PRAXISANLEITUNG · WINDOWS 10 / 11

Claude Code einrichten

Schritt für Schritt

Installation, Anmeldung, Arbeitsbereich, Regeln, Berechtigungen und ein sparsamer Arbeitsstil – vollständig und in der richtigen Reihenfolge.

Windows 10 & 11 | CLAUDE.md & Skills | Berechtigungen | Sparsam arbeiten | Checkliste

Roland H. Löffler

Ausgabe 2026 · Stand: 5. November 2026 · Geprüft gegen die offizielle Dokumentation unter code.claude.com/docs

Impressum

Claude Code einrichten – Schritt für Schritt
Praxisanleitung für Windows 10 und Windows 11

© 2026 Roland H. Löffler
Alle Rechte vorbehalten. Diese Anleitung einschließlich aller Texte ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechts ist ohne Zustimmung des Autors unzulässig.

Kontakt: Roland Löffler, Eisenbahnstraße 9, 72461 Albstadt

Stand: 5. November 2026

Claude und Claude Code sind Marken der Anthropic, PBC. Diese Anleitung ist eine unabhängige Veröffentlichung und keine offizielle Publikation von Anthropic. Windows, PowerShell und Visual Studio Code sind Marken der Microsoft Corporation. Git ist eine Marke der Software Freedom Conservancy.

Die Inhalte wurden mit großer Sorgfalt erarbeitet und am oben genannten Datum gegen die offizielle Dokumentation geprüft. Claude Code wird laufend weiterentwickelt; Menüs, Befehle und Voreinstellungen können sich mit neuen Versionen ändern. Eine Garantie für Richtigkeit, Vollständigkeit und Aktualität wird nicht übernommen. Maßgeblich ist im Zweifel die aktuelle Dokumentation unter code.claude.com/docs.

Inhalt

1. Was Sie brauchen
2. Git für Windows installieren
3. Claude Code installieren
4. Installation prüfen
5. Anmelden
6. Arbeitsbereich zuweisen
7. Regeln verwalten (CLAUDE.md, Skills, Praxisbeispiele)
8. Berechtigungen und Sicherheit
9. Sparsam und effektiv arbeiten
10. Aktualisieren
11. Wichtige Befehle
12. Fehlerbehebung
13. Deinstallieren
14. Checkliste

So lesen Sie diese Anleitung

- Arbeiten Sie die Kapitel **der Reihe nach** ab. Bei Installation, Anmeldung und Einstellungen ist jeweils angegeben, wie Sie prüfen, ob der Schritt geklappt hat.
- Dunkel hinterlegte Kästen sind Befehle. Die Beschriftung darüber sagt, wo sie eingegeben werden: **PowerShell**, **CMD** (Eingabeaufforderung) oder **Claude Code** (im laufenden Programm, beginnt mit `/`).
- `%USERPROFILE%` steht für Ihren Benutzerordner, z. B. `C:\Users\Anna`. In der Claude-Dokumentation wird derselbe Ordner `~` geschrieben.

KAPITEL 1

Was Sie brauchen

Voraussetzung	Mindestanforderung
Betriebssystem	Windows 10 Version 1809 oder neuer, Windows 11, Windows Server 2019 oder neuer
Hardware	mindestens 4 GB Arbeitsspeicher, Prozessor x64 oder ARM64
Internet	dauerhafte Verbindung (Claude arbeitet über die Server von Anthropic)
Konto	ein **kostenpflichtiges** Konto: Claude Pro, Max, Ultra, Team oder Enterprise – oder ein Claude-Console-Konto (Abrechnung nach Verbrauch). **Der kostenlose Claude-Tarif enthält Claude Code nicht.**
Rechte	Ein normales Benutzerkonto genügt. Administratorrechte sind für die Installation **nicht** nötig.
Empfohlen	Git für Windows (Kapitel 2) und ein Texteditor, z. B. Visual Studio Code oder Notepad++

Wählen Sie Ihre Oberfläche

Claude Code gibt es in mehreren Formen. Alle nutzen dieselben Einstellungs- und Regeldateien, die Sie in dieser Anleitung anlegen.

- **Terminal (Kommandozeile)** – der vollständige Funktionsumfang. Diese Anleitung beschreibt vor allem diesen Weg.
- **Claude Desktop-App** (Reiter *Code*) – grafisch, ohne Terminal. Download: claude.com/download.
- **Erweiterung für Visual Studio Code oder JetBrains** – Claude direkt im Editor.

> **Empfehlung**
> Installieren Sie zuerst die Terminal-Version (Kapitel 3). Damit stehen Diagnosebefehle wie `claude doctor` zur Verfügung, auch wenn Sie später hauptsächlich die Desktop-App oder VS Code nutzen.

Terminal öffnen

1. Drücken Sie `Win`, tippen Sie `PowerShell` und wählen Sie **Windows PowerShell** (nicht „Als Administrator ausführen“).
2. Sie erkennen PowerShell an der Eingabezeile `PS C:\Users\IhrName>`. Steht dort nur `C:\Users\IhrName>` ohne `PS`, sind Sie in der Eingabeaufforderung (CMD).

KAPITEL 2

Git für Windows installieren

Git ist offiziell *optional*, aber **empfohlen**: Es liefert die „Git Bash“, die Claude Code für sein Bash-Werkzeug und das Monitor-Werkzeug nutzt. Ohne Git verwendet Claude ausschließlich PowerShell. Außerdem erlaubt Git, jede Änderung von Claude nachzuvollziehen und rückgängig zu machen.

Variante A: mit WinGet (schnell)

```powershell
winget install --id Git.Git -e --source winget
```

Variante B: per Installer

1. Öffnen Sie git-scm.com/downloads/win und laden Sie den Installer für „64-bit Git for Windows Setup“.
2. Starten Sie die Datei und übernehmen Sie in allen Dialogen die **Standardeinstellungen** (Installationsordner `C:\Program Files\Git`).

Prüfen und Git einrichten

Schließen Sie PowerShell und öffnen Sie ein **neues** Fenster. Dann:

```powershell
git --version
```

Es erscheint z. B. `git version 2.5x.x.windows.1`. Hinterlegen Sie anschließend einmalig Name und E-Mail für Ihre Git-Einträge:

```powershell
git config --global user.name "Vorname Nachname"
git config --global user.email "ihre@adresse.de"
```

KAPITEL 3

Claude Code installieren

Wählen Sie **genau eine** der folgenden Varianten. Mehrere Installationen nebeneinander führen zu Verwechslungen.

Variante A (empfohlen): nativer Installer in PowerShell

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

- Der Befehl zeigt während des Downloads **keinen Fortschritt** – warten Sie, bis die Eingabezeile wieder erscheint.
- Das Programm landet in `%USERPROFILE%\.local\bin\claude.exe`.
- Diese Variante **aktualisiert sich automatisch** im Hintergrund.

Variante B: nativer Installer in CMD

CMD
```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del install.cmd
```

Variante C: WinGet

PowerShell
```
winget install Anthropic.ClaudeCode
```

Hinweis zu WinGet

WinGet-Installationen aktualisieren sich **nicht** automatisch. Sie müssen regelmäßig `winget upgrade Anthropic.ClaudeCode` ausführen (siehe Kapitel 10).

Typische Verwechslungen

- Meldung `The token '&&' is not a valid statement separator` → Sie haben den CMD-Befehl in PowerShell eingegeben. Nehmen Sie Variante A.
- Meldung `'irm' is not recognized …` → Sie haben den PowerShell-Befehl in CMD eingegeben. Nehmen Sie Variante B.

Sicherheit

Laden Sie Claude Code ausschließlich über die oben genannten Adressen (`claude.ai`, WinGet-Paket `Anthropic.ClaudeCode`) oder `claude.com/download`. Das Windows-Programm ist von „Anthropic, PBC" digital signiert. Kontrolle (im Ordner `%USERPROFILE%\.local\bin`): `Get-AuthenticodeSignature .\claude.exe` muss den Status `Valid` zeigen.

Eine Installation über npm (`npm install -g @anthropic-ai/claude-code`, benötigt Node.js 22+) ist ebenfalls möglich, auf einem Windows-PC aber nicht nötig. Verwenden Sie dabei niemals Administratorrechte.

KAPITEL 4

Installation prüfen

1. **Schließen** Sie das Terminal und öffnen Sie ein **neues** PowerShell-Fenster (sonst kennt Windows den neuen Programmpfad noch nicht).
2. Versionsprüfung:

   ```powershell
   claude --version
   ```

   Erwartet wird eine Versionsnummer wie `2.1.xxx (Claude Code)`.
3. Ausführliche Diagnose (ändert nichts, liest nur):

   ```powershell
   claude doctor
   ```

   Der Bericht zeigt Installationszustand, Fehler in Einstellungsdateien, Auto-Update-Status und Warnungen mit Lösungsvorschlägen.

Wenn „claude“ nicht gefunden wird

Der Ordner `%USERPROFILE%\.local\bin` fehlt im Suchpfad (PATH). So tragen Sie ihn ein:

1. `Win` + `R` drücken, `rundll32 sysdm.cpl,EditEnvironmentVariables` eingeben, **OK**.
2. Im oberen Bereich **Benutzervariablen** den Eintrag `Path` markieren → **Bearbeiten**.
3. **Neu** → `%USERPROFILE%\.local\bin` eintragen → alle Fenster mit **OK** schließen.
4. Neues PowerShell-Fenster öffnen und `claude --version` wiederholen.

Wenn Claude die Git Bash nicht findet

Tragen Sie den Pfad in Ihre persönliche Einstellungsdatei `%USERPROFILE%\.claude\settings.json` ein (Aufbau siehe Kapitel 8):

```settings.json
{
  "env": {
    "CLAUDE_CODE_GIT_BASH_PATH": "C:\\Program Files\\Git\\bin\\bash.exe"
  }
}
```

In JSON-Dateien muss jeder Backslash **doppelt** geschrieben werden.

KAPITEL 5

Anmelden

1. Starten Sie Claude Code im Terminal:

```
PowerShell
claude
```

2. Beim ersten Start wählen Sie ein Farbschema (mit Pfeiltasten und `Enter`).
3. Wählen Sie die Anmeldeart:
   - **Claude-Konto mit Abo** (Pro, Max, Ultra, Team, Enterprise) – für die meisten Nutzer richtig, feste Monatskosten.
   - **Claude Console** – Abrechnung nach Verbrauch über platform.claude.com.
4. Der Browser öffnet sich. Melden Sie sich dort an und bestätigen Sie den Zugriff. Danach kehren Sie ins Terminal zurück.
5. Prüfen Sie die Anmeldung im laufenden Claude Code:

```
Claude Code
/status
```

   Dort sehen Sie Konto, Modell und geladene Einstellungsdateien.
6. Beenden: `/exit` eingeben oder zweimal `Strg` + `C`.

API-Schlüssel schützen

Wer mit einem API-Schlüssel arbeitet (Umgebungsvariable `ANTHROPIC_API_KEY`), schreibt ihn **nie** in Projektdateien, CLAUDE.md oder `.claude/settings.json`. Ist die Variable gesetzt, fragt Claude Code beim Start, ob der Schlüssel verwendet werden soll – bei einem Abo antworten Sie mit Nein, sonst wird nach Verbrauch abgerechnet.

Konto wechseln: `/logout` und anschließend `/login`.

KAPITEL 6

Arbeitsbereich zuweisen

Claude Code arbeitet in dem Ordner, in dem Sie es starten (dem **Arbeitsverzeichnis**). Dateien darin darf es ohne Rückfrage *lesen*; Änderungen und Befehle laufen über die Berechtigungen (Kapitel 8). Der Arbeitsbereich ist damit Ihre wichtigste Sicherheitsgrenze.

Schritt 1: Eine feste Ordnerstruktur anlegen

Legen Sie einen eigenen Hauptordner an und darin je Projekt einen Unterordner, z. B.:

Beispiel

```
C:\ClaudeCode\
├── Website\
├── Rechnungstool\
└── Notizen\
```

PowerShell

```
New-Item -ItemType Directory -Force C:\ClaudeCode\Website
```

Nie als Arbeitsbereich verwenden

`C:\`, `C:\Windows`, `C:\Program Files`, Ihren gesamten Benutzerordner `%USERPROFILE%`, den Desktop oder Ordner mit Zugangsdaten, Steuer- und Bankunterlagen. Claude könnte sonst Dateien lesen, die es für die Aufgabe nicht braucht – das kostet Tokens und ist ein Datenschutzrisiko.

Schritt 2: Claude im Projektordner starten

PowerShell

```
cd C:\ClaudeCode\Website
claude
```

Beim ersten Start in einem Ordner erscheint die Frage, ob Sie diesem Ordner **vertrauen**. Bestätigen Sie nur bei eigenen Ordnern. Bei fremden Projekten (z. B. aus dem Internet geladen) können darin enthaltene Einstellungsdateien Befehle freigeben – erst prüfen, dann vertrauen.

Schritt 3 (empfohlen): Git im Projekt einschalten

PowerShell

```
cd C:\ClaudeCode\Website
git init
```

Damit lässt sich jede Änderung mit `git diff` ansehen und mit Git zurückholen. Claude Code hat zusätzlich eigene Rücksprungpunkte (`/rewind` oder zweimal `Esc`), Git ist aber die verlässlichere Sicherung.

Weitere Ordner freigeben

Wann	Wie	Wirkung
Beim Start	`claude --add-dir C:\ClaudeCode\Gemeinsam`	nur für diese Sitzung
In der Sitzung	`/add-dir C:\ClaudeCode\Gemeinsam`	nur für diese Sitzung
Dauerhaft	`"additionalDirectories"` in `settings.json` (Kapitel 8)	bei jedem Start
Ordner wechseln	`/cd C:\ClaudeCode\Rechnungstool`	Gespräch bleibt, neuer Ordner wird Hauptarbeitsbereich

Netzwerkfreigaben (`\\server\freigabe`) lassen sich nicht direkt hinzufügen. Verbinden Sie sie zuerst als Laufwerk (z. B. `Z:`) und starten Sie mit `claude --add-dir Z:\` .

In der Desktop-App

Im Reiter **Code** wählen Sie beim Anlegen einer neuen Sitzung den Projektordner aus. Es gelten dieselben Regeln wie oben: immer einen konkreten Projektordner, nie ein ganzes Laufwerk.

KAPITEL 7

Regeln verwalten (CLAUDE.md)

Jede Sitzung beginnt ohne Erinnerung. Dauerhafte Anweisungen schreiben Sie deshalb in **CLAUDE.md**-Dateien – einfache Textdateien im Markdown-Format, die Claude bei jedem Start liest.

> **Wichtig zu verstehen**
> CLAUDE.md ist **Kontext, keine Sperre**. Claude befolgt die Regeln in aller Regel, technisch erzwungen werden sie aber nicht. Was *niemals* passieren darf (z. B. Dateien löschen, Zugangsdaten lesen), regeln Sie zusätzlich über Berechtigungen in Kapitel 8.

Die vier Ebenen

Ebene	Ort (Windows)	Gilt für
Persönlich	`%USERPROFILE%\.claude\CLAUDE.md`	Sie, in allen Projekten
Projekt	`<Projektordner>\CLAUDE.md` oder `<Projektordner>\.claude\CLAUDE.md`	alle, die in diesem Projekt arbeiten (wird mit Git geteilt)
Projekt, privat	`<Projektordner>\CLAUDE.local.md`	nur Sie, nur dieses Projekt (in `.gitignore` eintragen)
Organisation	`C:\Program Files\ClaudeCode\CLAUDE.md`	alle Benutzer des PCs; wird von der IT verwaltet

Alle gefundenen Dateien werden **zusammengefügt**, nicht gegenseitig ersetzt. CLAUDE.md-Dateien in Ordnern *oberhalb* des Startordners werden beim Start geladen, solche in Unterordnern erst, wenn Claude dort arbeitet. Widersprechen sich zwei Regeln, wählt Claude möglicherweise willkürlich – halten Sie die Dateien widerspruchsfrei.

Schritt 1: Persönliche Regeln anlegen

1. Claude Code starten und eingeben:

   Claude Code
   ```
   /memory
   ```

2. Den Eintrag für die **persönliche** CLAUDE.md wählen. Existiert sie noch nicht, wird sie angelegt und im Editor geöffnet.

3. Kurze, eindeutige Regeln eintragen, z. B.:

   %USERPROFILE%\.claude\CLAUDE.md
   ```
   # Persönliche Regeln

   - Antworte immer auf Deutsch.
   - Verwende in deutschem Text echte Umlaute (ä, ö, ü, ß).
   - Frage nach, bevor du Dateien löschst oder umbenennst.
   - Nenne nach jeder Änderung alle geänderten Dateien.
   - Lege neue Projekte nur unter C:\ClaudeCode an.
   ```

4. Speichern. Die Regeln gelten ab der nächsten Nachricht bzw. Sitzung.

Schritt 2: Projektregeln anlegen

1. Im Projektordner Claude starten und eingeben:

```
Claude Code
/init
```

   Claude untersucht das Projekt und schreibt eine erste CLAUDE.md mit Aufbau, Build- und Testbefehlen. Gibt es schon eine, schlägt `/init` Verbesserungen vor, statt sie zu überschreiben.

2. Die Datei lesen und um Dinge ergänzen, die Claude nicht selbst herausfinden kann, z. B.:

```
CLAUDE.md (Projekt)
# Projekt Website

## Aufbau
- PHP 8.3, keine Datenbank, Inhalte in data/*.json

## Regeln
- Jede Änderung: Versionsnummer in inc/version.php erhöhen.
- CSS/JS immer mit ?v=VERSION einbinden (Cache).
- Keine externen Bibliotheken ohne Rückfrage.

## Befehle
- Test: php -l datei.php
```

Gute Regeln schreiben

- **Kurz:** unter 200 Zeilen je Datei. Lange Dateien kosten bei *jeder* Anfrage Tokens und werden schlechter befolgt.
- **Konkret:** „Einrückung 4 Leerzeichen" statt „Code ordentlich formatieren".
- **Gegliedert:** Überschriften und Aufzählungen.
- **Nur Dauerhaftes:** Abläufe mit vielen Schritten gehören in einen Skill (lädt nur bei Bedarf), nicht in CLAUDE.md.

Regeln aufteilen: Ordner .claude\rules

Für größere Projekte legen Sie einzelne Regeldateien in `<Projektordner>\.claude\rules\` ab (z. B. `php.md`, `tests.md`). Mit einer Kopfzeile `paths` gilt eine Regel nur, wenn Claude passende Dateien bearbeitet – das spart Kontext:

```
.claude\rules\php.md
---
paths:
  - "**/*.php"
---

# PHP-Regeln
- declare(strict_types=1) am Dateianfang
- Superglobals nie direkt verwenden
```

Persönliche Regeldateien für alle Projekte gehören nach `%USERPROFILE%\.claude\rules\`.

Andere Dateien einbinden

In einer CLAUDE.md bindet `@pfad/zur/datei.md` eine weitere Datei ein (relativ zur CLAUDE.md, maximal vier Ebenen tief). Pfade mit Leerzeichen schreiben Sie mit Backslash vor jedem Leerzeichen. Eingebundene Dateien

kosten genauso Tokens wie direkt geschriebener Text.

Automatisches Gedächtnis (Auto Memory)

- Sagen Sie Claude „Merke dir: …“, speichert es die Notiz selbst in `%USERPROFILE%\.claude\projects\<projekt>\memory\`. Die Übersichtsdatei `MEMORY.md` (erste 200 Zeilen bzw. 25 KB) wird bei jedem Start geladen.
- Soll etwas fest in CLAUDE.md stehen, sagen Sie ausdrücklich: „Trage das in CLAUDE.md ein.“
- Ein- und ausschalten sowie den Ordner öffnen: über `/memory`.
- Alle Notizen sind normale Textdateien – prüfen Sie sie gelegentlich und löschen Sie Veraltetes.

Prüfen, was geladen ist

Claude Code

```
/memory
/context
```

`/memory` listet alle Regeldateien, `/context` zeigt, wie viel Platz sie im Kontext belegen.

Praxisbeispiel 1: Eigene Entwicklungsrichtlinien einbinden

Ein Regelwerk, das für alle Projekte unterhalb eines Hauptordners gilt, legen Sie als eigene Datei ab und binden sie über eine CLAUDE.md im Hauptordner ein. Beispiel: Die JoomION-Entwicklungsrichtlinien liegen in `C:\ClaudeCode\JOOMION-REGELN.md`.

1. Im Hauptordner `C:\ClaudeCode` eine Datei `CLAUDE.md` mit folgendem Inhalt anlegen:

   C:\ClaudeCode\CLAUDE.md

   ```
   # Verbindliche Regeln
   @JOOMION-REGELN.md
   ```

2. Claude wie gewohnt in einem Projektordner darunter starten, z. B. `C:\ClaudeCode\dist`. CLAUDE.md-Dateien in **übergeordneten** Ordnern werden beim Start automatisch geladen, die Regeln gelten also in jedem Unterordner.

3. Mit `/memory` prüfen, dass `C:\ClaudeCode\CLAUDE.md` in der Liste erscheint.

Inhalt der Beispieldatei (rund 100 Zeilen, also unter der empfohlenen Grenze von 200):

C:\ClaudeCode\JOOMION-REGELN.md

```
JOOMION – ENTWICKLUNGSRICHTLINIEN

Verbindlich für alle bestehenden und neuen JoomION-Erweiterungen sowie für sämtliche im Rahmen von JoomION
erstellten Komponenten, Module, Plugins, Pakete, Webseiten, Dokumentationen, Anleitungen, FAQs, Word- und PDF-
Dokumente, Präsentationen, Grafiken, Logos, Screenshots, QR-Codes, Audio-Dateien, Videos, Reels, Social-Media-
Inhalte, Bücher und sonstige digitale oder gedruckte Medien.
Gültig ab sofort bis zur Veröffentlichung einer neueren Version.

# 1. Oberste Grundsatz- und Sicherheitsregeln

- Vor Beginn jeder Entwicklungsaufgabe sind diese Regeln als verbindliche Arbeitsgrundlage anzuwenden.
- Strikte Einhaltung aller Joomla-5- und Joomla-6-Vorgaben.
- Strikte Einhaltung von PSR-12 sowie Joomla Coding Standards.
- Ausschließlich Joomla-konforme APIs verwenden.
- Kein Schadcode, keine Backdoors, keine versteckten Funktionen.
- Verboten sind insbesondere: eval(), exec(), shell_exec(), system(), passthru(), proc_open().
```

```
- Jede PHP-Datei muss gegen direkten Zugriff geschützt werden:
  defined('_JEXEC') or die;
- SQL-Abfragen ausschließlich über Joomla DatabaseQuery.
- Keine SQL-Stringverkettungen.
- Alle Parameter müssen gebunden werden.
- Schutz gegen SQL-Injection, XSS, CSRF, Path Traversal, Directory Traversal und File Inclusion.
- ACL-Prüfungen verpflichtend.
- CSRF-Tokens verpflichtend.
- Uploads nur über Whitelists zulassen.
- realpath()-Prüfung bei Dateioperationen verpflichtend.
- Ausgabe sämtlicher Benutzereingaben escapen.
- Keine Debug-Ausgaben im Produktivcode.
- Logging ausschließlich über Joomla Logging Framework.
- Antworten und Kommentare ausschließlich in deutscher Sprache.
- Ehrlich antworten. Nicht schätzen. Nicht raten. Nichts verheimlichen.
- Bei Unklarheiten immer nachfragen.
- Keine eigenständigen fachlichen Entscheidungen.
- Vorhandene Versionen niemals überschreiben.
- Jede Änderung erzeugt eine neue Version.

# 2. Arbeitsverzeichnis

Ausschließlich innerhalb des Verzeichnisbaums C:\ClaudeCode\dist arbeiten.

# 3. Erweiterungsarchitektur & Backend-Pflichten

- Jede Komponente benötigt ein Dashboard als Standardansicht.
- Das Dashboard ist die erste Seite nach dem Aufruf.
- JoomION-Balken mit Logo, Name, Version, Datum und Link auf https://www.joomion.de.
- Komponenten zentral über den Joomla Administrator Dispatcher laden.
- Module und Plugins erhalten ebenfalls einen JoomION-Balken.
- Wechsel zwischen Komponenten, Modulen und Plugins muss über Buttons möglich sein.

# 4. Coding Standards

- PSR-12 verpflichtend.
- Joomla Coding Standards verpflichtend.
- Namespace-Pflicht.
- Dependency Injection bevorzugen.
- MVC-Struktur strikt einhalten.
- Keine Geschäftslogik in Views.
- Keine Datenbanklogik in Views.
- Keine Sprachtexte direkt im Code.
- declare(strict_types=1); verwenden sofern kompatibel.

# 5. Sprachdateien

- Alle sichtbaren Texte aus Sprachdateien.
- Deutsch und Englisch verpflichtend.
- Keine fest codierten Texte.

# 6. Barrierefreiheit

- aria-labels verwenden.
- Formulare und Buttons korrekt beschriften.
- Alternativtexte für Bilder.
- Grundlegende WCAG-Konformität.

# 7. Dokumentation, FAQ und Hilfeseiten

- Hilfe-/Anleitungsseite in Deutsch und Englisch.
- Anleitung und FAQ als Word, PDF und HTML.
- HTML ohne html-, body- und title-Tags.
- Ablage unter C:\ClaudeCode\dist\Anleitungen und FAQ\<Erweiterung>.
```

```
# 8. Versionierung, Builds und Updates

- Versionssprung bei jeder Änderung.
- Datum immer aktualisieren.
- ZIP-Dateien in Updateserver\Downloads und ZIPs speichern.
- Update-XML und Prüfsummen erzeugen.
- Joomla Updatefunktion integrieren.

# 9. Sicherheit & Audits

- Security Audit verpflichtend.
- Joomla-API-Audit verpflichtend.
- Joomla 5/6 Audit verpflichtend.
- PSR-12 Audit verpflichtend.
- Critical und High Fehler müssen vor Veröffentlichung 0 sein.
- Audit-PDFs in C:\ClaudeCode\dist\Audits speichern.

# 10. Verzeichnisstruktur

Arbeitsverzeichnis ist ausschließlich C:\ClaudeCode\dist mit der festgelegten Unterordnerstruktur.

# 11. Qualitätsziel

Jede JoomION-Erweiterung muss Joomla 5/6 kompatibel, PSR-12 konform, sicherheitsgeprüft, mehrsprachig,
updatefähig, dokumentiert und auditierbar sein.
```

Was an diesem Beispiel gut ist – und was man ergänzen sollte

- **Gut:** klare Gliederung mit Überschriften, kurze Aufzählungspunkte, konkrete Verbote (z. B. die Liste gefährlicher PHP-Funktionen) und feste Ablageorte.
- **Regel „Nur in C:\ClaudeCode\dist arbeiten" zusätzlich technisch absichern:** Claude dort starten (Kapitel 6). Wer eine harte Sperre möchte, setzt in der persönlichen `settings.json` zusätzlich `"permissions": { "blockWritesOutsideWorkingDirectories": true }` – dann verweigern die Datei-Werkzeuge Schreibzugriffe außerhalb der Arbeitsordner (benötigt Claude Code ab Version 2.1.314; Shell-Befehle erfasst die Sperre nicht).
- **Ausführliche Abläufe nicht in die Regeldatei:** Ein vollständiger Audit-Auftrag mit 20 Prüfpunkten würde bei *jeder* Anfrage mitgeladen. Er gehört in einen Skill – siehe Praxisbeispiel 2.

Praxisbeispiel 2: Wiederkehrenden Auftrag als Skill hinterlegen (PSR-12-Audit)

Ein **Skill** ist ein gespeicherter Auftrag, den Sie mit `/name` aufrufen. Anders als CLAUDE.md wird sein Inhalt **erst beim Aufruf** geladen – lange Prüfaufträge kosten also nichts, solange Sie sie nicht benutzen. Ablage:

Gültigkeit	Ort
persönlich, alle Projekte	`%USERPROFILE%\.claude\skills\<name>\SKILL.md`
nur ein Projekt	`<Projektordner>\.claude\skills\<name>\SKILL.md`

1. Ordner `%USERPROFILE%\.claude\skills\joomla-audit` anlegen:

   PowerShell
   ```
   New-Item -ItemType Directory -Force "$env:USERPROFILE\.claude\skills\joomla-audit"
   ```

2. Darin die Datei `SKILL.md` mit folgendem Inhalt speichern (UTF-8):

%USERPROFILE%\.claude\skills\joomla-audit\SKILL.md

```
---
name: joomla-audit
description: Vollständiger PSR-12-, Joomla-5/6- und Sicherheits-Audit einer Joomla-Erweiterung
argument-hint: "[Ordner der Erweiterung]"
disable-model-invocation: true
---

Führe einen vollständigen, detaillierten und technisch korrekten PSR-12-Audit
der Joomla-Erweiterung im Ordner $ARGUMENTS durch. Prüfe zusätzlich alle
Joomla Coding Standards (Joomla-PSR12), alle Joomla-5- und Joomla-6-Audit-Regeln
sowie alle sicherheitsrelevanten Aspekte gemäß Joomla Security Guidelines.

Vorgehen:
1. Lies alle Dateien der Erweiterung selbst ein. Fehlt etwas, das laut
   Manifest vorhanden sein müsste, melde es ausdrücklich.
2. Führe zusätzlich PHP_CodeSniffer aus:
   php C:\ClaudeCode\dist\tools\phpcs.phar --standard=C:\ClaudeCode\dist\tools\joomion-psr12.xml --
extensions=php $ARGUMENTS
3. Analysiere jede Datei einzeln und strukturiert. Liste je Datei alle Verstöße:
   - PSR-12-Fehler
   - Joomla-Coding-Standard-Fehler
   - Namespace-Fehler
   - Klassen- und Dateistruktur-Fehler
   - DocBlock-Fehler
   - Fehler bei use-Statements
   - Fehler bei Einrückungen, Spacing, Leerzeilen
   - Fehler bei HTML-Ausgabe in PHP-Klassen
   - Deprecated-Code für Joomla 5/6
   - API-Verstöße gegen Joomla 5/6
   - MVC-Fehler (Model, View, Controller)
   - Fehler in Helper-Klassen
   - Fehler in Layout-Overrides
   - Fehler in Manifest-XML
   - Fehler in Routing, Services, Factory, Dependency Injection
   - Fehler in Language-Files
   - Fehler in SQL-Install/Update-Skripten
   - Security-Fehler (XSS, CSRF, SQL-Injection, ACL, Token-Handling,
     Input-Filter, Output-Escaping)
   - Barrierefreiheit (aria-label, Beschriftungen, Alternativtexte);
     das CSS-Design dabei nicht verändern
4. Erstelle eine Gesamtbewertung mit Priorisierung nach Kritikalität
   (Critical, High, Medium, Low).
5. Gib zu jedem Fehler konkrete Korrekturvorschläge und, wo möglich,
   vollständige Code-Beispiele nach Joomla-PSR12-Standard.
6. Verwende ausschließlich Joomla-5- und Joomla-6-konforme Syntax,
   Namespaces und Best Practices.
7. Ändere während des Audits KEINE Dateien. Korrekturen erst nach meiner
   ausdrücklichen Freigabe, danach Versionssprung gemäß JOOMION-REGELN.md.
8. Speichere den Bericht als PDF in C:\ClaudeCode\dist\Audits.
```

3. Claude Code neu starten (oder eine neue Sitzung öffnen) und den Skill aufrufen:

Claude Code

```
/joomla-audit C:\ClaudeCode\dist\ZIPs\MeineErweiterung
```

`$ARGUMENTS` wird dabei durch den angegebenen Ordner ersetzt.

Was gegenüber einem frei eingetippten Prompt verbessert wurde

- **Prüfen und Ändern getrennt:** „Überprüfe und korrigiere“ würde Claude mitten im Audit Dateien ändern lassen – das widerspricht der Regel „Keine eigenständigen fachlichen Entscheidungen“ und „Jede Änderung erzeugt eine neue Version“. Punkt 7 legt fest: erst Bericht, dann Freigabe, dann Korrektur.
- **„Fehlende Dateien nachfordern“ ersetzt:** In Claude Code liest Claude den Ordner selbst. Sinnvoll ist stattdessen, Abweichungen zwischen Manifest und vorhandenen Dateien melden zu lassen.
- **Werkzeug statt Augenmaß:** PHP_CodeSniffer findet PSR-12-Verstöße zuverlässig und billiger, als wenn Claude jede Leerzeile selbst begutachtet. Claude konzentriert sich dann auf das, was ein Werkzeug nicht kann (Sicherheit, Joomla-API, MVC).
- `disable-model-invocation: true`**:** Der Audit startet nur, wenn Sie `/joomla-audit` eintippen – nie ungefragt. Zugleich wird die Beschreibung nicht in jede Sitzung geladen.
- **Normale Bindestriche:** Der Ursprungstext enthielt geschützte Bindestriche (-). Gewöhnliche Bindestriche vermeiden Probleme beim Kopieren und Durchsuchen.

Sparsam auditieren

Ein vollständiger Audit liest jede Datei der Erweiterung und ist eine der teuersten Anfragen überhaupt. Starten Sie ihn in einer frischen Sitzung (`/clear`), nicht am Ende eines langen Gesprächs. Bei großen Komponenten lohnt es sich, Ordner einzeln zu prüfen (z. B. erst `admin\src` , dann `site`).

KAPITEL 8

Berechtigungen und Sicherheit

Während CLAUDE.md Claude *bittet*, legen Berechtigungen technisch fest, was Claude darf. Sie stehen in **settings.json**-Dateien.

Die Einstellungsdateien

Datei	Gilt für	Zweck
`%USERPROFILE%\.claude\settings.json`	Sie, alle Projekte	Grundschutz, persönliche Vorlieben
`<Projekt>\.claude\settings.json`	alle im Projekt (mit Git geteilt)	gemeinsame Projektregeln
`<Projekt>\.claude\settings.local.json`	nur Sie, nur dieses Projekt	persönliche Ausnahmen; hier speichert Claude auch Ihre „Ja, nicht mehr fragen"-Antworten
verwaltete Einstellungen (IT)	ganze Organisation	Firmenrichtlinien, nicht überschreibbar

Rangfolge (höchste zuerst): verwaltete Einstellungen → Befehlszeile (`--settings`) → `settings.local.json` → Projekt-`settings.json` → persönliche `settings.json`. Listen wie `allow` und `deny` werden über alle Dateien **zusammengeführt**.

Wie Regeln ausgewertet werden

Es gibt drei Listen: `deny` (verbieten), `ask` (immer fragen), `allow` (ohne Rückfrage erlauben). Geprüft wird **immer in der Reihenfolge deny → ask → allow**; der erste Treffer entscheidet. Ein Verbot lässt sich also durch keine Erlaubnis aushebeln.

Regel	Bedeutung
`Bash(npm run build)`	genau dieser Befehl
`Bash(npm run *)`	alle Befehle, die mit `npm run` beginnen
`PowerShell(Remove-Item *)`	PowerShell-Befehl samt Aliassen (`del`, `rm` …)
`Read(./.env)`	Datei `.env` im Projektordner
`Read(~/Documents/*.pdf)`	Pfad ab Ihrem Benutzerordner
`Read(//c/Steuer/**)`	absoluter Pfad `C:\Steuer` mit allen Unterordnern. Windows-Pfade werden intern als `/c/…` geschrieben
`WebFetch(domain:example.com)`	Webabrufe von dieser Domain

Schritt 1: Sichere Grundeinstellung anlegen

Öffnen Sie `%USERPROFILE%\.claude\settings.json` im Editor (gibt es die Datei noch nicht, legen Sie sie an) und tragen Sie ein:

%USERPROFILE%\.claude\settings.json

```
{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "permissions": {
    "defaultMode": "default",
    "deny": [
      "Read(./.env)",
      "Read(./.env.*)",
      "Read(**/*.pem)",
      "Read(**/*.key)",
      "Read(~/.ssh/**)",
      "Bash(rm -rf *)",
      "PowerShell(Format-Volume *)"
    ],
    "ask": [
      "Bash(git push *)",
      "PowerShell(Remove-Item *)",
      "Bash(rm *)"
    ],
    "allow": [
      "Bash(git status)",
      "Bash(git diff *)",
      "Bash(git log *)"
    ]
  }
}
```

- Die Datei ist **striktes JSON**: keine Kommentare (`//`), kein Komma nach dem letzten Eintrag, Backslashes verdoppeln.
- Die Zeile `$schema` sorgt in VS Code für Autovervollständigung und Fehlerprüfung.
- Änderungen werden meist sofort übernommen. Kontrolle mit `/status` (Zeile „Setting sources“) und bei Fehlern mit `claude doctor`.

Grenzen von Befehlsregeln

Regeln wie `Bash(rm *)` greifen nicht, wenn derselbe Befehl über einen vollen Pfad oder verschachtelt (`bash -c "rm …"`) aufgerufen wird. Sie sind ein wirksamer Schutz gegen Versehen, aber kein Ersatz für Datensicherung. Wichtige Daten gehören nicht in den Arbeitsbereich.

Schritt 2: Dauerhafte Zusatzordner (optional)

settings.json – Ausschnitt

```
{
  "permissions": {
    "additionalDirectories": ["C:\\ClaudeCode\\Gemeinsam"]
  }
}
```

Berechtigungsmodi

Mit `Umschalt` + `Tab` wechseln Sie im laufenden Claude Code zwischen den Modi. Der aktuelle Modus steht unten in der Statuszeile.

Modus	Verhalten	Empfehlung
`default` (Manual)	fragt beim ersten Einsatz jedes Werkzeugs	**Standard für Einsteiger**
`plan`	liest und plant nur, ändert keine Dateien	vor größeren Aufgaben
`acceptEdits`	Dateiänderungen und einfache Dateibefehle im Arbeitsbereich ohne Rückfrage	wenn Sie die Änderungen per Git prüfen
`auto`	keine Routinefragen; ein Prüfmodell kontrolliert Befehle vorab	für Erfahrene
`dontAsk`	lehnt alles ab, was sonst fragen würde; nur Freigegebenes läuft	für feste, automatisierte Abläufe
`bypassPermissions`	überspringt fast alle Rückfragen	**nur in einer isolierten virtuellen Maschine**

Auf Rückfragen richtig antworten

- **Ja** – nur dieses eine Mal.
- **Ja, und nicht mehr fragen** – wird für Befehle und Webdomains dauerhaft in `.claude\settings.local.json` gespeichert. Nur für harmlose, wiederkehrende Befehle wählen.
- **Nein** – mit kurzer Begründung, was Claude stattdessen tun soll.

Alle aktuellen Regeln ansehen und bearbeiten: `/permissions`.

KAPITEL 9

Sparsam und effektiv arbeiten

Bei jeder Nachricht sendet Claude Code das **gesamte bisherige Gespräch** samt Regeln und gelesenen Dateien mit. Je länger und voller eine Sitzung ist, desto mehr Kontingent verbraucht jede einzelne Frage. Daraus ergeben sich die wichtigsten Sparregeln:

Die 10 wichtigsten Sparregeln

1. **Eine Aufgabe – eine Sitzung.** Vor einem neuen Thema `/clear`. Vorher mit `/rename Name` benennen, dann finden Sie die Sitzung später mit `/resume` wieder. `/clear` selbst kostet nichts.
2. **Präzise fragen.** „Füge in `login.php` eine Prüfung der E-Mail-Adresse ein" statt „Verbessere das Projekt". Vage Aufträge lösen breites Durchsuchen aus.
3. **Dateien und Fehler benennen.** Genaue Fehlermeldung, Dateiname, Zeile – Claude muss dann nicht suchen.
4. **Erst planen, dann bauen.** Bei größeren Aufgaben mit `Umschalt` + `Tab` in den Plan-Modus wechseln und den Plan freigeben. Das verhindert teure Irrwege.
5. **Früh stoppen.** Läuft Claude in die falsche Richtung: `Esc` drücken. Mit `/rewind` (oder zweimal `Esc`) zu einem früheren Stand zurück.
6. **Passendes Modell wählen.** Mit `/model`: Sonnet erledigt die meisten Programmieraufgaben gut und günstiger; Opus nur für schwierige Planung und Architektur; Haiku für einfache Aufgaben. Mit `/effort` lässt sich die Denktiefe für einfache Aufgaben senken.
7. **CLAUDE.md schlank halten** (unter 200 Zeilen). Spezialwissen in Skills oder pfadbezogene Regeln auslagern.
8. **Unbenutzte MCP-Server abschalten** mit `/mcp`. Vorhandene Kommandozeilenwerkzeuge (z. B. `git`, `gh`) sind sparsamer.
9. **Lange Ausgaben vermeiden.** Statt Claude eine 10 000-Zeilen-Logdatei lesen zu lassen: „Suche in `log.txt` nur Zeilen mit ERROR."
10. **Lange Pausen beachten.** Nach einer längeren Pause wird der Zwischenspeicher neu aufgebaut und das ganze Gespräch erneut verarbeitet. Nach der Pause lieber mit `/clear` und einer kurzen Zusammenfassung neu beginnen.

Verbrauch beobachten

Befehl	Zeigt
`/usage`	Verbrauch der Sitzung, Auslastung Ihres Abos und Sparempfehlungen (mit `f` lassen sich manche direkt umsetzen)
`/context`	wofür der Kontext gerade belegt ist (Regeln, Werkzeuge, Gespräch)
`/compact Fokus auf …`	fasst das Gespräch zusammen und behält das Genannte. Achtung: Das Zusammenfassen selbst ist eine große Anfrage – für einen echten Neuanfang ist `/clear` günstiger
`/insights`	Bericht über Ihre Arbeitsweise mit Verbesserungsvorschlägen (verbraucht selbst Kontingent)

Gute Arbeitsgewohnheiten

- Prüfziel mitgeben: „Danach muss `php -l` ohne Fehler laufen." So prüft Claude selbst.
- Schrittweise vorgehen: eine Datei ändern, testen, dann weiter.
- Nicht unnötig viele Sitzungen oder Hilfsagenten parallel laufen lassen – jede verbraucht eigenes Kontingent.
- Wiederkehrende Abläufe als Skill hinterlegen, statt sie jedes Mal neu zu erklären.

KAPITEL 10

Aktualisieren

Installationsart	Aktualisierung
Nativer Installer (Variante A/B)	automatisch im Hintergrund; sofort: `claude update`
WinGet (Variante C)	manuell: `winget upgrade Anthropic.ClaudeCode` (Claude Code vorher beenden)

Wer lieber eine erprobte, etwa eine Woche ältere Version möchte, setzt in der persönlichen `settings.json`:

settings.json – Ausschnitt

```
{
  "autoUpdatesChannel": "stable"
}
```

Ergebnis des letzten Updates: `claude doctor`.

KAPITEL 11

Wichtige Befehle

Im Terminal	Wirkung
`claude`	Claude Code im aktuellen Ordner starten
`claude --continue`	letzte Sitzung in diesem Ordner fortsetzen
`claude --resume`	frühere Sitzung auswählen
`claude --add-dir <Ordner>`	zusätzlichen Ordner freigeben
`claude --version` / `claude doctor` / `claude update`	Version, Diagnose, Update

In Claude Code	Wirkung
`/help`	alle Befehle anzeigen
`/init`	Projekt-CLAUDE.md erzeugen
`/memory`	Regeldateien und Gedächtnis öffnen
`/permissions`	Berechtigungsregeln ansehen und ändern
`/config`	Einstellungsmenü (Farbschema, Standardmodell …)
`/status`	Konto, Modell, geladene Einstellungsdateien
`/model` · `/effort`	Modell bzw. Denktiefe wählen
`/clear` · `/compact`	neu beginnen bzw. zusammenfassen
`/rename` · `/resume`	Sitzung benennen bzw. wieder aufnehmen
`/rewind`	Gespräch und Dateien auf früheren Stand zurücksetzen
`/context` · `/usage`	Kontextbelegung bzw. Verbrauch
`/add-dir` · `/cd`	Ordner hinzufügen bzw. wechseln
`/mcp`	MCP-Server verwalten
`/login` · `/logout` · `/exit`	An-, Abmelden, Beenden

Taste	Wirkung
`Umschalt` + `Tab`	Berechtigungsmodus wechseln (u. a. Plan-Modus)
`Esc`	laufende Aktion abbrechen
`Esc` `Esc`	zu einem früheren Stand zurückspringen
`Strg` + `C` (zweimal)	Claude Code beenden

KAPITEL 12

Fehlerbehebung

Problem	Lösung
`claude` wird nicht erkannt	neues Terminal öffnen; sonst PATH ergänzen (Kapitel 4)
`'irm' is not recognized`	Sie sind in CMD – PowerShell verwenden oder CMD-Befehl nehmen
`The token '&&' is not a valid …`	Sie sind in PowerShell – PowerShell-Befehl nehmen
Git Bash wird nicht gefunden	`CLAUDE_CODE_GIT_BASH_PATH` setzen (Kapitel 4)
„Settings Error“ beim Start	JSON-Fehler in einer settings.json (Komma, Kommentar, einfacher Backslash). Die Meldung nennt die Datei; Details mit `claude doctor`
Regel wird nicht befolgt	`/memory` prüfen, ob die Datei geladen ist; Regel konkreter formulieren; Widersprüche entfernen; harte Grenzen über Berechtigungen setzen
Anmeldung klappt nicht	`/logout` , dann `/login` ; prüfen, ob das Konto ein kostenpflichtiger Tarif ist
„You've hit your session limit“	Kontingent des Abos erschöpft; die Meldung nennt den Zeitpunkt der Freigabe

KAPITEL 13

Deinstallieren

Native Installation:

```powershell
Remove-Item -Path "$env:USERPROFILE\.local\bin\claude.exe" -Force
Remove-Item -Path "$env:USERPROFILE\.local\share\claude" -Recurse -Force
```

WinGet-Installation:

```powershell
winget uninstall Anthropic.ClaudeCode
```

Vorsicht

Der Ordner `%USERPROFILE%\.claude` und die Datei `%USERPROFILE%\.claude.json` enthalten Ihre Regeln, Einstellungen, Gedächtnisnotizen und den Sitzungsverlauf. Löschen Sie sie nur, wenn Sie wirklich alles entfernen wollen – vorher sichern. Die Desktop-App und die VS-Code-Erweiterung legen den Ordner sonst neu an.

KAPITEL 14

Checkliste

- ☐ Kostenpflichtiges Claude-Konto vorhanden
- ☐ Git für Windows installiert, `git --version` funktioniert, Name und E-Mail gesetzt
- ☐ Claude Code mit genau einer Methode installiert
- ☐ `claude --version` und `claude doctor` ohne Fehler
- ☐ Angemeldet, `/status` zeigt das richtige Konto
- ☐ Hauptordner (z. B. `C:\ClaudeCode`) mit einem Unterordner je Projekt angelegt
- ☐ Claude wird immer im Projektordner gestartet, nie in `C:\` oder im Benutzerordner
- ☐ Im Projekt `git init` ausgeführt
- ☐ Persönliche CLAUDE.md mit kurzen Grundregeln angelegt
- ☐ Projekt-CLAUDE.md mit `/init` erstellt und ergänzt
- ☐ Eigenes Regelwerk (z. B. `JOOMION-REGELN.md`) per `@`-Import eingebunden, `/memory` zeigt es an
- ☐ Lange, wiederkehrende Aufträge (z. B. Audit) als Skill angelegt statt in CLAUDE.md
- ☐ Persönliche `settings.json` mit deny-/ask-Regeln angelegt, `/status` zeigt sie an
- ☐ Arbeitsweise: eine Aufgabe je Sitzung, `/clear`, präzise Aufträge, Plan-Modus, passendes Modell

CLAUDE CODE · PRAXISANLEITUNG

Vom leeren PC zum sicheren, sparsamen Arbeitsplatz mit Claude Code.

Claude Code liest Ihre Dateien, schreibt Programmcode und führt Befehle aus. Damit das sicher, nachvollziehbar und ohne unnötigen Verbrauch geschieht, braucht es eine saubere Einrichtung.

Diese Anleitung führt Schritt für Schritt durch alles, was dafür nötig ist – mit jedem Befehl zum Abtippen und einer Prüfung nach jedem wichtigen Schritt.

- Installation unter Windows 10 und 11 – mit Git, ohne Administratorrechte
- Arbeitsbereich festlegen: welche Ordner Claude sehen darf und welche nie
- Regeln mit CLAUDE.md, Regelordnern und Skills dauerhaft verankern
- Berechtigungen: was Claude erlaubt, was verboten ist – mit fertiger Vorlage
- Zehn Regeln, um Kontingent und Kosten spürbar zu senken
- Praxisbeispiele: eigenes Regelwerk und ein Audit-Auftrag als Skill

Roland H. Löffler

Stand: 5. November 2026 · Unabhängige Anleitung, keine offizielle Veröffentlichung von Anthropic.
Claude und Claude Code sind Marken der Anthropic, PBC.