OPENCLAW HACKATHON · 13. MAI 2026 · BYTE5 OFFICE

OpenClaw Hackathon

AgentSkills bauen : wir, heute Abend, gemeinsam

Marcel Wege · CTO @ byte5 · mwege@byte5.de

Heute Abend: vom fertigen Skill zum eigenen Skill

- **Demo** — Non-Annoying News live · ~20 min
- **Phase 0: Brain-Check** — Skill-Anatomy + Smoke-Tests · ~30 min
- **Phase 1: Level-Wahl** — Entry / Intermediate / Expert · ~10 min
- **Phase 2: Build** — Pomodoro-Rhythmus 55 + 10 + 55 · ~120 min
- **Phase 3: Show & Tell** — 3 min pro Team · ~30 min

> **Erfolgs-Versprechen:** alle gehen mit etwas, das **läuft**. Egal welches Level.

Demo : Non-Annoying News

Non-Annoying News: kein Clickbait, keine Filler-Boxen

Ein **public AgentSkill** von @iret77 auf ClawHub — macht aus deinen Bookmarks, Reading-Lists und Feeds eine kompakte Zeitung im Magazin-Stil.

- **Quellen:** X-Bookmarks · Browser Reading List · Read-Later · RSS · Newsletter · gepastete URLs
- **Anti-Pattern:** kein Clickbait, keine Dashboard-Cards, keine vagen Link-Summaries
- **Versprechen:** jeder Artikel ist verständlich, **ohne** die Quelle zu öffnen

> v0.2.1 · ClawHub · `non-annoying-news` → Installation auf der nächsten Folie

Quelle: github.com/iret77/non-annoying-news — README v0.2.1

Installation 1/2 : Agent-geführt

Paste in deinen OpenClaw-Agent — er installiert **und** führt durchs Onboarding:

```
Install the public ClawHub skill `non-annoying-news` by @iret77,
then guide me through setup. Start with onboarding questions
(title, topics, sources, cadence, delivery, design) — do not
invent anything, create config only after I confirm.
```

> **Wichtig :** Keine Tokens oder Cookies in den Chat. Die **Personalization Gate** verhindert Auto-Setup ohne Bestätigung.

Quelle: github.com/iret77/non-annoying-news — README v0.2.1

Installation 2/2 : Manuell via CLI

`openclaw skills install` ist der native Befehl. `clawhub install` läuft alternativ — die Skills-README nutzt ihn.

```
openclaw skills install non-annoying-news    # native
clawhub install non-annoying-news            # alternative
```

Beide installieren in den aktiven Workspace. Details zur Location-Präzedenz kommen auf der nächsten Phase-0-Folie.

Quelle: docs.openclaw.ai/tools/skills

Drei Stufen : Signal → Standard → Issue

1. **Signals sammeln** — Bookmarks und Reading-Lists sind **Intent**, keine Fakten. Der Skill liest die Original-Quelle nach und markiert Zugriffs-Grenzen.
2. **Editorial Standard** — jeder Artikel beantwortet: **was** ist passiert · welcher **Mechanismus** · **warum** relevant · **Grenzen** der Evidenz.
3. **Render** — HTML zuerst, PDF danach. PNG-Previews jeder Seite vor Auslieferung.

> **Personalization Gate :** kein Issue, kein Cron, keine Auslieferung — bis du Titel, Topics, Sources, Cadence und Design bestätigt hast.

Quelle: github.com/iret77/non-annoying-news — README v0.2.1, SKILL.md

Phase 0 : OpenClaw-Brain

Anatomy: ein Folder, ein Manifest, optional mehr

```
my-skill/
├── SKILL.md          # Required: metadata + instructions
├── scripts/          # Optional: executable code
├── references/       # Optional: documentation
└── assets/           # Optional: templates, resources
```

> **Mental Model:** Skill = Manifest + Lazy-Loaded Knowledge + optionale Scripts. Der Agent lädt nur, was er für die Aufgabe braucht.

Quellen: agentskills.io · docs.openclaw.ai/tools/skills

Bevor wir bauen: zwei schnelle Checks

1. **Agent-Hello** — ein bekannter Prompt, du erkennst die Antwort.
2. **Demo-Skill installierbar** — `openclaw skills install non-annoying-news` läuft sauber durch und landet im aktiven Workspace.

```
# Sync-Punkt für alle: kurzer Trockenlauf
openclaw skills install non-annoying-news
```

Skills landen je nach Scope in `<workspace>/skills`, `~/.agents/skills` oder `~/.openclaw/skills` — bei Konflikten gewinnt die höhere Präzedenz.

> **Wenn was klemmt:** **jetzt** melden — wir lösen das hier zusammen, nicht später unter Zeitdruck.

Quelle: docs.openclaw.ai/tools/skills — Location-Präzedenz + CLI-Kommandos

Phase 1 : Level wählen

Pick dein Level: Entry · Intermediate · Expert

Level	Was du baust	Zeit-Realismus in 2 h
Entry	Du remixt einen bestehenden Skill — Config, Design, eigene Quelle	sicheres Ergebnis
Intermediate	Eigener kleiner Skill — **1 Input → 1 Artefakt**	ambitioniert
Expert	Meta-Skill, Composer oder Round-Trip auf ClawHub	sportlich, dafür spannend

Team-Bildung: 1–3 Personen pro Team · gleiches Level. Solo bauen ist auch okay.

Phase 2 : Build

Entry : Remix einen bestehenden Skill

Empfohlen — `non-annoying-news` personalisieren:

- Eigene Quellen (Bookmarks · RSS · Reading-List) durchs Onboarding fahren
- Ein **Design-Preset** ändern — Theme-Farben, Density-Token
- **Eine neue Signal-Quelle** ergänzen — z. B. dein Lieblings-Newsletter als RSS

Open Slot — du hast einen anderen public Skill im Auge, den du remixen willst? Genauso valide.

> **Erfolg :** ein gerendertes, personalisiertes Issue, das deine Sprache spricht.

demos/entry/byte5-news-remix/

```
byte5-news-remix/
├── README.md           # Anwendungs-Anleitung
├── config.json         # Topics, Quellen, Cadence — onboarding.complete=true
└── design-tokens.css   # byte5-Magenta, Cyan, Days One, Density
```

> Config-Overlay für `non-annoying-news`. Klonen · `config.json` auf deine Topics umbiegen · Tokens auf deine Farben.

demos/entry/byte5-news-remix

Entry · Schritt 1 : Onboarding starten

Dein Prompt:

> „Starte das non-annoying-news-Onboarding für mich."

Im Hintergrund:

- Skill lädt `references/onboarding.md`
- Personalization Gate öffnet das Interview
- Agent fragt durch: Titel · Sprache · Reader-Promise · Topics · Quellen · Cadence · Design

> Nichts persistiert, bis du jede Antwort **bestätigst**.

non-annoying-news SKILL.md — Personalization Gate

Entry · Schritt 2 : Antworten + Render

Du gibst die Antworten — z. B.:

> „Topics: AI-Agents, OpenClaw, Web-Standards. Quelle: hnrss.org/frontpage. Cadence wöchentlich. Design: byte5-Magenta, Days One."

Im Hintergrund:

- Antworten landen in einer lokalen Config **außerhalb** des Skill-Ordners
- `onboarding.complete = true` schaltet die Gate frei
- Trigger „Render das erste Issue" → HTML zuerst, dann PDF, PNG-Previews vor Auslieferung

→ Referenz: `demos/entry/byte5-news-remix`

non-annoying-news SKILL.md

Intermediate: 1 Input → 1 Artefakt

Empfohlen — ein kleiner Skill, der genau eine Quelle in genau ein Artefakt verwandelt:

- `RSS-Feed → 5-Bullet-Wochenbrief`
- `Meeting-Transcript → Action-Items-Liste`
- `GitHub-Notifications → Wochenrückblick`

Mindest-Pattern: `SKILL.md` + 1 Reference-Doc + 1 Script + 1 Asset-Template.

Open Slot — eigene Input-Output-Kombo? Pick dir was, woran du genuin interessiert bist.

> **Erfolg:** lokal installierbar, mit **einem** gezeigten Live-Beispiel.

demos/intermediate/rss-zu-wochenbrief/

```
rss-zu-wochenbrief/
├── SKILL.md                          # Manifest + Workflow
├── README.md
├── references/editorial-style.md     # Stil-Regeln pro Bullet
├── scripts/fetch_rss.py              # RSS/Atom-Parser (stdlib-only)
└── assets/template.md                # Markdown-Output-Template
```

> RSS-URL → 5-Bullet-Markdown. **Kopier-Vorlage für deine eigene 1-Input/1-Artefakt-Idee** — Script + Template tauschen, Pattern bleibt.

demos/intermediate/rss-zu-wochenbrief

Intermediate · Schritt 1: Skelett anfordern

Dein Prompt:

> „Bau einen Skill `rss-zu-wochenbrief` in `~/.openclaw/workspace/skills/`. Trigger: Nutzer gibt RSS-URL und will einen 5-Bullet-Wochenbrief im Markdown."

Im Hintergrund:

- Agent legt den Folder `~/.openclaw/workspace/skills/rss-zu-wochenbrief/` an
- Schreibt `SKILL.md` mit Frontmatter (`name`, `description`) + Workflow-Schritten
- Fragt nach Details: welche Felder pro Feed-Item? Bullet-Stil?

docs.openclaw.ai/tools/creating-skills — Skill-Anatomy + Frontmatter

Intermediate · Schritt 2 : Stil + Script

Dein Prompt:

> „Stil-Regeln in `references/editorial-style.md` : Was · Quelle · Datum · 8–25 Wörter · keine Hype-Wörter. Und `scripts/fetch_rss.py` als Python-Stdlib, parsed RSS und Atom."

Im Hintergrund:

- `references/editorial-style.md` wird geschrieben — der Workflow-Schritt aus SKILL.md lädt sie **bei Bedarf**
- `scripts/fetch_rss.py` mit `urllib.request` + `xml.etree.ElementTree` — kein `pip install` nötig

→ Referenz: Demo-Folder im Repo

agentskills.io · docs.openclaw.ai/tools/skills — lazy-loaded references

Intermediate · Schritt 3 : Template + erstes Issue

Dein Prompt:

> „`assets/template.md` mit Platzhaltern für 5 Bullets + Header. Dann `/new`. Wochenbrief aus `https://huggingface.co/blog/feed.xml`."

Im Hintergrund:

1. Template-Datei wird geschrieben — vier Schichten jetzt komplett
2. `/new` lädt frische Session — Skill erscheint im System-Prompt
3. Agent ruft `fetch_rss.py` auf · wählt 5 Items · schreibt Bullets nach Stil-Regeln · füllt Template

> Skill lebt jetzt im Workspace · jede neue Session sieht ihn.

docs.openclaw.ai/tools/skills — Skills-Loading + Token-Impact

Expert : Skill ist nur der Anfang

Empfohlen — wähle einen Pfad:

- **Meta-Skill** — analysiert andere Skills auf OpenClaw-Konventionen (`SKILL.md` -Schema, Personalization-Gate, Editorial-Standard)
- **Composer-Skill** — orchestriert 2+ Skills mit eigenem QA-Gate dazwischen
- **Round-Trip** — Skill bauen, validieren, auf ClawHub publishen

Open Slot — eigenes Expert-Vorhaben? Geh es an, wir reviewen mit.

> **Erfolg :** ein Pattern, das jemand anderes übernehmen kann.

demos/expert/daily-briefing/

```
daily-briefing/
├── SKILL.md                          # Composer-Workflow (3 Steps)
├── README.md
├── references/qa-gate.md             # QA-Regeln + Output-Schema
├── scripts/qa_check.py               # Validator (pass→weiter, fail→Stopp)
└── assets/composition-diagram.md     # Mermaid: Pipeline
```

> `rss-zu-wochenbrief` + QA-Gate + `non-annoying-news`. **Pipeline-Pattern — andere Skills, anderes Gate, anderes Ergebnis.**

demos/expert/daily-briefing

Expert · Schritt 1 : Composer-Skelett

Voraussetzung: `rss-zu-wochenbrief` schon im Workspace (Intermediate-Demo).

Dein Prompt:

> „Bau einen Composer `daily-briefing` in meinem Workspace. Pipeline: RSS-URL → rss-zu-wochenbrief → QA-Gate → non-annoying-news. Trigger: 'Daily Briefing aus <URL>'."

Im Hintergrund:

- Agent legt Skill-Folder + `SKILL.md` an
- Workflow-Body: 3 Steps mit **„rufe Skill X auf"** — keine Re-Implementierung
- Agent fragt nach Details: welche Regeln soll das QA-Gate prüfen?

docs.openclaw.ai/tools/creating-skills · docs.openclaw.ai/tools/skills

Expert · Schritt 2 : QA-Gate definieren

Dein Prompt:

> „Pack die QA-Regeln in `references/qa-gate.md` : 5 Bullets pflicht · jede mit Quelle + Datum · 8–25 Wörter · keine Hype-Wörter. Validator als `scripts/qa_check.py` (stdlib · Exit 0 oder 1)."

Im Hintergrund:

- Reference-Markdown schreibt die Regeln **menschlich lesbar** — Composer-Agent lädt sie als Kontext
- Python-Script implementiert die Regeln **maschinell** — Exit-Code steuert die Pipeline

→ Referenz: `scripts/qa_check.py`

docs.openclaw.ai/tools/skills — references als lazy-loaded Kontext

Expert · Schritt 3 : Pipeline live

Dein Prompt:

> „ `/new` . Daily Briefing aus `https://huggingface.co/blog/feed.xml` ."

Im Hintergrund:

1. `rss-zu-wochenbrief` läuft → 5-Bullet-Markdown
2. `qa_check.py` läuft → `passed: true` **oder** JSON-Diagnose
3. Bei pass: `non-annoying-news` rendert finales Issue · bei fail: Stopp + Report

→ Diagramm: `composition-diagram.md`

docs.openclaw.ai/tools/skills

Same Pattern : in Claude Desktop

Composer ohne OpenClaw — andere Primitives, gleiche Idee:

OpenClaw	Claude Desktop
`SKILL.md` + Workflow	**Projekt** mit System-Prompt
`references/qa-gate.md`	Attached Reference-File im Projekt
`scripts/qa_check.py`	Claude evaluiert die Regeln **inline**
Skill ruft Skill auf	Inline-Pipeline in einer Conversation
`/new` zum Reloaden	Neuer Chat im Projekt

> **Was bleibt:** Composer-Pattern · Reference-driven Workflow · Editorial Standard.

Claude Projects · System-Prompts + Reference-Files: claude.ai

Claude Desktop · Schritt 1: System-Prompt per Meta-Prompt

Dein Prompt (in einem leeren Claude-Chat):

> „Schreib mir einen Claude-Projekt-System-Prompt für einen Daily-Briefing-Workflow. Pipeline: RSS-URL → 5-Bullet-Markdown → QA gegen Editorial-Standards → finales Issue. Strukturiere ihn als nummerierte Schritte mit Fehler-Handling."

Im Hintergrund:

- Claude schreibt einen vollständigen System-Prompt als Artifact
- Inkludiert: Zweck · Eingabe-Format · Workflow 1–3 · Output-Format · Fehler-Handling
- Du kopierst den Artifact-Inhalt ins Projekt unter **Settings → Custom Instructions**

Claude Projects · System-Prompts: claude.ai

Claude Desktop · Schritt 2 : References per Meta-Prompt

Dein Prompt:

> „Generiere zwei Reference-Files für mein Daily-Briefing-Projekt:
>
> 1. `editorial-style.md` — Was · Quelle · Datum · 8–25 Wörter · keine Hype-Wörter
> 2. `qa-rules.md` — 5 Bullets pflicht · Hype-Wort-Blacklist · Quellen-Pflicht"

Im Hintergrund:

- Claude schreibt **beide Files** als separate Artifacts mit präziser Markdown-Struktur
- Du speicherst sie lokal und lädst sie im Projekt unter **Knowledge** hoch
- Damit sind beide Reference-Files für jede neue Project-Conversation verfügbar

Claude Projects · Project Knowledge: claude.ai

Claude Desktop · Schritt 3: Triggern + Iterieren

Dein Prompt (neuer Chat im Projekt):

> „Daily Briefing aus `https://huggingface.co/blog/feed.xml`"

Im Hintergrund:

- Claude liest System-Prompt + beide Reference-Files automatisch
- Web-Fetch des Feeds → 5-Bullet-Draft → QA gegen Rules → Issue als Artifact
- Bei QA-Fail: Claude meldet die Verstöße und schlägt Rewrites vor

Iterations-Prompt (wenn du nachschärfen willst):

> „Bullet 3 hat ein Hype-Wort, Bullet 5 ist zu lang. Rewrite beide gegen die `qa-rules.md`."

Claude Projects · Artifacts + Web-Fetch: claude.ai

Pomodoro : 55 + 10 + 55

1. **Block 1** — 55 min bauen, ausprobieren, scheitern dürfen
2. **Cross-Check** — 10 min gemeinsame Sync-Runde, was funktioniert nicht
3. **Block 2** — 55 min stabilisieren, demoreif machen

Deine Hilfe-Quellen:

- **Dein eigener Agent** — pair-programmt mit dir
- **ClawHub** — andere public Skills als Vorlage / Pattern-Quelle
- **Wir als Mentoren** — Marcel (+ ggf. weitere) bei echten Blöckern

> **Leitsatz :** funktioniert > schön · **kein Production-Skill in 2 h.**

Phase 3 : Show & Tell

Format : 3 min · Demo > Slides · kein Kritik-Format

Was zeigen:

- Was hast du gebaut? In **einem** Satz.
- Live-Demo — der Skill in Aktion. Wenn er kaputt geht: auch okay, ist Hackathon.
- Was war der Aha-Moment?

Was **nicht** zeigen:

- Code-Walkthrough Zeile für Zeile
- Folien-Deck — du hast gerade einen **Skill** gebaut, kein Tech-Talk

> **Feedback-Regel :** nur „cool, weil..." — Kritik bekommst du danach, einzeln, wenn du sie willst.

Vielen Dank.

Vielen Dank.

MATERIAL ZUM MITNEHMEN

- **Slides:** byte5ai.github.io/meetups/2026-05-13-openclaw-hackathon-1 · PDF
- **Repo:** github.com/byte5ai/meetups/tree/main/2026-05-13-openclaw-hackathon-1
- **Demo-Skills:** drei Levels im Repo · Entry · Intermediate · Expert
- **Original-Demo:** github.com/iret77/non-annoying-news
- **byte5:** byte5.de

DU SUCHST UNTERSTÜTZUNG BEI DEINEM DIGITALEN PROJEKT?

Dein digitales Projekt: unsere Expert:innen beraten dich transparent.