**Claude-Flow@alpha: Ausführung über .wsl-tmp/\*.sh (semikolon-sicherer WSL-Start)**

Diese Anleitung beschreibt den **exakten Ablauf**, wie du aus einer Windows-Anwendung heraus einen **Windows-Terminal/WSL**-Tab öffnest und darin **Claude-Flow@alpha** zuverlässig startest – **ohne** die typischen Quote-/Semikolon-Fehler. Kerngedanke: Statt bash -lc "<Einzeiler; …; …>" an wt.exe zu übergeben, wird eine **.sh-Datei** unter .wsl-tmp/ erzeugt und **nur diese Datei** gestartet. So sieht Windows Terminal **keine** Semikolons mehr in seiner eigenen Befehlszeile (Semikolons sind **Delimiter** für wt-Subcommands). ([Microsoft Learn](https://learn.microsoft.com/en-us/windows/terminal/command-line-arguments?utm_source=chatgpt.com))

**1) Ziel & Vorteile**

* **Stabiler Start:** Keine 0x80070002/„Datei nicht gefunden“-Fehler mehr durch Semikolon-Zerlegung in wt.exe. ([Microsoft Learn](https://learn.microsoft.com/en-us/windows/terminal/command-line-arguments?utm_source=chatgpt.com))
* **Klares Debugging:** Im WSL-Tab sind CMD=…, eval "$CMD", Exit-Code, Banner und optional eine Pause sichtbar.
* **Weniger Tabs:** Ein Start = **ein Tab** (bei Bedarf in ein bestehendes Fenster mit -w 0). ([Microsoft Learn](https://learn.microsoft.com/en-us/windows/terminal/command-line-arguments?utm_source=chatgpt.com))
* **Kompatibel:** Der eigentliche Run bleibt ein **normaler npx-Aufruf** des CLI-Pakets claude-flow@alpha. npx lädt/ausführt Paket-Binärdateien ohne vorherige globale Installation (ggf. mit -y). ([docs.npmjs.com](https://docs.npmjs.com/cli/v8/commands/npx/?utm_source=chatgpt.com))

**2) High-Level-Ablauf (Schritt für Schritt)**

1. **Windows-Seite (Host-App):**  
   Erzeuge unter <Projekt>/.wsl-tmp/ eine Datei flow\_YYYYmmdd\_HHMMSS.sh mit vollständigem Ablauf (ENV, cd, Banner, CMD=…, eval "$CMD", Exit-Handling). Optional auf ausführbar setzen (0755).
2. **Start des Tabs (Windows Terminal → WSL):**  
   **Ohne -lc** und ohne Semikolons in der WT-Befehlszeile starten:
3. wt.exe new-tab --title "<Titel>" `
4. wsl.exe -d <Distro> -- bash <WSL-Pfad-zur-Skriptdatei>

-d/--distribution wählt gezielt die WSL-Distribution (z. B. Ubuntu). ([Microsoft Learn](https://learn.microsoft.com/en-us/windows/wsl/install?utm_source=chatgpt.com))

1. **WSL-Seite (im Skript):**  
   ENV setzen, ins Projekt wechseln, Banner/Status ausgeben.  
   CMD='npx claude-flow@alpha hive-mind spawn "…"' … setzen.  
   eval "$CMD"; EXIT\_CODE=$? ausführen, Erfolg/Fehler melden, optional **Pause** (read -n 1 …) und exec bash -i.

**Warum das funktioniert:** wt.exe sieht **keine Semikolons** mehr (die stehen ausschließlich in der Datei); daher wird die Kommandozeile nicht von wt zerpflückt. Die eigentliche Kette interpretiert **Bash** in WSL – genau wie bei manueller Eingabe. ([Microsoft Learn](https://learn.microsoft.com/en-us/windows/terminal/command-line-arguments?utm_source=chatgpt.com))

**3) Konkretes Skript-Beispiel (.wsl-tmp/flow\_YYYYmmdd\_HHMMSS.sh)**

#!/usr/bin/env bash

set -e

export PATH="$PATH:/usr/local/bin:/usr/bin"

export npm\_config\_yes=true

export FORCE\_COLOR=1

export CI=false

# 1) Projektpfad (WSL-Pfad!)

cd /mnt/d/03\_Git/02\_Python/01\_AI\ Coding\ Station

# 2) Banner + Debug-Ausgaben

echo '╔══════════════════════════════════════════════════════════════╗'

echo '║ 🐝 CLAUDE FLOW HIVE MIND - SWARM MONITOR 🐝 ║'

echo '╚══════════════════════════════════════════════════════════════╝'

echo

printf '%s\n' '📋 Executing command...'

printf '%s\n' '🛠 Command:'

# 3) Der eigentliche Befehl (roh, wie man ihn manuell im WSL-Tab tippt)

CMD='npx claude-flow@alpha hive-mind spawn '\''<DEINE AUFGABE>'\'' \

--config '\''.claude-flow/saved-configs/<dein-config>.json'\'' \

--claude --verbose'

printf "%s\n" "$CMD"

echo '═══════════════════════════════════════════════════════════════'

echo

echo '- Spawning Hive Mind swarm...'

# 4) Ausführen + Exit-Code anzeigen

eval "$CMD"; EXIT\_CODE=$?

echo

echo '═══════════════════════════════════════════════════════════════'

if [ $EXIT\_CODE -eq 0 ]; then

echo "✅ Task completed successfully"

else

echo "❌ Task failed with exit code: $EXIT\_CODE"

fi

echo '═══════════════════════════════════════════════════════════════'

# 5) Optional offen halten und interaktiv bleiben

read -n 1 -s -r -p 'Press any key to close...' || true

echo

exec bash -i

**4) Start aus Windows Terminal (ohne -lc, semikolon-sicher)**

wt.exe new-tab --title "🐝 Hive Mind" `

wsl.exe -d Ubuntu -- bash /mnt/d/03\_Git/02\_Python/01\_AI\ Coding\ Station/.wsl-tmp/flow\_20250817\_164127.sh

* wt.exe öffnet einen neuen Tab mit Titel.
* wsl.exe -d <Distro> -- bash <skript> startet **nur** die Datei – **keine** Semikolons in der WT-Argumentliste.
* Für „ein Tab pro Aktion“ in ein bestehendes Fenster: -w 0 nutzen (siehe WT-Argumente/Beispiele). ([Microsoft Learn](https://learn.microsoft.com/en-us/windows/terminal/command-line-arguments?utm_source=chatgpt.com))

**5) Best Practices**

* **UTF-8 erzwingen**, falls Pfade/Text mit Umlauten/Emoji verwendet werden:
* export LANG=C.UTF-8
* export LC\_ALL=C.UTF-8
* **Keine Emojis** in **Dateinamen** der Configs (z. B. .json) – oder konsequent LANG/LC\_ALL setzen.
* **npx verwenden**, um eine konkrete Alpha zu testen oder zu pinnen, z. B.:
* npx claude-flow@2.0.0-alpha.86 hive-mind spawn "…" --config "…" --claude --verbose

(npx führt Paket-Binärdateien aus und installiert bei Bedarf temporär; Eingriff über -y/--yes.) ([docs.npmjs.com](https://docs.npmjs.com/cli/v8/commands/npx/?utm_source=chatgpt.com))

* **Ein Tab pro Aktion:** wt.exe -w 0 new-tab … öffnet in ein bestehendes Fenster. ([Microsoft Learn](https://learn.microsoft.com/en-us/windows/terminal/command-line-arguments?utm_source=chatgpt.com))
* **WSL-Pfad korrekt bilden:** D:\… → /mnt/d/... auf der Bash-Seite; Distribution gezielt mit -d/--distribution auswählen. ([Microsoft Learn](https://learn.microsoft.com/en-us/windows/wsl/install?utm_source=chatgpt.com))

**6) Troubleshooting**

| **Symptom** | **Ursache** | **Abhilfe** |
| --- | --- | --- |
| 0x80070002 und Fragmente wie " exit $rc" | WT hat frühere -lc-Payload an ; zerschnitten | Payload **nie** an WT übergeben, sondern **immer Skriptdatei** starten. ([Microsoft Learn](https://learn.microsoft.com/en-us/windows/terminal/command-line-arguments?utm_source=chatgpt.com)) |
| Task findet Config nicht / kein Output | Locale/Encoding/Pfad (z. B. Emoji im Dateinamen) | LANG/LC\_ALL=C.UTF-8 setzen; Dateinamen ASCII-safe halten |
| „Zu viele Tabs“ | Mehrfache Starts (Init **und** Spawn) | Init und Spawn trennen; pro Aktion **ein** open\_wsl\_terminal |
| Falsche Distro | WSL-Default anders gesetzt | wsl.exe -d <Distro> gezielt wählen. ([Microsoft Learn](https://learn.microsoft.com/en-us/windows/wsl/install?utm_source=chatgpt.com)) |

**7) Pseudocode (Host-Seite)**

[Windows-App]

├─ 1) Skriptinhalt bauen (ENV, cd, Banner, CMD=..., eval "$CMD", Exit-Code)

├─ 2) Datei schreiben: <Projekt>/.wsl-tmp/flow\_YYYYmmdd\_HHMMSS.sh

├─ 3) WT-Start:

│ wt.exe new-tab --title "<Titel>" wsl.exe -d <Distro> -- bash <WSL-Skriptpfad>

└─ 4) (optional) Log/Tracing der Aktion

Auf der WSL-Seite führt **Bash** das Skript **eins zu eins** aus – so, als würdest du es manuell tippen. Das ist die stabilste und transparenteste Methode für komplexe Aufrufketten.

**8) Hinweise zu MCP & Version-Pinning (optional)**

* Falls --claude genutzt wird, stelle sicher, dass **Claude Code CLI** installiert ist (npm i -g @anthropic-ai/claude-code, claude --version).
* Bei Alpha-Regressionsfehlern (z. B. Schema-Migrationen) hilft temporäres **Version-Pinning** via npx claude-flow@2.0.0-alpha.86 … und sauberer Re-Init des Projekts.

**Referenzen**

* **Windows Terminal – Command line arguments** (Semikolon als Befehls-Delimiter, Beispiele). ([Microsoft Learn](https://learn.microsoft.com/en-us/windows/terminal/command-line-arguments?utm_source=chatgpt.com))
* **Windows Terminal – Überblick** (Profile, Tabs, Optionen). ([Microsoft Learn](https://learn.microsoft.com/en-us/windows/terminal/?utm_source=chatgpt.com))
* **WSL – Install/Basic commands** (--distribution/-d wählen, Distroliste/Default). ([Microsoft Learn](https://learn.microsoft.com/en-us/windows/wsl/install?utm_source=chatgpt.com))
* **npx-Dokumentation** (Ausführen von Paket-Binärdateien, -y/--yes). ([docs.npmjs.com](https://docs.npmjs.com/cli/v8/commands/npx/?utm_source=chatgpt.com))

**# Claude‑Flow@alpha unter WSL: Stabiler Start via Here‑Doc/Temp‑Skript**

**\*\*Version:\*\*** 2025-08-17 12:55

**\*\*Zielgruppe:\*\*** Entwickler:innen (Windows + WSL, Node.js/NPX, Python‑App/GUI)

**\*\*Kurzfassung:\*\*** Komplexe CLI‑Aufrufe (z. B. `npx claude-flow@alpha hive-mind spawn …`) werden **\*\*nicht\*\*** direkt an `wt.exe`/`wsl.exe` gequotet, sondern in WSL als **\*\*temporäres Bash‑Skript\*\*** (`/tmp/flow\_XXXXXX.sh`) per **\*\*Here‑Doc\*\*** erzeugt und ausgeführt. Das vermeidet Quote-/Semikolon‑Fallen über die Parser‑Schichten (PowerShell/CMD → Windows Terminal → wsl.exe → bash).

---

**## 1. Hintergrund & Motivation**

Beim manuellen Tippen in einem interaktiven WSL‑Fenster gibt es nur **\*\*eine\*\*** Auslegungsebene (Bash). Beim Start aus einer Windows‑App treffen **\*\*mehrere Parser\*\*** aufeinander:

CMD/PowerShell, **\*\*Windows Terminal\*\*** (`wt.exe`), `wsl.exe`, `bash -lc`. Zeichen wie **\*\*Semikolon `;`\*\*** und Anführungszeichen werden in jeder Ebene teils **\*\*unterschiedlich\*\*** interpretiert. `wt.exe` nutzt `;` **\*\*selbst\*\*** als Delimiter für mehrere Top‑Level‑Kommandos – das kollidiert mit Semikolons in der Bash‑Payload. Ergebnis: sporadische Abbrüche (z. B. Exit 127), abgeschnittene Payloads oder sofort schließende Fenster.

**\*\*Designentscheidung:\*\***

- Windows‑Seite: **\*\*Argumentlisten\*\*** an `subprocess.Popen` (kein Einzelner String).

- Linux‑Seite: **\*\*Here‑Doc → /tmp/flow\_XXXXXX.sh → `bash $tmp`\*\***.

- Keine WT‑Escapes innerhalb der Bash‑Payload notwendig; Quoting‑Hölle entfällt.

---

**## 2. Voraussetzungen**

- Windows 10/11 mit **\*\*WSL\*\*** (empfohlen: Store‑Version, WSL2) und installierter Distro (z. B. Ubuntu).

- **\*\*Windows Terminal\*\*** (optional, aber empfohlen).

- **\*\*Node.js + npm + npx\*\*** im WSL‑Userland.

- `npx claude-flow@alpha` lauffähig (oder global `claude-flow`).

> **\*\*Leistungs‑Hinweis:\*\*** Für Git‑Repos und große Projekte – nach Möglichkeit **\*\*im Linux‑Dateisystem\*\*** (`~/…`) arbeiten, nicht unter `/mnt/<drive>/…` entwickeln.

---

**## 3. Aufrufprinzip (Kurzform)**

**\*\*Windows → WSL\*\*** (Argumentliste):

```text

wsl.exe -d <Distro> -- bash -lc "<Payload>"

```

**\*\*Payload in Bash\*\***: Erzeuge Temp‑Skript via Here‑Doc und führe es aus:

```bash

tmp=$(mktemp /tmp/flow\_XXXXXX.sh) || exit 1

cat > "$tmp" <<'BASH'

PATH=$PATH:/usr/local/bin:/usr/bin; export PATH

export npm\_config\_yes=true FORCE\_COLOR=1 CI=false

cd /mnt/d/03\_Git/02\_Python/01\_AI\ Coding\ Station

# (Optional) Debug-Spuren

# set -x

echo '=== Claude Flow Hive Mind – Swarm Monitor ==='

printf '%s\n' '📋 Executing command:'

CMD='npx claude-flow@alpha hive-mind spawn '''Dein Task''' --config '''.claude-flow/saved-configs/<cfg.json>''' --claude --verbose'

printf '%s\n' "$CMD"

echo '──────────────────────────────────────────────'

eval "$CMD"; EXIT\_CODE=$?

echo '──────────────────────────────────────────────'

[ $EXIT\_CODE -eq 0 ] && echo '✅ OK' || echo "❌ Exit $EXIT\_CODE"

read -n 1 -s -r -p 'Press any key to close...'; echo

exec bash -i

BASH

bash "$tmp"; rc=$?

rm -f "$tmp"

exit $rc

```

**\*\*Warum `eval "$CMD"`?\*\*** Damit der original eingegebene Befehl (inkl. Quotes) **\*\*exakt\*\*** wie in einer interaktiven Shell läuft. Wenn der Task‑Text *\*absolut\** kontrolliert ist, kannst du statt `eval` auch direkt das Array anlegen und `exec` aufrufen.

---

**## 4. Implementierung in der App (Beispiel‑API)**

**### 4.1 WSL‑Bridge**

- **\*\*Externes Terminal:\*\***

  `open\_wsl\_terminal(command: str, cwd: str, debug: bool=False)`

  → Öffnet vorzugsweise **\*\*Windows Terminal\*\*** mit `new-tab` und übergibt die **\*\*Here‑Doc‑Payload\*\*** als `bash -lc`‑Argument. Fallback: `cmd.exe /c start … wsl.exe …`.

- **\*\*Inline‑Ausführung (ohne neues Fenster):\*\***

  `execute(command: str, cwd: Optional[str]=None)`

  → Führt `bash -lc "<prefix>; cd '<cwd>'; <command>"` aus und gibt `(stdout, stderr, returncode)` zurück.

**### 4.2 Aufruf (GUI‑Beispiel)**

```python

launch\_cmd = (

    "npx claude-flow@alpha hive-mind spawn "

    + shlex.quote(task)

    + " --config "

    + shlex.quote(".claude-flow/saved-configs/<cfg.json>")

    + " --claude --verbose"

)

WSLBridge.open\_wsl\_terminal(launch\_cmd, project\_path, debug=debug\_enabled)

```

**\*\*Wichtig:\*\*** Der **\*\*Kommando‑String bleibt „roh“\*\*** (so wie du ihn manuell ins WSL‑Fenster tippst). Die heikle Quoting‑Arbeit übernimmt **\*\*nicht\*\*** mehr WT, sondern Bash **\*\*in\*\*** WSL – via Datei.

---

**## 5. Best Practices**

- **\*\*Argumentlisten\*\*** an `subprocess.Popen` übergeben (kein einzelner String).

- **\*\*Kein\*\*** WT‑Escaping (Backticks/`\;`) in der Bash‑Payload.

- Task‑Text immer **\*\*quoten\*\*** (z. B. `shlex.quote()`), bevor er in `CMD='…'` landet.

- Für Debug: in der Temp‑Datei **\*\*`set -x`\*\*** aktivieren.

- Lange Laufzeiten: `read -n 1 …` hält das Fenster offen; `exec bash -i` startet interaktiv.

---

**## 6. Fehlersuche**

- **\*\*Exit 127 / „command not found“ sofort nach Start\*\***

  → WT‑Delimiter/Quoting wurden doch „oben“ interpretiert. Sicherstellen: **\*\*Here‑Doc‑Weg\*\*** aktiv, **\*\*kein\*\*** Escaping in der Bash‑Payload, Start **\*\*nur\*\*** via `wsl.exe … bash -lc "<Payload>"`.

- **\*\*Fenster schließt sofort\*\***

  → Prüfe, ob `read -n 1` und `exec bash -i` in der Temp‑Datei stehen; `set -e` vermeiden.

- **\*\*Umlaute/UTF‑8‑Mojibake\*\***

  → `FORCE\_COLOR=1`, UTF‑8‑Locale sicherstellen; bei der Distro‑Liste ggf. `WSL\_UTF8=1` nutzen.

- **\*\*Leistung / hohe I/O‑Last\*\***

  → Repos unter `~/…` führen (Linux‑FS) – nicht im `/mnt/…` Bereich entwickeln.

---

**## 7. Beispiel: Vollbefehl**

```text

npx claude-flow@alpha hive-mind spawn 'Lese die Dokumente im Ordner \01\_documentation …'   --config '.claude-flow/saved-configs/swarm\_20250817\_043434.json'   --topology hierarchical --queen strategic --max-agents 5 --strategy adaptive   --claude --verbose

```

In der App: **\*\*unverändert\*\*** als `CMD='…'` in die Temp‑Datei schreiben; **\*\*keine\*\*** zusätzlichen Backslashes oder Semikolon‑Escapes für Windows Terminal.

---

**## 8. Sicherheitshinweise**

- `eval "$CMD"` führt **\*\*genau den zusammengebauten Text\*\*** aus. Verwende es nur, wenn die Quelle (GUI‑Eingabe, Konfig) **\*\*vertrauenswürdig\*\*** ist. Andernfalls: Kommandoteile als **\*\*Array\*\*** bauen und ohne `eval` starten.

- Temp‑Datei liegt unter `/tmp/flow\_XXXXXX.sh` und wird nach dem Lauf gelöscht.

---

**## 9. Referenzen (Auswahl)**

- Windows Terminal CLI: Semikolon als Delimiter, Escaping aus PowerShell; sowie bekannte Parsing‑Probleme.

- Python `subprocess`: Sequenz von Argumenten empfohlen (Quoting übernimmt Python).

- WSL‑Grundlagen/Kommandos; Aufruf von Distros via `wsl.exe`.

- Bash‑Quoting (Single‑Quotes, Semikolons), Referenz‑Manual.

- Arbeiten über Windows‑/Linux‑Dateisysteme (Performance‑Hinweise).

---

**## 10. Changelog**

- **\*\*Neu:\*\*** Start über Here‑Doc/Temp‑Skript, kein WT‑Escaping mehr nötig.

- **\*\*Entfernt:\*\*** Alte WT‑Payload‑Formatter und `\;`‑Workarounds.