Claude Code mit Codex verbinden – Anleitung

Inhalt

Warum sinnvoll? Einsatzgebiete

Claude Code und Codex CLI sind zwei unabhängige, konkurrierende Coding-Agenten – Claude Code von Anthropic, Codex CLI von OpenAI. Beide arbeiten direkt im Terminal mit einem Projekt, lesen und verändern Dateien und führen Shell-Befehle aus.

Der Grundgedanke, sie zu verbinden, ist nicht, eines der beiden Tools zu ersetzen, sondern ein zweites, unabhängiges Modell als Kontrollinstanz oder Zweitmeinung hinzuzuziehen, ohne das Terminal zu wechseln. Ein Modell schreibt Code, ein anderes Modell mit anderer Trainingsbasis und anderen blinden Flecken prüft ihn.

Konkrete Einsatzgebiete:

- **Code-Review durch ein zweites Modell.** Claude baut ein Feature, Codex (GPT) schaut sich den Diff an und meldet Probleme, ohne selbst etwas zu verändern.
- **Adversariales Review.** Codex bekommt gezielt den Auftrag, Designentscheidungen infrage zu stellen statt sie nur zu bestätigen.
- **Aufgaben abgeben.** Wenn Claude bei einem Bug nicht weiterkommt, lässt sich die Aufgabe an Codex übergeben oder an Codex überleiten.
- **Direkte Zweitmeinung per MCP.** Claude kann Codex als Werkzeug aufrufen und dessen Antwort in die eigene Analyse einbeziehen, ohne den Kontext zu verlassen.
- **Getrennte Kontingente nutzen.** Codex-Anfragen laufen über das ChatGPT- bzw. OpenAI-API-Kontingent, nicht über das Claude-Abo – bei knappen Claude-Limits eine zusätzliche Kapazität.

Ein Hinweis zur Einordnung: Die oft zu lesende Behauptung, ein zweites Modell finde grundsätzlich Fehler, die das erste übersieht, ist eine in der Praxis verbreitete Beobachtung Dritter, aber keine offiziell von Anthropic bestätigte Aussage. Es gibt keine offizielle Anthropic-Quelle, die das als Empfehlung ausspricht. Als praktische Erfahrung vieler Entwickler ist der Ansatz trotzdem plausibel: Unterschiedliche Modelle mit unterschiedlicher Trainingsbasis machen tendenziell unterschiedliche Fehler und übersehen unterschiedliche Dinge.

Die beiden Integrationswege im Überblick

Es gibt zwei unabhängige, offizielle Wege, Codex in Claude Code einzubinden. Sie schließen sich nicht aus und lassen sich parallel nutzen.

	Offizielles Plugin (`codex-plugin-cc`)	**Codex als MCP-Server**
Anbieter	OpenAI, offizielles Repository	Codex-CLI-eingebaute Funktion, in Claude Code über Standard-MCP eingebunden
Bedienung	Feste Slash-Commands (`/codex:review` usw.)	Claude ruft Codex wie ein normales Werkzeug auf, gesteuert per Prompt
Status	Stabil, aktiv gepflegt	Von OpenAI selbst als **experimentell** markiert
Stärke	Fertige Workflows für Review, Rescue, Review-Gate	Flexibel, lässt sich in eigene Subagenten einbauen
Rechteumfang	Read-only für Reviews, ändert selbst keine Dateien	Abhängig von Codex-Sandbox-Einstellung, standardmäßig read-only empfehlenswert
Installation	Über Claude Code Plugin-System	Über `claude mcp add`

Für den Einstieg ist das Plugin der einfachere Weg: vier Befehle, feste Kommandos, kein eigenes Setup nötig. Der MCP-Weg lohnt sich, wenn Codex als frei benutzbares Werkzeug in eigenen Prompts oder in einem eigenen Subagenten erscheinen soll.

Installation

Voraussetzungen

- Claude Code installiert und eingerichtet
- Node.js 18.18 oder neuer (die Codex-CLI selbst verlangt laut npm-Paket mindestens Node 16, das Claude-Code-Plugin setzt 18.18+ voraus)
- Ein ChatGPT-Konto (auch der kostenlose Tarif funktioniert eingeschränkt) oder ein OpenAI-API-Key

Schritt 1: Codex CLI installieren

Es gibt mehrere offizielle Installationswege. Einer reicht.

macOS/Linux, offizielles Installationsskript:

```
curl -fsSL https://chatgpt.com/codex/install.sh | sh
```

macOS mit Homebrew:

```
brew install --cask codex
```

Über npm (plattformunabhängig):

```
npm install -g @openai/codex
```

Windows (PowerShell):

```
powershell -ExecutionPolicy ByPass -c "irm
https://chatgpt.com/codex/install.ps1 | iex"
```

Alternativ lassen sich fertige Binaries (macOS arm64/x64, Linux x64/arm64) direkt von den GitHub Releases des Projekts `openai/codex` herunterladen.

Installation prüfen:

```
codex --version
```

Schritt 2: Bei Codex anmelden

```
codex login
```

Das öffnet einen Browser-Login-Flow für das ChatGPT-Konto (Plus, Pro, Business, Edu, Enterprise; der kostenlose Tarif hat laut Plugin-Dokumentation nur eingeschränkten Zugriff). Für die Authentifizierung mit einem OpenAI-API-Key wird der Schlüssel über stdin an die Codex-CLI übergeben:

```
printenv OPENAI_API_KEY | codex login --with-api-key
```

Dabei muss `OPENAI_API_KEY` zuvor in der Umgebung gesetzt sein; die CLI liest den Schlüssel ein und speichert die Anmeldedaten.

Schritt 3a: Offizielles Plugin in Claude Code installieren

Die folgenden vier Zeilen nacheinander direkt in eine laufende Claude-Code-Session eintippen:

```
/plugin marketplace add openai/codex-plugin-cc
/plugin install codex@openai-codex
/reload-plugins
/codex:setup
```

`/codex:setup` prüft am Ende, ob Codex CLI installiert und eingeloggt ist, und weist auf fehlende Schritte hin.

Schritt 3b: Alternative – Codex als MCP-Server einbinden

Statt oder zusätzlich zum Plugin lässt sich die Codex-CLI direkt als MCP-Server registrieren. Dieser Modus ist von OpenAI ausdrücklich als **experimentell** gekennzeichnet.

Nur für das aktuelle Projekt:

```
claude mcp add codex -- codex mcp-server
```

Für alle Projekte des Nutzers:

```
claude mcp add codex --scope user -- codex mcp-server
```

`--scope` kennt drei Werte: `local` (Standard, nur das aktuelle Projekt, nicht geteilt), `project` (in `.mcp.json` abgelegt, teilbar über Git) und `user` (in der Nutzerkonfiguration, projektübergreifend).

Verbindung prüfen:

```
claude mcp list
claude mcp get codex
```

Typische Kommandos

Slash-Commands des offiziellen Plugins

Command	Zweck
`/codex:setup`	Installation und Login-Status prüfen, Review-Gate an/aus schalten
`/codex:review`	Codex prüft lokale Git-Änderungen im Working Tree oder den Branch-Diff zu einer Basis wie `main`, read-only
`/codex:adversarial-review`	Codex hinterfragt gezielt Designentscheidungen statt sie nur zu bestätigen

Command	Zweck
`/codex:rescue`	Eine Aufgabe komplett an Codex übergeben, etwa zur Fehlersuche
`/codex:transfer`	Claude-Transkript in einen persistenten Codex-Thread importieren und den Befehl zum Fortsetzen ausgeben
`/codex:status`	Status laufender Hintergrund-Jobs anzeigen
`/codex:result`	Ergebnis eines fertigen Hintergrund-Jobs abholen
`/codex:cancel`	Einen laufenden Job abbrechen

Jeder länger laufende Befehl lässt sich mit `--background` im Hintergrund starten, Status und Ergebnis dann über `/codex:status` und `/codex:result` abrufen.

Parameter der Slash-Commands im Detail

Parameter in eckigen Klammern sind optional.

`/codex:setup` – prüft, ob die Codex CLI installiert und angemeldet ist, und verwaltet das automatische Review-Gate.

```
/codex:setup [--enable-review-gate|--disable-review-gate]
```

- `--enable-review-gate` – prüft Claudes Arbeit automatisch durch Codex, bevor Claude eine Aufgabe beendet.
- `--disable-review-gate` – deaktiviert diese automatische Prüfung.

Beispiele:

```
/codex:setup
/codex:setup --enable-review-gate
/codex:setup --disable-review-gate
```

`/codex:review` – führt eine normale, ausschließlich lesende Codeprüfung durch.

```
/codex:review [--wait|--background] [--base <ref>] [--scope auto|working-tree|branch]
```

- `--wait` – wartet auf das Ergebnis.
- `--background` – startet die Prüfung im Hintergrund.
- `--base <ref>` – vergleicht den aktuellen Branch mit einer Basis wie `main`.
- `--scope auto` – lässt das Plugin den passenden Prüfbereich auswählen.
- `--scope working-tree` – prüft lokale Änderungen, einschließlich neuer Dateien.
- `--scope branch` – prüft den aktuellen Branch gegenüber dem Basis-Branch.

Beispiele:

```
/codex:review
/codex:review --wait
/codex:review --background
/codex:review --base main --background
/codex:review --scope working-tree --wait
```

Der Befehl nimmt keinen zusätzlichen Prüfungstext entgegen. Für einen eigenen Schwerpunkt dient `/codex:adversarial-review`.

`/codex:adversarial-review` – hinterfragt Architektur, Annahmen und Designentscheidungen, ohne selbst Dateien zu verändern.

`/codex:adversarial-review [--wait|--background] [--base <ref>] [--scope auto|working-tree|branch] [Schwerpunkt]`

Die technischen Parameter entsprechen `/codex:review`. Zusätzlich lässt sich ein freier Prüfungsschwerpunkt anhängen. Die Bereiche `staged` und `unstaged` werden nicht als eigene `--scope`-Werte unterstützt.

Beispiele:

```
/codex:adversarial-review
/codex:adversarial-review --wait Prüfe auf mögliche Datenverluste
/codex:adversarial-review --base main Hinterfrage das gewählte Caching-Konzept
/codex:adversarial-review --background Suche nach Race Conditions
```

`/codex:rescue` – übergibt eine Untersuchung oder eine konkrete Änderung an den Codex-Subagenten.

`/codex:rescue [--background|--wait] [--resume|--fresh] [--model <Modell|spark>] [--effort <Stufe>] [Aufgabe]`

- `--wait` – führt Codex im Vordergrund aus.
- `--background` – startet Codex im Hintergrund.
- `--resume` – setzt den letzten passenden Codex-Thread fort.
- `--fresh` – beginnt einen neuen Codex-Thread.
- `--model <Modell>` – verwendet für diese Aufgabe ein bestimmtes Codex-Modell; `spark` ist die Kurzform für `gpt-5.3-codex-spark`.
- `--effort <Stufe>` – bestimmt den Denkaufwand, unterstützt werden `none`, `minimal`, `low`, `medium`, `high`, `xhigh`.

Beispiele:

```
/codex:rescue Untersuche, warum die Tests fehlschlagen
/codex:rescue --fresh Behebe den Fehler mit der kleinsten sicheren Änderung
/codex:rescue --resume Setze die zuletzt begonnene Fehlerbehebung fort
/codex:rescue --model gpt-5.4-mini --effort high Untersuche den Fehler
/codex:rescue --model spark --background Behebe den fehlerhaften Test
```

Ohne `--model` und `--effort` gelten die normalen Codex-Standardeinstellungen, konfigurierbar in `~/.codex/config.toml` bzw. `.codex/config.toml`. `--model` und `--effort` werden ausschließlich von `/codex:rescue` angeboten – die Review-Befehle nutzen immer das konfigurierte Standardmodell.

`/codex:transfer` – überträgt die aktuelle Claude-Code-Unterhaltung in einen fortsetzbaren Codex-Thread.

`/codex:transfer [--source <claude-jsonl>]`

- `--source <claude-jsonl>` – verwendet manuell eine bestimmte Claude-Code-Transkriptdatei, statt die aktuelle automatisch zu erkennen.

Beispiele:

```
/codex:transfer
/codex:transfer --source ~/.claude/projects/projekt/session.jsonl
```

Als Ergebnis erhältst du eine Session-ID und den Befehl `codex resume <session-id>` zum Fortsetzen.

`/codex:status` – zeigt laufende und kürzlich abgeschlossene Codex-Jobs an.

`/codex:status [job-id] [--wait] [--timeout-ms <ms>] [--all]`

- `job-id` – zeigt einen bestimmten Job ausführlich an.
- `--wait` – wartet auf eine Statusänderung bzw. den Abschluss.
- `--timeout-ms <ms>` – begrenzt die Wartezeit in Millisekunden.

- `--all` – zeigt alle gespeicherten Jobs des Repositorys an.

Beispiele:

```
/codex:status
/codex:status task-abc123
/codex:status task-abc123 --wait
/codex:status task-abc123 --wait --timeout-ms 30000
/codex:status --all
```

`/codex:result` – ruft das vollständige Ergebnis eines abgeschlossenen Hintergrund-Jobs ab, inklusive Befunde, Zusammenfassung, Dateipfade und Zeilennummern.

```
/codex:result [job-id]
```

Beispiele:

```
/codex:result
/codex:result task-abc123
```

`/codex:cancel` – bricht einen aktiven Hintergrund-Job ab.

```
/codex:cancel [job-id]
```

Beispiele:

```
/codex:cancel
/codex:cancel task-abc123
```

Ohne Job-ID wirken `/codex:result` und `/codex:cancel` auf den zuletzt bzw. aktuell passenden Job in diesem Repository.

Die vollständige Parameter-Referenz pflegt OpenAI im offiziellen Plugin-Repository.

Welcher Befehl passt zu welcher Frage?

Die Befehle unterscheiden sich weniger technisch als in der Haltung, die Codex dabei einnimmt. Als Faustregel:

Typische Frage / Prompt	Befehl	Warum
„Überprüfe meine aktuellen lokalen Git-Änderungen"	`/codex:review`	Prüft den Working Tree als Diff, nicht gezielt eine beliebige einzelne Datei
„Ist die Code-Änderung korrekt?"	`/codex:review`	Korrektheitsfrage zum aktuellen Diff
„Führe ein Code-Review aus"	`/codex:review`	Standardfall nach Abschluss einer Aufgabe
„Ist die letzte Änderung valide?"	`/codex:review`	Gleiche Kategorie wie oben
„Ist das wirklich der richtige Ansatz?"	`/codex:adversarial-review`	Zielt auf Designentscheidung und Annahmen, nicht nur auf Tippfehler
„Wo könnte das im Betrieb brechen?"	`/codex:adversarial-review`	Codex sucht aktiv nach Schwachstellen im Konzept
„Hinterfrage bei den aktuellen Änderungen besonders das Error-Handling"	`/codex:adversarial-review` mit Fokus-Text	Prüft denselben Git-Diff, lässt sich aber auf ein Risiko oder eine Designfrage fokussieren

Typische Frage / Prompt	Befehl	Warum
„Ich komme hier nicht weiter, finde die Ursache"	`/codex:rescue`	Übergibt Diagnose oder Fix komplett an Codex
„Gib mir eine zweite Meinung zur Fehlerursache"	`/codex:rescue`	Codex bearbeitet die Aufgabe eigenständig, nicht nur kommentierend
„Lass mich diese Claude-Session direkt in Codex fortsetzen"	`/codex:transfer`, danach ausgegebenes `codex resume <session-id>`	Importiert zuerst den Kontext; der ausgegebene CLI-Befehl öffnet anschließend den Codex-Thread
„Ist Codex korrekt eingerichtet?"	`/codex:setup`	Prüft Installation/Login, schaltet auch das Review-Gate
„Wie weit ist der Hintergrund-Job?"	`/codex:status`	Statusabfrage laufender Jobs
„Zeig mir das fertige Ergebnis"	`/codex:result`	Ergebnis eines abgeschlossenen Hintergrund-Jobs abholen
„Brich den laufenden Job ab"	`/codex:cancel`	Laufenden Hintergrund-Job abbrechen

Kurz zusammengefasst: `/codex:review` fragt „sind die lokalen Git-Änderungen korrekt". `/codex:adversarial-review` fragt „ist es die richtige Entscheidung". `/codex:rescue` heißt „ich bin blockiert, übernimm die Aufgabe". `/codex:transfer` bereitet den Wechsel aus Claude Code in einen fortsetzbaren Codex-Thread vor. Im Zweifel reicht `/codex:review` für den aktuellen Git-Diff – die anderen drei lohnen sich erst, wenn wirklich eine kritische Zweitmeinung, eine Aufgabenübergabe oder Hintergrundverarbeitung gebraucht wird.

Der Transfer selbst ist zweistufig:

1. In Claude Code `/codex:transfer` ausführen. Das Plugin importiert das aktuelle Claude-Transkript in einen persistenten Codex-Thread und gibt `codex resume <session-id>` aus.
2. Den exakt ausgegebenen `codex resume <session-id>`-Befehl anschließend im Terminal ausführen. Dadurch wird der importierte Thread in Codex geöffnet und dort fortgesetzt.

Das Review-Gate wird über `/codex:setup` geschaltet:

```
/codex:setup --enable-review-gate
/codex:setup --disable-review-gate
```

Ist es aktiviert, prüft Codex jede Antwort von Claude, bevor sie als abgeschlossen gilt, technisch über einen Stop-Hook. Das Plugin weist selbst darauf hin, dass dieser Modus zu langen Rückfrage-Schleifen zwischen beiden Modellen führen und das Nutzungskontingent schnell aufbrauchen kann.

Kommandos rund um die MCP-Einbindung

Command	Zweck
`claude mcp add codex -- codex mcp-server`	Codex als MCP-Server registrieren (Projekt-Scope)
`claude mcp add codex --scope user -- codex mcp-server`	Dasselbe, projektübergreifend
`claude mcp list`	Alle registrierten MCP-Server anzeigen
`claude mcp get codex`	Details und Status der Codex-Verbindung anzeigen

Command	Zweck
`claude mcp remove codex`	Verbindung wieder entfernen

Relevante Codex-CLI-Kommandos

Command	Zweck
`codex login` / `codex logout`	Anmeldung verwalten
`codex --version`	Installierte Version prüfen
`codex mcp-server`	Codex im MCP-Server-Modus starten (wird von Claude Code aufgerufen, nicht manuell)
`codex mcp`	Client-seitige Verwaltung von MCP-Servern innerhalb der Codex-CLI selbst

Praktische Beispiele

Beispiel 1: Normales Review nach einer Aufgabe

Claude baut wie gewohnt ein Feature. Danach im Hintergrund prüfen lassen:

```
/codex:review --background
```

Status und Ergebnis abrufen, sobald die Prüfung fertig ist:

```
/codex:status
/codex:result
```

Beispiel 2: Adversariales Review mit eigenem Fokus

```
/codex:adversarial-review prüfe besonders das Error-Handling und mögliche Race Conditions in diesem Feature
```

Codex bekommt damit gezielt eine kritische Rolle statt einer allgemeinen Prüfung.

Beispiel 3: Aufgabe an Codex abgeben

```
/codex:rescue --background untersuch, warum der Login-Test in CI flaky ist
```

Sinnvoll, wenn Claude bei einer Fehlersuche feststeckt oder eine zweite Herangehensweise gefragt ist.

Beispiel 4: Review-Gate für eine kritische Session

```
/codex:setup --enable-review-gate
```

Danach prüft Codex automatisch jede Antwort von Claude. Für den Alltag ist das meist zu viel, für sicherheitskritische Änderungen oder größere Refactorings kann es sich lohnen. Nach der Session wieder ausschalten:

```
/codex:setup --disable-review-gate
```

Beispiel 5: Codex direkt per MCP befragen

Voraussetzung: Codex ist als MCP-Server eingebunden (Schritt 3b oben). Dann reicht ein normaler Prompt in Claude Code:

```
Frage Codex über das MCP-Tool, ob es in dieser Implementierung
Sicherheitsprobleme erkennt, und bewerte anschließend selbst,
ob die Einschätzung zutrifft.
```

Claude ruft das Codex-Werkzeug auf, übernimmt die Antwort und ordnet sie ein, statt sie unkommentiert zu übernehmen.

Beispiel 6: Eigenen Codex-Subagenten anlegen

Für wiederkehrende Zweitmeinungen lohnt sich ein fester Subagent statt eines ausformulierten Prompts jedes Mal neu. Datei `.claude/agents/codex-consultant.md`:

```
---
name: codex-consultant
description: Fragt OpenAI Codex nach einer unabhängigen Analyse oder zweiten
Meinung.
tools: Read, Grep, Glob, mcp__codex
model: sonnet
---

Du bist die Schnittstelle zu OpenAI Codex.

Übergib die gestellte Aufgabe über das Codex-MCP-Werkzeug an Codex.
Verwende standardmäßig:

- das aktuelle Projektverzeichnis
- read-only, keine Änderungen an Dateien
- keine automatisch genehmigten Befehle

Gib Codex ausreichend Kontext. Fasse seine Antwort anschließend
nicht verfälschend zusammen und kennzeichne deutlich, was von
Codex stammt und was deine eigene Einschätzung ist.
```

Aufruf danach im normalen Chat:

```
Nutze den codex-consultant, um Codex nach einer zweiten Lösung
für dieses Problem zu fragen.
```

`tools: mcp__codex` gibt dem Subagenten Zugriff auf sämtliche Werkzeuge des Codex-MCP-Servers. Für einzelne, konkrete Werkzeuge lässt sich stattdessen `mcp__codex__<werkzeugname>` eintragen; bei einem über ein Plugin gebündelten MCP-Server lautet das Namensschema `mcp__plugin_<plugin-name>_<server-name>__<werkzeugname>`.

Kosten und Kontingente

Beide Seiten rechnen unabhängig voneinander ab, es gibt kein gemeinsames Kontingent:

- Die Codex-Seite läuft über das ChatGPT-Abo (Plus, Pro, Business, Edu, Enterprise, eingeschränkt auch Free) oder über einen OpenAI-API-Key mit klassischer Tokenabrechnung. Sie zählt auf das jeweilige Codex-Nutzungslimit.
- Die Claude-Seite läuft wie gewohnt über das bestehende Claude-Abo bzw. den Anthropic-API-Key.

Zwei Modelle bedeuten also zwei getrennte Kontingente. Das Review-Gate verbraucht davon am meisten, weil jede Claude-Antwort zusätzlich eine Codex-Anfrage auslöst.

Grenzen und Stolperfallen

Ein paar Dinge, die im echten Einsatz auffallen und in keiner Feature-Liste stehen:

- **MCP-Modus ist experimentell.** OpenAI kennzeichnet `codex mcp-server` selbst so. Für produktive, dauerhaft genutzte Workflows ist aktuell das Plugin der robustere Weg, für Experimente und eigene Subagenten eignet sich MCP gut.
- **Review-Gate kann Endlos-Schleifen erzeugen.** Codex und Claude können sich gegenseitig wiederholt korrigieren, ohne zu einem Ergebnis zu kommen. Nur aktivieren, wenn die Session aktiv im Blick behalten wird, und für den Alltag eher gezielt `/codex:review` nach Abschluss einer Aufgabe nutzen.
- **Modellnamen ändern sich häufig.** Feste Versionsbezeichnungen wie in älteren Anleitungen (etwa erfundene Codenamen für Modellversionen) veralten schnell und sollten nicht unkritisch übernommen werden. Das aktuell verwendete Modell lässt sich in `~/.codex/config.toml` über den Schlüssel `model` prüfen und setzen, oder direkt in der Codex-Session umschalten.
- **Reviews sind standardmäßig read-only, Rechte aber konfigurierbar.** Vor dem produktiven Einsatz lohnt ein Blick in die Sandbox- und Berechtigungseinstellungen der Codex-CLI, besonders wenn Codex nicht nur beratend, sondern auch mit Schreibzugriff arbeiten soll.
- **Getrennte Konten, getrennte Datenschutzregeln.** Anfragen an Codex laufen über OpenAI-Infrastruktur, Anfragen an Claude über Anthropic-Infrastruktur. Bei sensiblen Projekten vorher prüfen, ob beide Anbieter für den jeweiligen Anwendungsfall zulässig sind.

Quellen

Stand der Prüfung: 2026-07-23. Alle Angaben wurden gegen die folgenden Primärquellen verifiziert.

Codex-Plugin für Claude Code

- github.com/openai/codex-plugin-cc – offizielles Repository, README

Codex CLI

- github.com/openai/codex – Hauptrepository, README, Installationswege
- developers.openai.com/codex – Codex-CLI-Dokumentation
- developers.openai.com/codex/config-basic – Konfiguration, Modellauswahl
- npm-Paket @openai/codex

Claude Code

- code.claude.com/docs/en/mcp – MCP-Server einbinden, `claude mcp add`, Scopes
- code.claude.com/docs/en/sub-agents – Subagenten, Frontmatter, MCP-Tool-Namensschema