

---

11. Die häufigsten Fehler von Einsteigern

1. **Terminal nach der Installation nicht neu gestartet** → "command not found". Einfach iTerm2 komplett beenden und neu öffnen.
2. **Mit dem Console-/API-Konto statt dem Claude-Abo angemeldet** → unerwartete Kosten pro Anfrage. Beim Login "Claude account" wählen; korrigieren mit `/login`.
3. **Im falschen Team/der falschen Organisation gelandet** → mit `/login` neu anmelden und die richtige Organisation wählen.
4. **Ohne CLAUDE.md arbeiten** → jede Session bei null anfangen. Die halbe Stunde für eine gute CLAUDE.md ist die beste Investition im ganzen Setup.

5. **Claude im Home-Verzeichnis starten statt im Arbeitsordner** → Claude sieht seinen Kontext nicht. Immer erst `cd ~/Assistant` (oder wo dein Setup liegt), dann `claude`. Am bequemsten mit dem eigenen Startbefehl aus Kapitel 3, der beides in einem macht.
6. **Alles in einer einzigen Session machen** → der Verlauf wird lang und unübersichtlich. Pro Thema ein Tab, bei Themenwechsel `/clear` oder neue Session.
7. **Jeden Connector verbinden, "weil man kann"** → unnötige Zugriffe. Nur anschließen, was gebraucht wird, und Schreibrechte bewusst vergeben.
8. **Ctrl+V fügt kein Bild ein** → entweder Cmd+V statt Ctrl+V gedrückt, oder iTerm2 darf nicht auf die Zwischenablage zugreifen. Haken bei "Applications in terminal may access clipboard" setzen (Settings, General, Selection) und iTerm2 komplett neu starten (siehe Kapitel 2).
9. **Harte Grenzen nur in die CLAUDE.md schreiben** → das ist eine Bitte ans Modell, kein Schutz. Was nie passieren darf (Mails senden, rekursiv löschen, Live-Deploy), gehört zusätzlich als Deny-Regel oder Hook in die Settings (Kapitel 12).

---

12. Advanced: Wenn das Grundsetup läuft

Alles in diesem Kapitel ist optional. Es lohnt sich, sobald das Setup aus den Kapiteln 2 bis 8 ein paar Wochen im Alltag läuft und du merkst, wo es zwickt: Backup, zweiter Rechner, ein Vault, der sich selbst pflegt, und Berechtigungen, die auch dann halten, wenn der Agent fremde Inhalte liest. Die Reihenfolge ist zugleich meine Empfehlung, in welcher Reihenfolge man das angeht.

Advanced: Den Vault als GitHub-Repo sichern

Wenn das Grundsetup läuft, lohnt sich ein Schritt weiter: den Vault-Ordner in ein **privates GitHub-Repository** verwandeln. Das musst du nicht selbst können, das ist selbst schon ein Claude-Auftrag ("Mach aus meinem Vault ein privates GitHub-Repo und richte ein, dass Änderungen automatisch gesichert werden"). Was du davon hast:

- **Backup und Zukunftssicherheit.** Jede Notiz ist versioniert. Nichts geht mehr verloren, auch wenn der Rechner stirbt, und du kannst jede alte Version jeder Notiz wiederherstellen.
- **Mehrere Rechner.** Auf einem zweiten Mac (privat/Arbeit) das Repo klonen, und derselbe Vault mit derselben CLAUDE.md steht dort zur Verfügung. Das ist aber nur ein Teil dessen, was der Agent über dich weiß. Was mitkommt und was nicht, steht im nächsten Abschnitt.
- **Claude committet selbst.** Bei mir steht in der CLAUDE.md die Regel, dass Claude nach jeder Änderung im Vault automatisch committet und pusht. Das Backup passiert dadurch nebenbei, ohne dass ich je daran denke.

Sensible Ordner (z. B. das private Mail-Archiv) lassen sich per `.gitignore` vom Repo ausnehmen, sie bleiben dann nur lokal.

Advanced: Zweiter Rechner und Umzug, was mitkommt und was nicht

Das Repo deckt nur einen Teil dessen ab, was Claude Code über dich weiß. Der Rest liegt im Home-Verzeichnis unter `~/.claude/` und synchronisiert sich nicht von selbst. Die Aufteilung bei mir:

- **Kommt über das Repo mit:** der Vault, die CLAUDE.md des Projekts, die Projekt-Commands und Hooks im Ordner `.claude/` des Arbeitsordners.
- **Liegt global in `~/.claude/` und kommt nicht von selbst mit:** das automatisch gewachsene Memory (Kapitel 4), die globale CLAUDE.md, die Settings (Sprache, Modell, Statuszeile, Berechtigungsregeln und Hooks), eigene Helfer-Skripte, installierte Plugins und Skills.
- **Bleibt bewusst lokal:** die Datei `~/.claude.json`, die direkt im Home liegt (nicht in `~/.claude/`) und Login, MCP-Server samt API-Keys und den Trust-Status der Ordner enthält, sowie die Session-Verläufe.

Zwei Stolpersteine, die man kennen muss. Erstens liegt `~/.claude.json` außerhalb von `~/.claude/`. Wer nur den Ordner kopiert, verliert Login und MCP-Server. Zweitens benennt Claude Code das Memory-Verzeichnis und die Session-Verläufe nach dem absoluten Pfad des Arbeitsordners (bei mir `~/.claude/projects/-Users-despot-b-Assistant/`). Ein anderer Benutzername oder ein anderer Ordnerpfad auf dem zweiten Rechner, und das Memory ist zwar da, wird aber nicht gefunden. Darum auf allen Geräten derselbe Benutzername und derselbe Pfad.

Meine Lösung: Die Teile, die mitkommen sollen, liegen im Repo in einem Ordner `claude-home/` (Memory, globale CLAUDE.md, Skripte, ein Abbild der Settings) und sind per Symlink in `~/.claude/` eingehängt. Der Agent schreibt sein Memory damit direkt ins Repo, und die Commit-Regel aus der CLAUDE.md sichert es mit. Auf einem neuen Rechner heißt das: Repo klonen, das Skript `claude-home/link.sh` ausführen (setzt die Symlinks), `claude` starten, einloggen, MCP-Server und Plugins neu einrichten. Eingerichtet hat das Claude, inklusive Skript und README, ein Auftrag im Chat. Für einen einmaligen Umzug ohne Repo reicht auch rsync: `~/.claude/` und `~/.claude.json` auf den neuen Rechner kopieren. Bei gleichem Benutzernamen und Pfad läuft alles weiter, nur die Anmeldung wird einmal neu fällig, weil die Zugangsdaten im Schlüsselbund von macOS liegen und nicht in den Dateien.

Advanced: Der Vault als gepflegtes Wiki (drei Schichten)

Der nächste Reifegrad, inspiriert von Andrej Karpathys "LLM Wiki"-Idee: Der Vault ist nicht nur Ablage, sondern ein Wiki, das der Agent aktiv pflegt. Das Setup hat drei Schichten:

1. **Rohquellen (`vault/raw/`).** Artikel, PDFs und Web-Clips landen hier und werden nie verändert. Praktischer Zubringer: die Browser-Erweiterung **Obsidian Web Clipper** macht aus jedem Artikel eine Markdown-Datei. In Obsidian dazu den Attachment-Ordner auf `raw/assets/` stellen und die Funktion "Download attachments for current file" auf einen Hotkey legen (bei mir Ctrl+Shift+D), dann liegen auch alle Bilder lokal und Claude kann sie ansehen.
2. **Das Wiki (der restliche Vault).** Die von Claude geschriebenen, untereinander verlinkten Notizen. Die Regel, die den Unterschied macht: Beim Einlesen einer neuen Quelle legt Claude nicht nur eine Zusammenfassung ab, sondern **aktualisiert auch die bestehenden Seiten**. Es setzt Querverweise, markiert Widersprüche zu älterem Wissen und legt für wiederkehrende Themen eigene Seiten an. So wird Wissen einmal verdichtet und bleibt aktuell, statt bei jeder Frage neu zusammengesucht zu werden (der Unterschied zu klassischem RAG bzw. NotebookLM-artigen Tools).

3. **Das Schema (die CLAUDE.md).** Dort stehen genau diese Pflege-Regeln, damit jede Session sie automatisch befolgt. Du schreibst das Wiki nie selbst: Du lieferst Quellen und stellst Fragen, Claude macht die Buchhaltung. Obsidian ist die Leseansicht (die Graph-Ansicht zeigt dir die Struktur), Claude ist der Autor.

Dazu zwei kleine Helfer, die sich Claude auf Zuruf selbst baut:

- `vault/log.md`: eine append-only Chronik (was wurde wann eingelesen, recherchiert, geprüft). Gibt jeder neuen Session sofort Kontext über den Stand des Wikis.
- `/lint-vault`: ein eigener Befehl, der das Wiki periodisch auf verwaiste Seiten, tote Links, Widersprüche und fehlende Verknüpfungen prüft und Fixes vorschlägt.

Der Effekt: Gute Antworten verschwinden nicht im Chatverlauf, sondern werden als verlinkte Notiz ins Wiki zurückgeschrieben. Das Wissen kumuliert, jede Recherche macht die nächste besser. (Karpathys Original-Idee: gist.github.com/karpathy/442a6bf555914893e9891c11519de94f)

Advanced: Berechtigungen technisch absichern

In Kapitel 4 steht, dass Shift+Tab bei mir der erste Tastendruck jeder Session ist. Das ist bequem, aber mit dem Connector-Set aus Kapitel 5 auch eine andere Größenordnung als beim Notizenschreiben: Der Agent kann Mails senden, in Datenbanken schreiben, Websites deployen und DNS-Einträge ändern. Gleichzeitig liest er laufend fremde Inhalte (Mails, Web-Clips, gescrapte Seiten), und fremder Text kann Anweisungen enthalten, die nicht von dir stammen. Die harten Regeln in der CLAUDE.md ("nie Mails senden", "Firmen-Drive nur lesen", "kein rm -rf") sind dagegen kein Schutz. Sie sind eine Bitte an das Modell, die es fast immer befolgt, aber eben nicht garantiert.

Belastbar sind drei Dinge, alle in der Datei `~/.claude/settings.json`:

1. **Deny-Regeln.** Eine Liste von Werkzeugen und Befehlen, die Claude nie ausführen darf, egal in welchem Modus. Bei mir stehen dort unter anderem die Sende-Funktionen des Gmail-Connectors, rekursives Löschen (`rm -rf`), `git push --force`, jedes Schreiben ins Firmen-Drive und das Lesen von Schlüsseldateien wie `~/.ssh`. Eine Deny-Regel gewinnt immer, auch gegen "immer erlauben" und auch im auto-Modus.
2. **Ask-Regeln.** Eine Liste von Aktionen, bei denen Claude immer nachfragt, auch wenn Shift+Tab aktiv ist. Bei mir: DNS-Änderungen, Datenbank-Migrationen, Deployments auf die Live-Website, Löschen von Mails oder Kalenderterminen, Teilen von Drive-Dateien, Lesen von `.env`-Dateien mit Zugangsdaten. Alles, was nach außen wirkt oder schwer rückgängig zu machen ist.
3. **Ein Hook.** Deny-Regeln prüfen nur den Anfang eines Befehls. Was dahinter kommt (`cd x && rm -rf y`, ein Skript in einem Heredoc), sehen sie nicht. Dafür gibt es Hooks: ein kleines Skript, das vor jedem Shell-Befehl läuft, den kompletten Befehl bekommt und ihn ablehnen kann. Meines prüft per Regex auf rekursives Löschen in jeder Schreibweise, force-push, schreibende Befehle mit dem Pfad des Firmen-Drives (Lesen und Kopieren aus dem Drive bleibt erlaubt) und auf Mail-Versand per AppleScript oder Kommandozeile.

So sieht der Kern der Datei bei mir aus (gekürzt):

```
{
  "permissions": {
    "deny": [
      "Bash(rm -rf*)",
      "Bash(git push --force*)",
      "Edit(//Users/despot_b/Library/CloudStorage/GoogleDrive-despot@arcneo.
      "mcp__claude_ai_Gmail__send_message",
      "Read(~/.ssh/**)"
    ],
    "ask": [
      "mcp__namecheap__update_dns_record",
      "mcp__claude_ai_Supabase__apply_migration",
      "Bash(vercel deploy --prod*)",
      "mcp__claude_ai_Gmail__trash_message"
    ]
  },
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          { "type": "command", "command": "/Users/despot_b/.claude/hooks/gua
        ]
      }
    ]
  }
}
```

Drei Dinge dazu, die man wissen muss:

- **Global statt pro Projekt.** Die Datei `~/.claude/settings.json` gilt in jedem Arbeitsordner und auch für alle Unteragenten, die Claude selbst startet (die erben sonst nur die Erlaubnisliste der Session). Projektspezifische Regeln gehören in `.claude/settings.json` im jeweiligen Arbeitsordner. Das passt gut zum Tab-pro-Thema-Workflow aus Kapitel 9: Der Coding-Ordner darf andere Dinge als der Assistenten-Ordner, weil jeder Ordner seine eigene Regel-Datei hat.
- **Das ist ein Claude-Auftrag.** Du musst keine Regel-Syntax lernen. "Lies meine CLAUDE.md und setz jede harte Grenze darin als Deny-Regel, Ask-Regel oder Hook um, mit Tests" reicht. Danach prüfen lassen: "Versuch mal, eine Datei ins Firmen-Drive zu kopieren." Die Ablehnung muss kommen, mit Begründung.
- **Die Faustregel.** Jedes "nie" in der CLAUDE.md bekommt ein technisches Gegenstück. Kommt eine neue harte Regel dazu, wird sie am selben Tag auch in die Settings eingetragen. Dazu gehört Least Privilege bei den Zugängen: Der DNS-Connector bei mir kann nur DNS-Einträge lesen und schreiben, nicht Domains kündigen; das Firmen-Drive ist per Regel nur lesbar; Schreibrechte bekommt nur der Connector, der sie braucht (siehe Kapitel 5).

Eine Nebenwirkung: Der Hook lehnt auch Befehle ab, die die verbotenen Wörter nur *enthalten*, etwa wenn Claude einen Text mit "osascript" und "send" in eine Datei schreiben will. Dann weicht Claude auf seine Datei-Werkzeuge aus, was gewollt ist. Lieber einmal zu viel abgelehnt als einmal zu wenig.