

Bauanleitung: Ihre erste lokale KI

Windows, macOS und Linux · ohne Docker · mit Git, Secrets und Deploy · Stand 11.10.2026

Lesen Sie das hier zuerst

Teil 1 bis 4 bringt Sie in 20 bis 30 Minuten zu einer laufenden lokalen KI auf Ihrem eigenen Rechner — nichts verlässt ihn, solange nur lokale Modelle und lokale Dokumentverarbeitung laufen. Teil 5 bis 8 ist der Schritt danach: eigenes Repository, Secrets und Deploy auf einen Server. Wenn etwas klemmt: Abschnitt 9.

0 · Vorbereitung — am besten zu Hause, nicht im Konferenz-WLAN

- Rund 10 bis 12 GB freier Speicherplatz. Gemessen: allein der Paketcache von Open WebUI belegt 7,5 GB, dazu kommen das Modell mit 1,9 GB und die Laufzeit selbst.
- Administratorrechte für die Installation. Auf Firmenlaptops ist das der häufigste echte Stopper — vorher prüfen.
- Eine gute Internetverbindung für den ERSTEN Start — es werden mehrere Gigabyte geladen. Ab dem zweiten Start geht es ohne Netz, wenn Sie ihn so aufrufen wie am Ende von Abschnitt 3 beschrieben.
- Windows: PowerShell. macOS und Linux: Terminal. Unter Windows funktioniert Git Bash für Schritt 1 NICHT — nehmen Sie PowerShell.

1 · Laufzeit installieren

Die Laufzeit ist das Programm, das ein Sprachmodell auf Ihrem Rechner ausführt. Wir nehmen Ollama. Hier — und nur hier — unterscheiden sich die Systeme.

WINDOWS · PowerShell

```
winget install -e --id ollama.ollama
```

Kein winget auf dem Rechner? Dann das Installationsprogramm von ollama.com/download laden und ausführen.

LINUX / macOS · Terminal

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

Unter macOS geht alternativ auch die .dmg-Datei von ollama.com/download.

PRÜFEN, OB ES GEKLAPPT HAT

```
ollama --version
```

2 · Ein Modell laden — klein anfangen

Ab hier sind alle Befehle auf allen Systemen gleich.

```
ollama pull qwen2.5:3b
```

Der häufigste Anfängerfehler

Nehmen Sie nicht das größte Modell, das Sie finden. 'mistral-small' lädt rund 14 GB herunter und braucht eine Grafikkarte, um brauchbar schnell zu sein. 'qwen2.5:3b' sind 1,9 GB und laufen auf jedem Laptop ohne Grafikkarte. Größer können Sie jederzeit später — der Befehl bleibt derselbe.

Modell	Download	Läuft auf	Taugt für
qwen2.5:0.5b	398 MB	allem	Ausprobieren, sehr einfache Aufgaben
qwen2.5:1.5b	986 MB	allem	Zusammenfassen, sortieren
qwen2.5:3b	1,9 GB	CPU, 8–16 GB RAM	Empfehlung für den Start
qwen2.5:7b	4,7 GB	16 GB RAM oder GPU	spürbar besser, noch erträglich schnell
qwen2.5:14b	9,0 GB	GPU mit 24 GB VRAM	Alltagstauglich im Betrieb

Größenangaben: ollama.com/library/qwen2.5, abgerufen am 04.10.2026.

GLEICH AUSPROBIEREN — NOCH OHNE OBERFLÄCHE

```
ollama run qwen2.5:3b

> Erkläre mir in drei Sätzen, was ein Token ist.

# Beenden mit: /bye
```

3 · Oberfläche danebenstellen — ohne Docker

Open WebUI ist die Oberfläche: ein Chatfenster im Browser, in das Sie auch eigene Dokumente hochladen können. Der folgende Weg braucht weder Docker noch WSL und ist auf allen drei Systemen identisch. 'uv' holt sich die passende Python-Version selbst — das erspart Ihnen die häufigste Fehlerquelle.

EINMALIG, ALLE SYSTEME

```
pip install uv
```

WINDOWS · PowerShell

```
$env:DATA_DIR="$HOME\open-webui-daten"; uvx --python 3.11 open-webui@latest serve
```

LINUX / macOS · Terminal

```
DATA_DIR=~/.open-webui-daten uvx --python 3.11 open-webui@latest serve
```

DATA_DIR nicht weglassen — sonst sind Ihre Daten beim nächsten Start weg

Ohne diese Angabe legt Open WebUI Datenbank, Konto, hochgeladene Dokumente und den Vektor-Index INNERHALB des uv-Paketcaches ab, in einem Ordner mit einer Prüfsumme im Namen. Sobald eine neue Version erscheint — und „@latest“ holt immer die neueste — zeigt der Pfad woanders hin: leere Datenbank, alle Migrationen laufen erneut, Konto und Dokumente sind verschwunden. Mit DATA_DIR liegen die Daten an einer Stelle, die Sie kennen und sichern können.

Der erste Start dauert Minuten — nicht abbrechen

Nach den Datenbank-Meldungen kommt eine Warnung zu CORS. Danach passiert mehrere Minuten lang nichts Sichtbares. Das ist kein Absturz: Open WebUI lädt in dieser Zeit ein Embedding-Modell von Hugging Face nach, bevor es den Port öffnet.

- Das Startsignal ist NICHT die letzte Logzeile, sondern wenn `http://localhost:8080` im Browser antwortet.
- Erst danach Strg+C verwenden. Wer vorher abbricht, fängt beim nächsten Mal wieder von vorn an.
- Der zweite Start ist deutlich schneller: Migrationen sind erledigt, das Modell liegt im Cache.
- Starten Sie immer aus demselben Ordner. Der Sitzungsschlüssel `.webui_secret_key` liegt im Arbeitsverzeichnis, nicht in DATA_DIR — anderer Ordner heißt: neu anmelden.

Ein Beispiel, das die ganze Werkstatt erklärt

Open WebUI gilt als die lokale Oberfläche — und holt sich beim allerersten Start trotzdem ein Modell von einem amerikanischen Dienst. Das ist kein Vorwurf an das Projekt, sondern die Lektion in einem Satz: Lokal heißt nicht, dass nichts hinausgeht. Lokal heißt, dass Sie nachsehen können. Tun Sie es.

Das Fenster bleibt offen, solange die Oberfläche läuft.

Ab dem zweiten Start ohne Netz

Das Anhängsel @latest lässt uv bei JEDEM Start im Netz nach einer neueren Version fragen. Sobald alles einmal gelaufen ist, lassen Sie es weg und setzen --offline. Dann wird ausschließlich aus dem lokalen Cache gestartet, und es geht keine Anfrage mehr hinaus.

```
# Windows · PowerShell
$env:DATA_DIR="$HOME\open-webui-daten"; uvx --offline --python 3.11 open-webui serve

# Linux / macOS · Terminal
DATA_DIR=~/.open-webui-daten uvx --offline --python 3.11 open-webui serve
```

- Zusätzlich OFFLINE_MODE=true setzen, damit auch Open WebUI selbst nicht mehr bei Hugging Face nachfragt. Das wirkt erst, wenn das Modell schon im Cache liegt — also nach dem ersten erfolgreichen Start.
- Dieser Ablauf ist nachgemessen: ohne @latest und mit --offline startet die Oberfläche ohne jede Netzanfrage.

Alternative A — ohne uv

```
pip install open-webui

# ... unter Windows, in PowerShell:
$env:DATA_DIR="$HOME\open-webui-daten"; open-webui serve

# ... unter Linux und macOS, im Terminal:
DATA_DIR=~/.open-webui-daten open-webui serve
```

Achtung: Diese Variante braucht Python 3.11 oder 3.12. Mit Python 3.13 bricht die Installation ab. Prüfen mit 'python --version'.

Alternative B — mit Docker, falls Sie Docker ohnehin nutzen

```
docker run -d -p 3000:8080 \
  --add-host=host.docker.internal:host-gateway \
  -v open-webui:/app/backend/data \
  --name open-webui ghcr.io/open-webui/open-webui:main
```

Der Backslash am Zeilenende trennt einen langen Befehl — das funktioniert im Terminal von Linux und macOS. In PowerShell funktioniert er NICHT: dort die vier Zeilen ohne Backslashes zu einer einzigen zusammenziehen. Und Docker Desktop muss laufen, bevor Sie den Befehl absetzen.

4 · Öffnen und eigenes Wissen hochladen

Installiert mit	Adresse im Browser
uv oder pip	http://localhost:8080
Docker	http://localhost:3000

- Beim ersten Aufruf legen Sie ein Konto an. Das bleibt auf Ihrem Rechner — es geht nirgendwohin.
- Oben rechts das Modell auswählen: qwen2.5:3b.
- Über Arbeitsbereich → Wissen eine eigene Datei hochladen (PDF, Word, Text).
- Im Chat die Sammlung mit # ansprechen und eine Frage dazu stellen.

Jetzt haben Sie das Prinzip in der Hand

Ein Modell, das auf Ihrem Rechner rechnet. Eine Oberfläche, die Ihnen gehört. Ihre Dokumente, die den Rechner nicht verlassen. Alles Weitere — Repository, Freigaben, Server — ist Betrieb um genau diesen Kern herum.

5 · Die Werkstatt — Ihr eigenes Repository

Sobald Sie mehr machen als ausprobieren, brauchen Sie Git. Nicht weil es modern ist, sondern weil ein Eigenbau ohne Versionsverwaltung nicht wartbar ist: Sie können nicht sehen, was sich geändert hat, und nicht zurück, wenn etwas kaputtgeht.

Zuerst: drei Ordner, drei Zwecke

Ordner	Was darin liegt	Ins Repository?
Ihr Projektordner, z. B. dev/mein-ki-projekt	Startskript, README, .gitignore	Ja — das IST das Repository
Ihr Datenordner, z. B. open-webui-daten	Datenbank, hochgeladene Dokumente, Index	Nie. Gehört ins Backup.
Der Paketcache (legt uv selbst an)	die Programmdateien von Open WebUI	Nein, das ist nicht Ihr Code

Warum der Datenordner außerhalb des Projektordners liegt

In „uploads“ liegen die Dokumente, die Sie hochladen — bei einem Pilotbetrieb also echte Unterlagen. In der Datenbank stehen Konten. Liegt dieser Ordner neben Ihrem Code, fängt ihn irgendwann ein unbedachtes „git add.“ ein, und dann steht er für immer in der Historie. Deshalb: DATA_DIR zeigt bewusst woanders hin.

Lesen Sie diesen Kasten, bevor Sie etwas abtippen

Manche der folgenden Blöcke sind DATEIINHALT, keine Befehle. Sie erkennen es an der Zeile „cat > dateiname <<'EOF'“ davor: Das ist der Befehl, der die Datei anlegt. Alles danach bis zum abschließenden EOF ist der Inhalt. Markieren Sie den ganzen Block und fügen Sie ihn in einem Stück ein — Zeile für Zeile funktioniert nicht, und einzelne Zeilen allein ergeben nur Fehlermeldungen wie „command not found“.

Schritt 1 — In den Projektordner wechseln und Git starten

```
cd ~/dev/mein-ki-projekt
git init
```

Git meldet „Initialized empty Git repository“. Ab jetzt steht hinter dem Ordnernamen in Ihrer Eingabeaufforderung der Zweigname, meist (master) oder (main). Daran sehen Sie, dass Sie im Repository sind.

Schritt 2 — Die .gitignore anlegen (das Wichtigste zuerst)

Diese Datei bestimmt, was Git NIEMALS anfasst. Sie muss vor dem ersten Commit stehen.

```
cat > .gitignore <<'EOF'
.webui_secret_key
.env
*.key
*.pem
open-webui-daten/
data/
uploads/
*.db
*.sqlite
backups/
node_modules/
__pycache__/
EOF
```

Kontrolle: „cat .gitignore“ zeigt den Inhalt. Unter Windows in PowerShell statt des Blocks oben: die Datei mit „notepad .gitignore“ anlegen und die Zeilen zwischen den EOF-Markierungen hineinschreiben.

Schritt 3 — Zeilenenden festlegen

Unter Windows wandelt Git Zeilenenden um. Bei einem Shell-Skript macht das aus einer funktionierenden Datei eine kaputte — beim Start kommt dann „bad interpreter: /usr/bin/env bash^M“. Diese eine Datei verhindert das:

```
cat > .gitattributes <<'EOF'
* text=auto
*.sh text eol=lf
*.ps1 text eol=crlf
EOF
```

Schritt 4 — Das Startskript anlegen

Damit Sie sich DATA_DIR nie wieder merken müssen. Nehmen Sie die Fassung für die Umgebung, in der Sie tatsächlich arbeiten.

GIT BASH UNTER WINDOWS, macOS UND LINUX

```
cat > start.sh <<'EOF'
#!/usr/bin/env bash
export DATA_DIR="$(cygpath -m "$HOME")/open-webui-daten"
echo "Daten liegen in: $DATA_DIR"
uvx --offline --python 3.11 open-webui serve
EOF
chmod +x start.sh
```

Unter macOS und Linux gibt es „cygpath“ nicht — dort stattdessen einfach: `export DATA_DIR="$HOME/open-webui-daten"`. Die `echo`-Zeile zeigt Ihnen beim Start, wo die Daten wirklich landen. Schauen Sie beim ersten Mal hin; steht dort etwas Unerwartetes, tragen Sie den Pfad fest ein.

WINDOWS · POWERSHELL

```
notepad start.ps1

# ... und in das Fenster diese zwei Zeilen schreiben, dann speichern:
$env:DATA_DIR = "$HOME\open-webui-daten"
uvx --offline --python 3.11 open-webui serve
```

Gestartet wird danach mit „./start.sh“ beziehungsweise „.\start.ps1“ aus dem Projektordner.

Ein Hinweis zu `--offline`: Das funktioniert erst, wenn die Oberfläche einmal mit Netz gestartet ist — sonst liegt nichts im Cache. Beim allerersten Mal also ohne `--offline` und mit `@latest` starten, danach bleibt das Skript, wie es hier steht.

Schritt 5 — Den ersten Stand sichern und prüfen

```
git add .gitignore .gitattributes start.sh
git commit -m "Startskript und Schutz vor versehentlichen Datencommits"
git status
```

Die eine Kontrolle, die Sie nicht überspringen

Nach dem Commit muss „git status“ melden: `nothing to commit, working tree clean`. Taucht dort `.webui_secret_key`, `open-webui-daten/` oder eine `.db`-Datei auf, greift Ihre `.gitignore` nicht — dann aufhören und erst das klären. Was einmal committet ist, bekommen Sie nicht mehr sauber heraus.

Was ins Repository gehört — und was nie

Gehört hinein	Gehört nie hinein
Startskript, Konfigurationsvorlagen	Datenbank, hochgeladene Dokumente
README mit zwei Sätzen zum Start	.webui_secret_key und jeder andere Schlüssel
eigener Code, eigene Oberfläche	.env mit echten Werten
.gitignore und .gitattributes	Sicherungen und Exporte

Der Merksatz dahinter: Ins Repository gehört die Anleitung, nicht die Daten. Wer seine Datenbank eincheckt, hat kein Backup — er hat ein Leck mit Versionsverlauf.

Wo das Repository liegt — dieselbe Souveränitätsfrage wie beim Modell

Variante	Wofür	Was Sie abgeben
GitHub / GitLab (gehostet)	schneller Start, gute Werkzeuge	Ihr Quellcode liegt beim Anbieter
GitLab self-hosted	große Teams, viel Funktion	nichts — kostet aber Betrieb
Gitea / Forgejo	Mittelstand, ein Server, schlank	nichts — läuft neben Ihrer KI mit

Private Repositories sind bei den gehosteten Anbietern kostenlos — aber privat heißt nicht souverän. Für Quellcode mit Geschäftslogik gilt dieselbe Frage wie für Daten: Wer kann abschalten, und was passiert dann?

Mit einem entfernten Repository verbinden

```
git remote add origin git@github.com:IHR-KONTO/ihr-projekt.git
git branch -M main
git push -u origin main
```

Gitea im eigenen Haus — falls der Code nicht nach außen soll

```
docker run -d --name gitea -p 3001:3000 -p 222:22 \
-v gitea-data:/data \
--restart always gitea/gitea:latest

# danach: http://localhost:3001 im Browser, Konto anlegen
```

6 · Secrets — die teuerste Nachlässigkeit

Die Regel, an der die meisten Projekte einmal scheitern

Zugangsdaten gehören NIE ins Repository. Nicht in den Code, nicht in eine Konfigurationsdatei, nicht „nur kurz zum Testen“. Git vergisst nichts: Ein einmal eingecheckter Schlüssel steht in der Historie, auch nachdem Sie ihn gelöscht haben. Wer es doch tut, muss den Schlüssel beim Anbieter widerrufen — Löschen allein reicht nicht.

Die .gitignore aus Abschnitt 5 ist der technische Teil. Der zweite Teil ist eine Gewohnheit: Echte Werte stehen in einer Datei .env, die nicht im Repository liegt. Im Repository liegt nur eine Musterdatei ohne Werte — damit ein Kollege weiß, welche Variablen er setzen muss.

```
cat > .env.example <<'EOF'
MODEL_PROVIDER=ollama
OLLAMA_HOST=http://127.0.0.1:11434
EU_API_KEY=                # leer lassen, echter Wert nur in .env
WEBUI_SECRET_KEY=
EOF
```

- Einen vergessenen Schlüssel finden Sie mit: `git log -p | grep -i "api_key"`
- Wenn doch etwas durchgerutscht ist: Schlüssel beim Anbieter sofort widerrufen und neu erzeugen. Erst danach die Historie bereinigen.

7 · Der Arbeitsablauf — so entsteht der Code

So arbeite ich an dem System, das Sie im Workshop gesehen haben. Die KI schlägt Code vor, ich prüfe ihn, ich führe ihn zusammen, und ich bringe ihn auf den Server. Das ist dasselbe Prinzip wie im laufenden System: Die Maschine schlägt vor, der Mensch drückt den Knopf — nur eine Etage tiefer.

Schritt	Wer	Was passiert
1 · Zweig anlegen	Mensch	git switch -c funktion/kundenakte
2 · Code entwerfen	KI	Vorschlag im Zweig, nie direkt auf main
3 · Prüfen	Mensch	lesen, Tests laufen lassen, verstehen
4 · Zusammenführen	Mensch	Merge nach main — der Knopf gehört Ihnen
5 · Ausrollen	Mensch	per SSH auf den Server, siehe Abschnitt 8

```
git switch -c funktion/kundenakte
# ... arbeiten, prüfen, testen ...
git add -A
git commit -m "Kundenakte: Belege je Aussage"
git switch main
git merge funktion/kundenakte
git push
```

Warum der Merge beim Menschen bleibt

Ein Vorschlag, den niemand gelesen hat, ist kein geprüfter Code — er ist nur schnell entstandener Code. Solange ein Mensch zusammenführt, ist ein Fehler eine Korrektur. Ohne diesen Schritt ist derselbe Fehler ein Vorfall im Betrieb. Genau deshalb gibt es diese Trennung auch im laufenden System.

8 · Vom Laptop auf einen Server

Erst wenn mehrere Leute gleichzeitig arbeiten, lohnt der Server. Vorher kostet er nur Zeit. Ein kleiner VPS mit 6 Kernen und 8 GB RAM reicht für den Anfang — ohne Grafikkarte.

Zugang — Schlüssel statt Passwort

```
# Auf Ihrem Rechner einen Schlüssel erzeugen:
ssh-keygen -t ed25519 -C "ihr-name@ihre-firma"

# Öffentlichen Teil auf den Server bringen:
ssh-copy-id benutzer@ihr-server

# Danach anmelden:
ssh benutzer@ihr-server
```

- Passwort-Anmeldung danach abschalten. Ein Server mit Passwort-Login wird innerhalb von Stunden durchprobiert.
- Niemals als root arbeiten. Einen eigenen Benutzer anlegen und per sudo erhöhen.

Ausrollen — der immer gleiche Dreischritt

```
ssh benutzer@ihr-server
cd /opt/ihr-projekt
git pull
docker compose up -d --build

# Prüfen, ob alles läuft:
docker compose ps
docker compose logs -f --tail=50
```

Eine Tür nach außen — Reverse-Proxy mit TLS

Genau ein Dienst bekommt offene Ports, alles andere liegt dahinter. Caddy holt und erneuert die Zertifikate von selbst — das ist der kürzeste Weg zu sauberem HTTPS.

```
# Datei: Caddyfile

ki.ihre-firma.de {
    reverse_proxy localhost:8080
}
```

Sicherung — bevor echte Daten hineinkommen, nicht danach

```
# Nächtlich, verschlüsselt, außer Haus – Schlüssel bleibt bei Ihnen
tar czf - /opt/ihr-projekt/data \
  | age -r age1ihr-oeffentlicher-schluessel \
  > /tmp/backup-$(date +%F).tar.gz.age

rclone copy /tmp/backup-$(date +%F).tar.gz.age ihr-speicher:backups/
```

Die Sicherung, die nie getestet wurde, ist keine

Stellen Sie einmal im Quartal eine Sicherung auf einem leeren System wieder her. Das dauert eine Stunde und ist der einzige Beweis, dass Ihr Backup funktioniert. Und: Wenn der Schlüssel beim Speicheranbieter liegt, sichern Sie für jemand anderen mit.

Was spätestens jetzt dazugehört

- Update-Plan: Wer patcht was, wie oft? Jede Komponente ist Angriffsfläche — auch die quelloffene.
- Freigaben: Lesen ist frei, Schreiben bekommt einen eigenen Schalter, Vorgabe aus.
- Protokoll ohne Inhalt: nachvollziehbar, aber ohne zweite Datenhalde mit Kundendaten.
- Ein benannter Mensch, der das betreibt. Ohne den ist nichts davon haltbar.

9 · Wenn etwas klemmt

■ This script is intended to run on Linux and macOS only.

Ursache Sie haben den Linux-Befehl unter Windows ausgeführt — meist in Git Bash.
Lösung PowerShell öffnen und 'winget install -e --id Ollama.Ollama' nehmen.

■ failed to connect to the docker API ... dockerDesktopLinuxEngine

Ursache Docker Desktop ist nicht gestartet oder gar nicht installiert.
Lösung Den Weg ohne Docker aus Abschnitt 3 nehmen. Oder Docker Desktop starten und warten, bis es grün meldet.

■ 'winget' wird nicht als Name eines Cmdlets erkannt

Ursache Der Windows-Paketmanager fehlt (ältere Windows-Installationen).
Lösung Installationsprogramm direkt von ollama.com/download laden.

■ ERROR: Could not find a version that satisfies open-webui

Ursache Ihre Python-Version ist zu neu. Mit 3.13 läuft Open WebUI noch nicht.
Lösung Den uv-Weg aus Abschnitt 3 nehmen — der bringt sein eigenes Python 3.11 mit.

■ Die Seite im Browser bleibt leer

Ursache Falscher Port. uv und pip starten auf 8080, Docker bildet auf 3000 ab.
Lösung Beide Adressen probieren. Und prüfen, ob das Fenster mit 'open-webui serve' noch offen ist.

■ Das Modell antwortet, aber sehr langsam

Ursache Normal auf einer CPU ohne Grafikkarte. Das ist kein Fehler.
Lösung Ein kleineres Modell nehmen: qwen2.5:1.5b. Oder akzeptieren — zum Zusammenfassen reicht es.

■ Open WebUI findet kein Modell

Ursache Die Oberfläche erreicht Ollama nicht.
Lösung Prüfen mit 'ollama list' im zweiten Fenster. Unter Windows läuft Ollama als Dienst im Hintergrund.

■ git push: Permission denied (publickey)

Ursache Der Server kennt Ihren SSH-Schlüssel nicht.
Lösung Öffentlichen Schlüssel aus ~/.ssh/id_ed25519.pub im Git-Konto hinterlegen. Test: `ssh -T git@github.com`

■ Nach der CORS-Warnung passiert minutenlang nichts

Ursache Kein Fehler. Open WebUI lädt im Hintergrund ein Embedding-Modell von Hugging Face, bevor es den Port öffnet.
Lösung Warten, bis `http://localhost:8080` antwortet. Das ist das Startsignal, nicht die letzte Logzeile.

■ Konto und hochgeladene Dokumente sind nach einem Neustart weg

Ursache Open WebUI lief ohne DATA_DIR. Die Daten lagen im uv-Paketcache und sind mit der nächsten Version umgezogen.
Lösung Immer mit DATA_DIR starten, siehe Abschnitt 3. Was im Cache lag, ist verloren — einmal neu anlegen.

■ bash: .webui_secret_key: command not found

Ursache Die Zeilen eines cat-Blocks wurden einzeln eingefügt. Die Shell hat sie als Befehle gelesen, nicht als Dateiinhalt.
Lösung Den Block ab 'cat >' bis einschließlich EOF am Stück markieren und in einem Zug einfügen. Danach prüfen: `cat .gitignore`

■ warning: LF will be replaced by CRLF / bad interpreter: ...bash^M

Ursache Git hat unter Windows die Zeilenenden des Startskripts umgeschrieben. Das Skript ist damit nicht mehr startbar.
Lösung Die .gitattributes aus Schritt 3 anlegen, dann einmal: `git rm --cached start.sh` und `git add start.sh`

■ Ein Zugangsschlüssel ist versehentlich im Repository gelandet

Ursache Eine Datei mit Werten wurde eingecheckt, bevor .gitignore stand.
Lösung Schlüssel beim Anbieter SOFORT widerrufen und neu erzeugen. Löschen allein genügt nicht — die Historie bleibt.

10 · Alle Befehle auf einen Blick

```
# 1 Laufzeit (Windows, PowerShell)
winget install -e --id Ollama.Ollama

# 1 Laufzeit (Linux / macOS, Terminal)
curl -fsSL https://ollama.com/install.sh | sh

# 2 Modell (alle Systeme)
ollama pull qwen2.5:3b

# 3 Oberfläche (ohne Docker) – DATA_DIR nicht weglassen!
pip install uv

# ... unter Windows, in PowerShell:
$env:DATA_DIR="$HOME\open-webui-daten"; uvx --python 3.11 open-webui@latest serve

# ... unter Linux und macOS, im Terminal:
DATA_DIR=~/.open-webui-daten uvx --python 3.11 open-webui@latest serve

# 4 Öffnen
http://localhost:8080

# 5 Werkstatt – ERST .gitignore, DANN der erste Commit!
git init
git add .gitignore .gitattributes start.sh
git commit -m "Erster Stand"
git status # muss 'working tree clean' melden

# 8 Ausrollen
ssh benutzer@ihr-server
cd /opt/ihr-projekt && git pull && docker compose up -d --build
```

Diese Bauanleitung darf frei weitergegeben werden. Keine Registrierung, keine E-Mail-Abfrage.