X Research Agent

OpenClaw Skill – Einrichtungshandbuch

Automatisierte X/Twitter-Recherche
nach personalisierten Interessen

Version:	1.0
Datum:	11. Februar 2026
Autor:	VA Engineering GmbH
Projekt:	Granaria Foundation
Plattform:	OpenClaw Agent Framework

Dieses Handbuch beschreibt die vollständige Einrichtung eines X/Twitter-Suchagenten
als OpenClaw-Skill mit interessenbasierter Filterung.

0 Inhaltsverzeichnis

1 Überblick

Der **X Research Agent** ist ein OpenClaw-Skill, der die X/Twitter Search API nutzt, um automatisiert relevante Posts zu recherchieren. Der Agent zerlegt Forschungsfragen in gezielte Suchanfragen, filtert nach konfigurierten Interessen und erstellt zusammengefasste Briefings.

1.1 Funktionsumfang

Der Agent unterstützt folgende Aufgaben:

- Zerlegung von Forschungsfragen in 3–5 gezielte API-Queries
- Suche über die X Search API (letzte 7 Tage)
- Engagement-basierte Sortierung (Likes, Impressions, Retweets)
- Thread-Verfolgung via `conversation_id`
- Deep-Dive in verlinkte Inhalte (GitHub, Blogs, Docs)
- Personalisierte Filterung nach gespeicherten Interessen
- Export als Markdown-Briefing

1.2 Voraussetzungen

Komponente	Anforderung
OpenClaw	Installiert und lauffähig
X Developer Account	Free Tier ausreichend
Bearer Token	Read-only (OAuth 2.0 App-Only)
Python 3	Für JSON-Parsing
curl	Für API-Aufrufe

2 X API Zugang einrichten

2.1 Developer Account erstellen

1. Öffne https://developer.x.com/en/portal/petition/essential/basic-info
2. Melde dich mit deinem X-Account an
3. Beschreibe den Anwendungsfall (z. B. “Research and analysis of public discourse”)
4. Akzeptiere die Developer Agreement

2.2 App erstellen und Bearer Token generieren

1. Im Developer Portal: **Projects & Apps** → **+ Add App**
2. App-Name vergeben (z. B. `openclaw-x-research`)
3. Unter **Keys and Tokens**: **Bearer Token** generieren
4. Token kopieren und sicher speichern

2.3 Token in der Shell konfigurieren

Trage den Bearer Token in deine Shell-Konfiguration ein:

```
# ~/.zshrc (oder ~/.bashrc)
export X_BEARER_TOKEN="AAAAAAAAAAAAAAAAAAAAAxxxxxxxxxx..."
```

Danach Shell neu laden:

```
source ~/.zshrc
```

Verbindung testen:

```
curl -s -H "Authorization: Bearer $X_BEARER_TOKEN" \
  "https://api.x.com/2/tweets/search/recent?query=test&max_results=10"
      \
  | python3 -m json.tool | head -5
```

Ergebnis

Bei erfolgreicher Verbindung erscheint eine JSON-Antwort mit `"data: [{...}]`.

! Wichtig

Der Bearer Token erlaubt nur **Lese-Zugriff**. Zum Posten werden zusätzlich OAuth 1.0a User-Credentials benötigt (Consumer Key, Consumer Secret, Access Token, Access Token Secret) – siehe Abschnitt 12.

3 Verzeichnisstruktur

Der Skill wird unter `~/.openclaw/skills/` installiert:

```
~/.openclaw/skills/x-research/
  SKILL.md                        # Skill-Definition (Trigger,
     Beschreibung)
  references/
    x-api.md                      # X API Referenzdokumentation
    interests.yaml                # Personalisierte Interessen-
       Konfiguration
  scripts/
    xparse.py                     # JSON-Parser fuer API-Responses
    xsearch.sh                    # Batch-Suchskript
~/clawd/drafts/                   # Output-Verzeichnis fuer Briefings
```

3.1 Verzeichnisse anlegen

```
mkdir -p ~/.openclaw/skills/x-research/references
mkdir -p ~/.openclaw/skills/x-research/scripts
mkdir -p ~/clawd/drafts
```

4 Datei 1: SKILL.md

Die zentrale Skill-Datei definiert Trigger-Keywords, Beschreibung und den Research-Loop.

~/.openclaw/skills/x-research/SKILL.md

```
---
name: x-research
description: >
  General-purpose X/Twitter research agent. Searches X for
  real-time perspectives, dev discussions, product feedback,
  cultural takes, breaking news, and expert opinions.
  Use when: (1) user says "x research", "search x for",
  "search twitter for", "what are people saying about",
```

```
  "what's twitter saying", "check x for", "x search",
  "/x-research", (2) user is working on something where
  recent X discourse would provide useful context,
  (3) user wants to find what devs/experts/community thinks.
  NOT for: posting tweets, account management, or historical
  archive searches beyond 7 days.
---
```

Unterhalb des YAML-Headers folgt die Markdown-Dokumentation mit dem Research-Loop (Abschnitte 1–6): Decompose, Search, Follow Threads, Deep-Dive, Synthesize, Save.

Die **Trigger-Keywords** im Überblick:

Trigger-Phrase	Beispiel
`x research`	"X research zu Polymarket"
`search x for`	"Search X for MCP server news"
`search twitter for`	"Search Twitter for trading bots"
`what are people saying about`	"What are people saying about OpenClaw?"
`what's twitter saying`	"What's Twitter saying about multi-agent?"
`check x for`	"Check X for WhatsApp automation"
`x search`	"X search: self-hosting AI"
`/x-research`	"/x-research Hyperliquid"

5 Datei 2: x-api.md (API-Referenz)

~/.openclaw/skills/x-research/references/x-api.md

```
# X API Reference

## Authentication
Bearer token from env var X_BEARER_TOKEN.
  -H "Authorization: Bearer $X_BEARER_TOKEN"

## Search Endpoint
GET https://api.x.com/2/tweets/search/recent
Covers last 7 days. Max 100 results per request.

### Standard Query Params
tweet.fields=created_at,public_metrics,author_id,
             conversation_id,entities
expansions=author_id
user.fields=username,name,public_metrics
max_results=100
sort_order=relevancy  (oder recency)
```

5.1 Suchoperatoren

Operator	Beispiel	Beschreibung
Keyword	`OpenClaw agent`	Implizites AND
`OR`	`bun OR deno`	Muss Großbuchstaben sein
`-`	`-is:retweet`	Negation / Ausschluss
`()`	`(fast OR perf)`	Gruppierung
`from:`	`from:elonmusk`	Posts eines bestimmten Users
`to:`	`to:elonmusk`	Replies an einen User

Operator	Beispiel	Beschreibung
`#`	`#buildinpublic`	Hashtag-Suche
`$`	`$AAPL`	Cashtag (Finanz-Ticker)
`lang:`	`lang:en`	Sprachfilter (BCP-47)
`is:retweet`	`-is:retweet`	Retweets filtern
`is:reply`	`-is:reply`	Replies filtern
`has:media`	`has:media`	Enthält Medien
`has:links`	`has:links`	Enthält Links
`url:`	`url:github.com`	Links zu Domain
`conversation_id:`	`conversation_id:123`	Thread nach Root-Tweet

! Wichtig

Die Operatoren `min_likes`, `min_retweets` und `min_replies` sind im Free Tier **nicht verfügbar**. Engagement muss nachträglich über `public_metrics` gefiltert werden.

5.2 Response-Struktur

```
{
  "data": [{
    "id": "tweet_id",
    "text": "...",
    "author_id": "user_id",
    "created_at": "2026-...",
    "public_metrics": {
      "retweet_count": 0, "reply_count": 0,
      "like_count": 0, "impression_count": 0
    },
    "entities": {
      "urls": [{"expanded_url": "https://..."}],
      "mentions": [{"username": "..."}],
      "hashtags": [{"tag": "..."}]
    }
  }],
  "includes": {
    "users": [{"id": "...", "username": "...",
               "name": "...", "public_metrics": {...}}]
  },
  "meta": {"next_token": "...", "result_count": 100}
}
```

5.3 Rate Limits

Ebene	Limit
App-Level	450 Requests / 15 Minuten
User-Level	300 Requests / 15 Minuten
Max Query-Länge	512 Zeichen
Max Operatoren	ca. 10 pro Query

6 Datei 3: interests.yaml (Interessen-Konfiguration)

Diese Datei definiert die personalisierten Interessen für die Suche. Der Agent generiert daraus automatisch die passenden Queries.

~/.openclaw/skills/x-research/references/interests.yaml

```yaml
# X Research Agent -- Interessen-Konfiguration
# Jedes Interesse generiert eine eigene API-Query.
# Passe Keywords und Sprache nach Bedarf an.

interests:
  - name: OpenClaw Ecosystem
    query: "OpenClaw -is:retweet"
    lang: null  # alle Sprachen
    priority: high

  - name: AI Trading Bots
    query: '"AI trading" (bot OR automation OR agent) -is:retweet'
    lang: en
    priority: high

  - name: Polymarket
    query: "Polymarket (prediction OR AI OR agent) -is:retweet"
    lang: en
    priority: high

  - name: MCP Server
    query: '"MCP server" (build OR integration OR tool) -is:retweet'
    lang: null
    priority: medium

  - name: WhatsApp AI Integration
    query: "WhatsApp (AI OR bot OR automation OR agent) -is:retweet"
    lang: en
    priority: medium

  - name: Multi-Agent AI
    query: '"multi-agent" (AI OR LLM OR autonomous) -is:retweet'
    lang: en
    priority: medium

  - name: Self-Hosting AI
    query: "(self-host OR self-hosted) (AI OR LLM OR agent) -is:retweet"
    lang: en
    priority: low

  - name: Crypto DeFi Agents
    query: "(DeFi OR crypto) (AI agent OR autonomous) -is:retweet"
    lang: en
    priority: low

# Globale Einstellungen
settings:
  max_results_per_query: 20
  sort_order: relevancy
  min_impressions: 100       # Post-hoc Filter
  min_likes: 5               # Post-hoc Filter
  output_dir: ~/clawd/drafts
  filename_pattern: "x-digest-{date}.md"
```

Um neue Interessen hinzuzufügen, einfach einen neuen Block unter `interests:` ergänzen. Das Feld `priority` steuert die Reihenfolge im Digest.

7 Datei 4: xparse.py (JSON-Parser)

Das Parser-Skript wandelt rohe API-Responses in lesbaren Output um.

~/.openclaw/skills/x-research/scripts/xparse.py

```
#!/usr/bin/env python3
"""X API Response Parser fuer OpenClaw x-research Skill.
Liest JSON von stdin, gibt formatierte Tweet-Liste aus.

Usage:
    curl -s ... | python3 xparse.py
    curl -s ... | python3 xparse.py --min-likes 10
    curl -s ... | python3 xparse.py --min-impressions 500
"""

import json
import sys
import argparse

def parse_args():
    p = argparse.ArgumentParser(description='X API Response Parser')
    p.add_argument('--min-likes', type=int, default=0,
                   help='Minimum Likes fuer Anzeige')
    p.add_argument('--min-impressions', type=int, default=0,
                   help='Minimum Impressions fuer Anzeige')
    p.add_argument('--sort-by', choices=['likes', 'impressions', 'recency'],
                   default='likes', help='Sortierung')
    p.add_argument('--limit', type=int, default=0,
                   help='Max Anzahl Ergebnisse (0=alle)')
    p.add_argument('--markdown', action='store_true',
                   help='Markdown-Ausgabe')
    return p.parse_args()

def main():
    args = parse_args()
    data = json.load(sys.stdin)

    # Error handling
    if 'errors' in data and 'data' not in data:
        for e in data['errors']:
            print(f"API Error: {e.get('message', 'Unknown')}", file=sys.stderr)
        sys.exit(1)

    if 'status' in data and data.get('status') == 429:
        print("Rate Limit erreicht. 15 Min warten.", file=sys.stderr)
        sys.exit(429)

    # User-Lookup
    users = {}
    for u in data.get('includes', {}).get('users', []):
        users[u['id']] = u

    tweets = data.get('data', [])
    count = data.get('meta', {}).get('result_count', 0)

    # Engagement-Filter
    filtered = []
    for t in tweets:
        m = t['public_metrics']
        if m['like_count'] >= args.min_likes and \
           m['impression_count'] >= args.min_impressions:
            filtered.append(t)

    # Sortierung
    if args.sort_by == 'likes':
        filtered.sort(key=lambda t: t['public_metrics']['like_count'],
                      reverse=True)
    elif args.sort_by == 'impressions':
        filtered.sort(key=lambda t: t['public_metrics']['impression_count'],
                      reverse=True)

    # Limit
    if args.limit > 0:
        filtered = filtered[:args.limit]
```

```
    # Ausgabe
    for t in filtered:
        u = users.get(t['author_id'], {})
        m = t['public_metrics']
        uname = u.get('username', '?')
        urls = [
            url.get('expanded_url', '')
            for url in t.get('entities', {}).get('urls', [])
            if url.get('expanded_url')
        ]

        if args.markdown:
            print(f"- **@{uname}** ({m['like_count']}L, "
                  f"{m['impression_count']}I): "
                  f"{t['text'][:200]}...")
            link = f"https://x.com/{uname}/status/{t['id']}"
            print(f"  [Tweet]({link})")
            for lnk in urls[:2]:
                print(f"  - [{lnk[:60]}...]({lnk})")
            print()
        else:
            print(f"@{uname} | {m['like_count']}L "
                  f"{m['retweet_count']}RT {m['impression_count']}I")
            print(f"  {t['text'][:300]}")
            print(f"  https://x.com/{uname}/status/{t['id']}")
            for lnk in urls[:2]:
                print(f"  -> {lnk}")
            print()

    # Summary
    shown = len(filtered)
    print(f"--- {shown}/{count} Tweets angezeigt ---")
    if count > shown:
        skipped = count - shown
        print(f"    ({skipped} durch Filter ausgeblendet)")

if __name__ == '__main__':
    main()
```

Skript ausführbar machen:

```
chmod +x ~/.openclaw/skills/x-research/scripts/xparse.py
```

8 Datei 5: xsearch.sh (Batch-Suchskript)

Dieses Shell-Skript liest die `interests.yaml` und führt alle Searches automatisch durch.

~/.openclaw/skills/x-research/scripts/xsearch.sh

```
#!/bin/zsh
# X Research Agent -- Batch Search
# Sucht alle Interessen aus interests.yaml
# Usage: ./xsearch.sh [--min-likes N] [--markdown]

set -e

# Konfiguration
SKILL_DIR="$HOME/.openclaw/skills/x-research"
PARSER="$SKILL_DIR/scripts/xparse.py"
INTERESTS="$SKILL_DIR/references/interests.yaml"
OUTPUT_DIR="$HOME/clawd/drafts"
DATE=$(date +%Y-%m-%d)
OUTPUT="$OUTPUT_DIR/x-digest-$DATE.md"
```

```
# API Parameter
FIELDS="tweet.fields=created_at,public_metrics,author_id"
FIELDS="$FIELDS,conversation_id,entities"
FIELDS="$FIELDS&expansions=author_id"
FIELDS="$FIELDS&user.fields=username,name,public_metrics"
FIELDS="$FIELDS&sort_order=relevancy"
FIELDS="$FIELDS&max_results=20"

# Argumente weiterleiten an Parser
PARSER_ARGS="$@"

# Header
echo "# X Research Digest -- $DATE" > "$OUTPUT"
echo "" >> "$OUTPUT"
echo "Automatisch generiert vom x-research Skill." >> "$OUTPUT"
echo "" >> "$OUTPUT"

# Queries aus YAML extrahieren (einfacher Parser)
# Liest name: und query: Felder
CURRENT_NAME=""
while IFS= read -r line; do
  case "$line" in
    *"- name:"*)
      CURRENT_NAME=$(echo "$line" | sed 's/.*- name: //')
      ;;
    *"query:"*)
      QUERY=$(echo "$line" | sed "s/.*query: //;s/^['\"]//;s/['\"]$//")
      if [[ -n "$QUERY" && -n "$CURRENT_NAME" ]]; then
        echo ">> Suche: $CURRENT_NAME"
        echo "   Query: $QUERY"

        # URL-Encode
        ENCODED=$(python3 -c "import urllib.parse; \
          print(urllib.parse.quote('$QUERY'))")

        # API Call
        echo "" >> "$OUTPUT"
        echo "## $CURRENT_NAME" >> "$OUTPUT"
        echo "" >> "$OUTPUT"

        curl -s -H "Authorization: Bearer $X_BEARER_TOKEN" \
          "https://api.x.com/2/tweets/search/recent?\
query=$ENCODED&$FIELDS" \
          | python3 "$PARSER" --markdown $PARSER_ARGS \
          >> "$OUTPUT" 2>&1

        echo "" >> "$OUTPUT"

        # Rate Limit: kurze Pause zwischen Queries
        sleep 1
      fi
      ;;
  esac
done < "$INTERESTS"

# Footer
echo "---" >> "$OUTPUT"
echo "## Metadaten" >> "$OUTPUT"
echo "- **Datum:** $DATE" >> "$OUTPUT"
echo "- **Config:** $INTERESTS" >> "$OUTPUT"
echo "- **Output:** $OUTPUT" >> "$OUTPUT"

echo ""
echo "Digest gespeichert: $OUTPUT"
```

Skript ausführbar machen:

```
chmod +x ~/.openclaw/skills/x-research/scripts/xsearch.sh
```

9 Installation – Schritt für Schritt

Alle Befehle in einem Terminal ausführen:

9.1 Schritt 1: Verzeichnisse erstellen

```
mkdir -p ~/.openclaw/skills/x-research/references
mkdir -p ~/.openclaw/skills/x-research/scripts
mkdir -p ~/clawd/drafts
```

9.2 Schritt 2: Bearer Token setzen

```
echo 'export X_BEARER_TOKEN="DEIN_TOKEN_HIER"' >> ~/.zshrc
source ~/.zshrc
```

9.3 Schritt 3: Dateien erstellen

Erstelle die fünf Dateien aus den Abschnitten 4–8. Alternativ aus einem Git-Repository klonen, falls verfügbar.

9.4 Schritt 4: Berechtigungen setzen

```
chmod +x ~/.openclaw/skills/x-research/scripts/xparse.py
chmod +x ~/.openclaw/skills/x-research/scripts/xsearch.sh
```

9.5 Schritt 5: Verbindung testen

```
curl -s -H "Authorization: Bearer $X_BEARER_TOKEN" \
  "https://api.x.com/2/tweets/search/recent?query=test&max_results=10"
      \
  | python3 ~/.openclaw/skills/x-research/scripts/xparse.py
```

9.6 Schritt 6: Batch-Suche ausführen

```
~/.openclaw/skills/x-research/scripts/xsearch.sh --min-likes 5
```

Ergebnis

Das Ergebnis wird gespeichert unter: `~/clawd/drafts/x-digest-YYYY-MM-DD.md`

10 Verwendung

10.1 Im OpenClaw Chat

Einfach einen der Trigger-Phrasen verwenden:

```
> x research: was sagen Devs über Polymarket AI agents?
> search x for MCP server integrations
> /x-research trading automation 2026
> check x for WhatsApp AI bots
```

OpenClaw erkennt den Trigger, liest die `SKILL.md`, und führt den Research-Loop automatisch aus.

10.2 Direkt im Terminal

Einzelne Query manuell ausführen:

```
P=~/.openclaw/skills/x-research/scripts/xparse.py

curl -s -H "Authorization: Bearer $X_BEARER_TOKEN" \
  "https://api.x.com/2/tweets/search/recent?\
query=Polymarket%20prediction%20-is%3Aretweet&\
max_results=50&\
tweet.fields=created_at,public_metrics,author_id,entities&\
expansions=author_id&\
user.fields=username,name,public_metrics&\
sort_order=relevancy" \
  | python3 $P --min-likes 10 --sort-by impressions
```

10.3 Vollständigen Digest generieren

```
# Alle Interessen durchsuchen, min. 5 Likes
~/.openclaw/skills/x-research/scripts/xsearch.sh --min-likes 5

# Nur Top-Posts mit Markdown-Output
~/.openclaw/skills/x-research/scripts/xsearch.sh \
  --min-likes 20 --limit 5 --markdown
```

10.4 Thread verfolgen

Wenn ein Tweet interessant ist, den Thread laden:

```
# conversation_id aus dem urspruenglichen Tweet
curl -s -H "Authorization: Bearer $X_BEARER_TOKEN" \
  "https://api.x.com/2/tweets/search/recent?\
query=conversation_id:TWEET_ID&\
max_results=100&\
tweet.fields=created_at,public_metrics,author_id&\
expansions=author_id&\
user.fields=username,name&\
sort_order=recency" \
  | python3 $P
```

10.5 Eigenes Profil analysieren

```
curl -s -H "Authorization: Bearer $X_BEARER_TOKEN" \
  "https://api.x.com/2/tweets/search/recent?\
query=from%3ADEIN_HANDLE&\
```

```
max_results=100&\
tweet.fields=created_at,public_metrics,author_id,entities&\
expansions=author_id&\
user.fields=username,name,public_metrics,description&\
sort_order=recency" \
  | python3 $P --sort-by impressions
```

11 Interessen anpassen

Die Datei `interests.yaml` ist die zentrale Konfiguration. Hier einige Beispiele für neue Einträge:

```
  - name: Hyperliquid Trading
    query: "Hyperliquid (bot OR trading OR perps) -is:retweet"
    lang: en
    priority: high

  - name: Alpaca API
    query: "Alpaca (trading OR API OR stocks) -is:retweet"
    lang: en
    priority: medium

  - name: LaTeX Tipps
    query: "LaTeX (tips OR template OR Texifier) -is:retweet"
    lang: null
    priority: low
```

Die `settings`-Sektion steuert globale Filter:

```
settings:
  max_results_per_query: 20     # API-Limit pro Query
  sort_order: relevancy         # oder: recency
  min_impressions: 100          # Post-hoc Mindest-Impressions
  min_likes: 5                  # Post-hoc Mindest-Likes
  output_dir: ~/clawd/drafts
  filename_pattern: "x-digest-{date}.md"
```

12 Posting auf X (optional)

! Wichtig

Posting erfordert **OAuth 1.0a** mit Write-Berechtigung – der Bearer Token reicht dafür **nicht** aus.

12.1 Zusätzliche Credentials

Im X Developer Portal unter **User authentication settings**:

1. App Permissions auf **Read and Write** setzen
2. Unter **Keys and Tokens** generieren:
   - Consumer Key (API Key)
   - Consumer Secret (API Key Secret)
   - Access Token
   - Access Token Secret

Diese in die Shell-Konfiguration eintragen:

```
# ~/.zshrc
export X_CONSUMER_KEY="..."
export X_CONSUMER_SECRET="..."
export X_ACCESS_TOKEN="..."
export X_ACCESS_TOKEN_SECRET="..."
```

12.2 Post senden (Python)

```python
#!/usr/bin/env python3
"""Post a tweet via X API v2 with OAuth 1.0a."""

import os
from requests_oauthlib import OAuth1Session

oauth = OAuth1Session(
    os.environ['X_CONSUMER_KEY'],
    client_secret=os.environ['X_CONSUMER_SECRET'],
    resource_owner_key=os.environ['X_ACCESS_TOKEN'],
    resource_owner_secret=os.environ['X_ACCESS_TOKEN_SECRET'],
)

payload = {"text": "Gepostet via OpenClaw x-research Skill!"}
resp = oauth.post(
    "https://api.x.com/2/tweets",
    json=payload,
)
print(f"Status: {resp.status_code}")
print(resp.json())
```

Voraussetzung: `pip install requests-oauthlib`

13 Fehlersuche

Problem	Ursache	Lösung
`401 Unauthorized`	Token ungültig oder abgelaufen	Neuen Bearer Token generieren
`429 Too Many Requests`	Rate Limit erreicht	15 Minuten warten
`0 results`	Query zu spezifisch	Weniger Operatoren, breitere Keywords
`0 results`	Token nicht in Shell	`echo $X_BEARER_TOKEN` prüfen
Spam in Ergebnissen	Crypto-Spam	`-$` und `-airdrop -giveaway` hinzufügen
Nur englische Posts	`lang:en` gesetzt	`lang:` entfernen oder ändern
Parser-Fehler	Python-Version	`python3 -version` (min. 3.8)

13.1 Debugging-Befehle

```
# Token pruefen
echo $X_BEARER_TOKEN | head -c 20

# Rohe API-Response anzeigen
curl -s -H "Authorization: Bearer $X_BEARER_TOKEN" \
```

```
  "https://api.x.com/2/tweets/search/recent?query=test&max_results=10"
      \
  | python3 -m json.tool

# Rate Limit Status pruefen (im Response Header)
curl -si -H "Authorization: Bearer $X_BEARER_TOKEN" \
  "https://api.x.com/2/tweets/search/recent?query=test&max_results=10"
      \
  | grep -i "x-rate-limit"
```

14 Dateiübersicht

Zusammenfassung aller Dateien und deren Funktion:

Datei	Funktion
`~/.zshrc`	Bearer Token als Umgebungsvariable
`skills/x-research/SKILL.md`	Skill-Definition mit Triggern und Research-Loop
`skills/x-research/references/x-api.md`	API-Referenz (Endpoints, Operatoren, Limits)
`skills/x-research/references/interests.yaml`	Personalisierte Interessen-Konfiguration
`skills/x-research/scripts/xparse.py`	JSON-Parser mit Filtern und Sortierung
`skills/x-research/scripts/xsearch.sh`	Batch-Suchskript für alle Interessen
`~/clawd/drafts/x-digest-*.md`	Generierte Digest-Dateien