# eigene Programme, headless

Linux, Bash, Serversysteme.

Diese Scripte nutzen ausnahmslos den Auslagerungsordner /home/$NAME/script-data/script-name/$0.
Der Stand des Scripts ist in Klammern versehen der einzelnen Scripts.
Die Distribution der Scripte findet sich ebenfalls in den Klammern.

- IP Logger (Debian / 04.01.2024)
- M4A zu MP3 Massenumwandlung (Debian / 09.04.2025)
- DynDNS Updater (Debian / 10.07.2024)
- Webcamloader □□
- Faultnotify □□
- IPlogger □□

# IP Logger (Debian / 04.01.2024)

Dieser IP Logger loggt die eigene IP-Adresse in eine CSV-Datei. Dies hat den Grund, dass man Vorwürfe jeder Art durch Vorlage der zu der Zeit genutzten IP-Adresse vorweisen könnte. Nützlich in Verbindung mit einem Cronjob.

Das Script holt via "icanhazip.com" die aktuelle IP-Adresse und speichert sie in die CSV-Datei. Wenn via Cronjob das Script ausgeführt wird, empfehle ich den Intervall auf stündlich zu setzen. Eine Zeit von-bis einzubauen halte ich für nicht sinnvoll, da das Internet zu dieser Zeit auch ausfallen könnte. Daher halte ich es für nicht sinnvoll, die IP-Adresse zu speichern und erst bei Änderung erneut zu schreiben.

## □□ Was macht das Script?

### □□ Allgemeine Initialisierung:

- Bestimme den aktuellen Benutzer (`whoami`)
- Setze `$data` auf:
  `/home/<user>/script-data/<scriptname>`
  (z. B. `/home/pi/script-data/iplogger.sh`)
- Setze `$ippage` auf `icanhazip.com`
- Hole aktuelle öffentliche IP via `curl`

### □□ Benötigte Tools:

- `curl`
- `nc` (netcat)
- `sed`, `awk`, `grep`, `uniq`
- Bash >= 4

Script:

```
#!/bin/bash

user=$(whoami)
data="/home/$user/script-data/$(basename "$0")"
ippage="icanhazip.com"
ipneu=$(curl -s "$ippage")

if [ ! -d "$data" ]; then
	mkdir -p "$data"
fi

if [ -n "$1" ] && [ "$1" = "suche" ]; then
	cat "$data/ipdb.csv" | sed 's/;/ /g' | sed 's/\([0-9]\{4\}\)-\([0-9]\{2\}\)-\([0-9]\{2\}\)/\3.\2.\1/' | grep "$2" | awk '{print $1, "", "", $3}' | uniq
	exit
	else
	if nc -z -w 1 "$ippage" 443 2>/dev/null; then
		echo "$(date +"%Y-%m-%d;%H:%M");$ipneu" >> "$data/ipdb.csv"
		echo "OK: $ipneu"
	else
		if nc -z -w 1 "google.de" 443 2>/dev/null; then
			echo "Fehler, $ippage nicht erreichbar."
			echo "$(date +"%Y-%m-%d;%H:%M");$ippage nicht erreichbar" >> "$data/ipdb.csv"
		else
			echo "Kein Internet."
			echo "$(date +"%Y-%m-%d;%H:%M");offline" >> "$data/ipdb.csv"
		fi
	fi
fi
```

# M4A zu MP3 Massenumwandlung (Debian / 09.04.2025)

Das Script wandelt rekursiv beginnend ab einem Pfad ($1) M4A-Audiodateien in MP3 um. Die Qualität bleibt nahezu gleich. Der Pfad ist dem $1-Argument zu übergeben.
Die daraus entstehende MP3 ist gleichnamig wie die M4A Datei.

ffmpeg und find werden dafür benötigt.

Script:

```
#!/bin/bash

DIR="$1"

if [ -z "$DIR" ] || [ ! -d "$DIR" ]; then
    echo "Usage: $0 /pfad/zum/verzeichnis"
    exit 1
fi

find "$DIR" -type f -iname "*.m4a" | while read -r M4A; do
    MP3="${M4A%.m4a}.mp3"

    # Nur umwandeln, wenn mp3 nicht schon existiert
    if [ -f "$MP3" ]; then
        echo "Überspringe (bereits vorhanden): $MP3"
        continue
    fi

    echo "Wandle um: $M4A → $MP3"
```

```
    ffmpeg -i "$M4A" -codec:a libmp3lame -qscale:a 4 "$MP3" -y < /dev/null

    if [ $? -eq 0 ]; then
        echo "Erfolg: $MP3"
    else
        echo "Fehler bei: $M4A"
    fi
done
```

# DynDNS Updater (Debian / 10.07.2024)

Der DynDNS Updater aktualisiert eine DynDNS-Domain mit der aktuellen IP-Adresse.
Wenn das Script zum ersten Mal ausgeführt wird, muss man die API zur Aktualisierung eingeben, die dieses Script dann aufruft.
Diese API wird dann im script-data-Ordner gespeichert.

**Genaue Zusammenfassung:**
*data=/home/$USER/script-data/$0*

Wenn noch keine IP vorhanden ist ($data/ip.txt), dann wird die IP-Adresse von "icanhazip.com" abgerufen.
API wird aufgerufen, IP wurde aktualisiert.
Jeder Schritt wird mitgeloggt in $data/allgemein.csv (besondere Ereignisse), tagesaktuell in $data/logs/YYYYMMTT.csv (Updates).

## □□ Was macht das Script?

### □□ Lade bisherigen Status:

- Wenn `ip.txt` vorhanden ist, wird die bisherige öffentliche IP geladen.
- Wenn nicht: → erste Ausführung → IP bleibt leer, Log-Eintrag in `allgemein.csv`.

---

### □□ Prüfe Update-URL:

- Wenn `updateurl.txt` existiert → lese erste Zeile ein.
- Wenn nicht vorhanden:
  - Benutzer wird zur Eingabe einer gültigen `http(s)`-URL aufgefordert.
  - Wenn gültig: speichere in `updateurl.txt` & logge es.
  - Wenn ungültig: Abbruch & Fehlerlog.

---

### □□ Online-Check:

- Ping an `icanhazip.com` (IPv4, 1 Paket, Timeout 3s)
- Wenn erreichbar:
  - Hole neue IP mit `curl`
  - Vergleiche mit vorheriger IP
    - Unverändert: nur Statusmeldung
    - Geändert:
      - speichere neue IP in `ip.txt`
      - rufe `updateurl` auf (z. B. DynDNS-Anbieter)
      - logge Aktualisierung in Tageslog
      - sende E-Mail-Benachrichtigung
- Wenn nicht online:
  - logge, dass keine Aktualisierung möglich ist

# □□ Benötigte Tools:

- `curl`
- `ping`
- `mail`
- `sed`, `head`, `tail`
- Bash >= 4

Inhalt der $data/updateurl.txt:

```
https://api.dyndns.tld/update.php?key=q12w3e4r5t6z7u8i9o0p
```

Script:

```
user=$(whoami)
data="/home/$user/script-data/$(basename "$0")"
ippage="icanhazip.com"

□if [ ! -d "$data" ]; then
□□mkdir -p "$data"
□fi

□if [ ! -d "$data/logs" ]; then
□□mkdir "$data/logs"
□fi
```

```
function ipcheck {
	echo "$(date +"[%d.%m.%Y %H:%M:%S]")"

	if [ -f "$data/ip.txt" ]; then
		ip=$(head -n1 "$data/ip.txt")
		echo "Bisherige IP: $ip"
	else
		echo "Erste Ausführung."
		ip=""
		echo "$(date +"%d.%m.%Y;%H:%M:%S");Erste Ausführung;$1" >> "$data/logs/allgemein.csv"
	fi

	if [ -f "$data/updateurl.txt" ]; then
		updateurl=$(sed 's/ //g' "$data/updateurl.txt" | head -n1 | tail -n1)
	else
		echo "Keine Update-URL hinterlegt. URL hier einfügen:"
		echo "$(date +"%d.%m.%Y;%H:%M:%S");Update-URL fehlt" >> "$data/logs/allgemein.csv"
		read -p "Update-URL: " updateurl

		# Überprüfe, ob die URL mit "http://" oder "https://" beginnt
		if [[ "$updateurl" =~ ^(http|https):// ]]; then
			echo "URL OK."
			echo "$updateurl" > "$data/updateurl.txt"
			echo "$(date +"%d.%m.%Y;%H:%M:%S");Update-URL hinterlegt;$1" >> "$data/logs/allgemein.csv"
		else
			echo "Die URL ist ungültig."
			echo "$(date +"%d.%m.%Y;%H:%M:%S");Update-URL ungültig;$1" >> "$data/logs/allgemein.csv"
			exit 1
		fi
	fi

	echo "Prüfe Online-Status"
	if ping -4 -c 1 -W 3 "$ippage" >/dev/null 2>&1; then
		echo "Anschluss Online"
		ipneu=$(curl -s "$ippage")
		echo "IP: $ipneu"

		if [ "$ip" = "$ipneu" ]; then
			echo "IP unverändert."
```

```
	#		echo "$(date +"%d.%m.%Y;%H:%M:%S");IP unverändert;$1" >> "$data/logs/$(date +"%Y%m%d").csv"
		else
			echo "Bisherige IP: $ip"
			echo "Neue IP     : $ipneu"
			echo "Aktualisiere..."
			echo "$ipneu" > "$data/ip.txt"
			curl -sSL "$updateurl" >/dev/null
			echo "OK."
			echo "$(date +"%d.%m.%Y;%H:%M:%S");Aktualisiert auf $ipneu;$1" >> "$data/logs/$(date +"%Y%m%d").csv"
			echo "$(date +"[%d.%m.%Y %H:%M:%S]") IP aktualisiert auf $ipneu" | mail -s "DynDNSupdater"
"mail@mariobeh.de"
		fi
	else
		echo "Client ist selbst offline, kein Aktualisieren möglich."
		echo "$(date +"%d.%m.%Y;%H:%M:%S");Client offline;$1" >> "$data/logs/$(date +"%Y%m%d").csv"
	fi
}



if [ -z "$1" ]; then
	ipcheck
fi

exit 0
```

# Webcamloader □□

**Webcamloader, Debian (+Derivate)**
***Wiki-Stand: 27.12.2025***
***Script-Stand: 28.12.2025***

***Download***

---

# □□ Vorwort

Der **Webcamloader** ist ein universelles Bash-Script zur automatisierten Erfassung von Kamerabildern. Es eignet sich ideal für den Einsatz bei **Baufortschrittsdokumentationen**, **Wetter- und Landschaftsbeobachtungen**, **Langzeitstudien** und ähnlichen Zeitraffer-Projekten.

Das Script lädt in regelmäßigen Intervallen **Einzelbilder von Bildquellen oder Videostreams** herunter und speichert diese lokal. Die Bilder können später zu einem Zeitraffer-Video weiterverarbeitet werden.

Unterstützte Quellen: **Bildquellen:** z. B. `snapshot.jpg`, `snapshot.cgi` und **Videostreams:** z. B. `faststream`, `motion-jpeg`, `video.cgi`.

Das Script speichert alle Daten zentral unter `/home/$USER/script-data/webcamloader/`
Darin befinden sich: **Projektverzeichnisse, Statusdaten, Logfiles und Konfigurationen**.
Optional kann ein abweichender **Medienordner** (z. B. auf einer externen Festplatte) angegeben werden. Falls keiner definiert ist, wird der Medienpfad automatisch unterhalb des `script-data/webcamloader/`-Verzeichnisses angelegt.

Bitte unbedingt Datenschutzbestimmungen beachten. Mehr dazu weiter unten im Haftungsausschluss.

# ⚙ Funktionen im Wesentlichen

Das Script prüft zu Beginn, ob die Kamera erreichbar ist, lädt ein Testbild, errechnet anhand diesem den erforderlichen Speicher für die angegebene Gesamtbildanzahl, erstellt im nächsten Schritt Projektordner und Statusdatei und nimmt die Arbeit auf.
Falls ein Zeitfenster angegeben wurde, wird dieses berücksichtigt.

Im Fehlerfall wird der Benutzer benachrichtigt per Email, sofern angegeben.

Wenn ein Projekt abgeschlossen wurde, erhält man wahlweise eine Mail. Daraufhin kann man über einen Menüpunkt ein Video erstellen. Hierfür sind die FPS (Frame per second) nötig anzugeben.

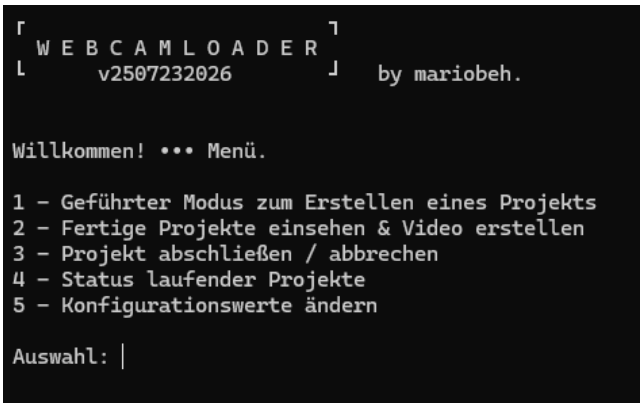
Das Script bietet drei Modi zum Besorgen der Bilder, die weiter unten genauer erklärt werden:
• **Menü**: Interaktives Menü mit allen Funktionen.
• **Quicky**: Komplette Steuerung via Argumente, kein Menü.
• **Cron**: Einmalaufrufe, zeitgesteuert z. B. über Crontab.
Für die **Videoerstellung** steht ein geführter Ablauf über das Menü zur Verfügung, der Schritt für Schritt und intuitiv durch den Prozess führt.

# ☝ Technische Hinweise

• Die **Kamera-URL muss direkt** auf ein Bild oder einen Videostream verweisen – nicht auf eine HTML-Seite oder Steueroberfläche.
• Für **E-Mail-Benachrichtigungen** muss der Server `mail` installiert und korrekt eingerichtet haben.
• Bei dem **Funktionsschalter** gilt: -f 0 ist Hintergrundmodus (keine Ausgabe), -f 1 ist Vordergrundmodus, komplette Ausgabe. Falls Ausgabe erwünscht ist, wird das Programm `screen` empfohlen, da beim Schließen des Terminals auch das Script beendet wird.
• Wird keine Funktion -f angegeben, startet automatisch der Vordergrundmodus.
• **Hinweis:** Nicht jede Kamera wird unterstützt, da manche Videostreams nicht mit dem Script kompatibel sind. In solchen Fällen liefert die Kamera entweder ungeeignetes Material oder einen endlosen Stream, der das Script dauerhaft blockiert. Dieses Verhalten ist technisch bedingt und wurde im Rahmen der Möglichkeiten umfassend getestet.
Wenn die Kamera ein klares Einzelbild liefert, funktioniert alles reibungslos, bei Videostreams kann es jedoch zu Problemen kommen. Sollte ein inkompatibler Stream erkannt werden, verweigert das Script automatisch die weitere Verarbeitung.

Das Script ist ressourcenschonend und benötigt keine Root-Rechte. Es erzeugt keine externen Weiterleitungen außer zur Kamera selbst.
Die heruntergeladenen Bilder werden im vorher festgelegten Medien-Ordner gespeichert und können dort jederzeit angeschaut werden. Videos landen ebenfalls dort.

# □□ Das Menü



Hier sieht man das Hauptmenü, welches beim Aufruf ohne Parameter gestartet wird. Dieses ist nötig für den geführten Modus, der Videoerstellung, Projekte abzubrechen oder den Status laufender Projekte zu begutachten.

# □□ Geführter Modus □□□

Der geführte Modus über das Hauptmenü ist die zentrale und **benutzerfreundlichste** Funktion des Webcamloaders.
Wird das Skript ohne Parameter aufgerufen, startet es automatisch im interaktiven Hauptmenü.

Hier erfolgt die Bedienung schrittweise und intuitiv – ganz ohne Kenntnisse über Parameter oder deren Syntax. Besonders für Einsteiger ist dieser Modus ideal: Alle Einstellungen werden nacheinander abgefragt und erklärt, sodass man sicher und zielgerichtet zum gewünschten Ergebnis kommt.

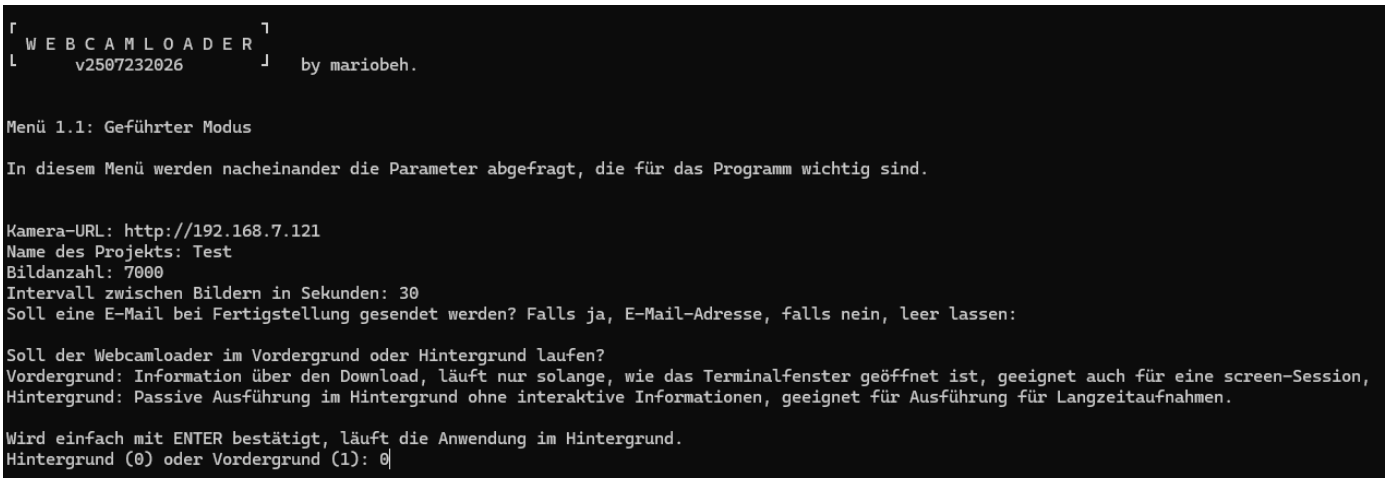
Im Menü kann zwischen verschiedenen Aktionen gewählt werden: Ein neues Projekt starten, ein bestehendes Projekt fortsetzen, Einstellungen ändern, den Status prüfen oder das Projekt abbrechen. Auch ein Video lässt sich hier erstellen, diese Funktion ist im Videosektor genauer erklärt. Der Einstieg erfolgt über die Eingabe grundlegender Informationen wie Kamera-URL, Projektname, Anzahl der Bilder und Aufnahmeintervall. Optional kann ein Zeitfenster für die aktiven Aufnahmezeiten angegeben werden. Außerdem besteht die Möglichkeit, eine Benachrichtigungs-Mailadresse zu hinterlegen und zu entscheiden, ob die Ausgabe im Vordergrund oder still im Hintergrund erfolgen soll.

Sobald alle Angaben gemacht wurden, erhält man eine Übersicht mit geschätztem Speicherbedarf und Laufzeit. Mit einem einfachen Tastendruck startet die Aufzeichnung. Auch nach Projektstart können noch Informationen zum Projektstatus abgerufen oder Einstellungen angepasst werden, etwa um das Projekt frühzeitig zu beenden oder ein neues zu beginnen.

Der Menümodus ist somit der flexibelste und komfortabelste Weg, ein Zeitraffer-Projekt zu starten, und bietet zugleich volle Kontrolle über den Ablauf – ohne direkte Kommandozeilenparameter.

***Hinweis: im geführtem Modus wird derzeit die Funktion des Zeitfensters und der Zeitzone noch nicht unterstüzt. Wird diese Funktion benötigt, bitte ich den Quicky-***

***Modus** zu benutzen.*



Hier sieht man die Durchführung des geführten Modus', bei dem beispielhaft Daten eingetragen wurden

# □□ Quicky-Modus □□

Der Quicky-Modus ist für erfahrene Benutzer gedacht, die ein Projekt schnell starten möchten. Nach Übergabe aller erforderlichen Parameter zeigt der Webcamloader eine Zusammenfassung inklusive Berechnung der Aufnahmedauer. Mit ENTER kann das Projekt sofort gestartet werden.

Aufruf:

```
./webcamloader.sh quicky -u <URL> -n <Name> -b <Anzahl> -i <Intervall> -e <E-Mail> -f <Funktion> -t <Zeitfenster>
```

Parameter im Einzelnen:

`-u` Kamera URL komplett mit http(s).
`-n` Projektname. Freier Name für das Projekt.
`-b` Bilderanzahl für das Gesamtprojekt.
`-i` Intervall für die Pausenzeiten zwischen den Bilderdownloads.
`-e` E-Mail für die Benachrichtigung (optional).
`-f` Funktion/Ausführung - 0 oder 1 (optional).
`-t` Zeitfenster im Format 8-18 (optional).
`-T` Zeitzone, hier nachlesen (optional).

```
./webcamloader.sh quicky -u "http://192.168.7.199/cgi-bin/api.cgi?cmd=Snap&channel=0&rs=0&user=user1&password=passwort" -b 15000 -p 15 -n "Vordach" -e "name@domain.tld" -f 0 -t 7-20
```

```
./webcamloader.sh quicky -u "http://192.168.7.199/mjpg/video.mjpg" -b 15000 -p 15 -n "Vordach" -e "name@domain.tld" -f 0 -t 7-20
... -t 7-20 -T ch
... -t 7-20 -T usa-nc
```

Hier sieht man demonstrativ und beispielhaft Varianten zur Eingabe im schnellen Quicky-Modus. Hier übergibt man alles, was wichtig ist, einmal dem Script und es kann dann nach einer Bestätigung die Arbeit beginnen. Beide Varianten haben als Beispiel 15.000 Bilder mit einem Intervall von 15 Sekunden.
Zeile 1 zeigt eine Reolink-Kamera mit Passwortschutz (via URL) und Zeile 2 zeigt einen MJPEG-Videostream ohne Passwortschutz.
Zeile 3 und 4 zeigt jeweils eine Zeitzone, die hinten an den ganzen Webcamloader-Aufruf angehängt wird.
Zeile 3 "ch" (stellvertretend für Europe/Zurich) und 4 "usa-nc" (North Carolina, stellvertretend für America/New_York).

# □□ Cron-Modus □

Der Cron-Modus ist ein Einmalaufruf, bei dem genau ein Bild gespeichert wird. Dieser ist gedacht für Langzeitaufnahmen mit Bild einmal am Tag per Crontab. Dabei können auch Folgebilder gesetzt werden. Dies ist sinnvoll, wenn der Cron-Befehl um 12:00 Uhr losgeht aber man nochmal ein Bild um 16:00 Uhr haben möchte. Klar könnte man den Crontab als *12,16* kennzeichnen - man könnte hier auch 2 Bilder definieren im Abstand von 4 Stunden. Ist Geschmackssache.

Aufruf:

```
./webcamloader.sh cron -u <URL> -n <Name> -b <Anzahl> -i <Intervall>
```

Parameter im Einzelnen:
`-u` Kamera URL komplett mit http(s).
`-n` Projektname. Freier Name für das Projekt.
`-b` Folgebilder (optional).
`-i` Intervall (optional).

```
./webcamloader.sh cron -u http://192.168.7.199/cgi-bin/faststream.jpg -n "Testkamera"
./webcamloader.sh cron -u http://192.168.7.199/cgi-bin/faststream.jpg -n "Testkamera" -b 2 -i 4h
```

Der Cron-Aufruf wie hier beispielhaft zu sehen, ist dasselbe Thema wie bei Quicky, nur einmalig ein Bild. Die Daten von Quicky sind hier auch parallel anzuwenden. In Zeile 1 sieht man einen

normalen, einmaligen Aufruf wie es standard- & zweckmäßig beim Script mit Cron erwartet wird, bei Zeile 2 sieht man einen Folgeauftrag, der dann 2 Bilder im Abstand von 4 Stunden macht. Hier wird als Beispiel um 12:00 Uhr ein Cronjob gestartet mit der Ausführung um ein Bild. Dann erfolgt nach 4 Stunden das Bild 2.
Hier könnte man auch klassisch einen Cronjob mit einer Einmalausführung mit *00 12,16 * * ** anlegen, hier passiert genau dasselbe.

# □□ Bildherstellung  Zusammenfassung □□

Die Zusammenfassung ist bei den Varianten vom geführten Modus und des Quickys gleich. Hier eine Übersicht der Zusammenfassung, die in den Fällen bestätigt werden muss.

Hier wieder beispielhafte Daten. Zum Thema Berechnung, zum Zeitpunkt dieses Bildes ist der 25.07.25, 21:45. Mit 24/7 Aufnahme und Intervall von 25 Sekunden mit 15.000 Bildern dauert die Bildspeicherung also ca. 4,5 Tage.

# □□ Status laufender Projekte

Unter diesem Menüpunkt lässt sich der aktuelle Status aller laufenden Projekte einsehen. Die Übersicht ist besonders hilfreich zur Kontrolle – etwa wenn ein Projekt aufgrund von

Netzwerkproblemen nur selten neue Bilder liefert.

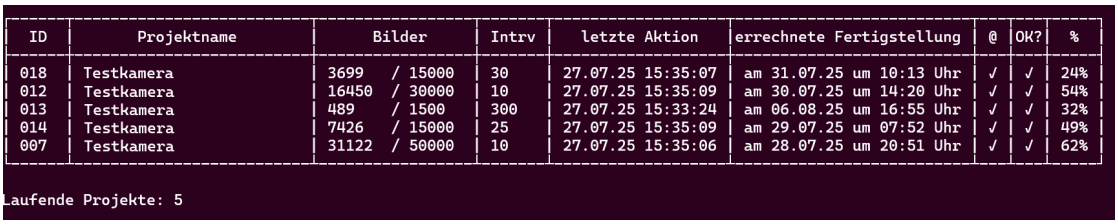
Der Status wird visuell durch ein ✔ (alles in Ordnung) oder ein ✖ (Problem erkannt) angezeigt. Zusätzlich ist ersichtlich, ob beim Start des Projekts eine Benachrichtigungs-E-Mail aktiviert wurde.

**Wichtig:**
Projekte, die per Cron gestartet wurden, gelten technisch als immer abgeschlossen und erscheinen **nicht** in dieser Übersicht.

Die Tabelle enthält folgende Informationen:

- Projektnummer (ID)
- Projektname
- Bilder (Bildanzahl IST / SOLL)
- Intervall
- letzte Aktion (Download oder Fehler)
- errechnete Fertigstellung
- Benachrichtigung aktiv? (als ✖ oder ✔)
- Projekt OK? (als ✖ oder ✔)
- Fertigstellung in %

Außerdem werden die Projekte aufgezählt.



| ID | Projektname | Bilder | Intrv | letzte Aktion | errechnete Fertigstellung | @ | OK? | % |
|---|---|---|---|---|---|---|---|---|
| 018 | Testkamera | 3699 / 15000 | 30 | 27.07.25 15:35:07 | am 31.07.25 um 10:13 Uhr | ✓ | ✓ | 24% |
| 012 | Testkamera | 16450 / 30000 | 10 | 27.07.25 15:35:09 | am 30.07.25 um 14:20 Uhr | ✓ | ✓ | 54% |
| 013 | Testkamera | 489 / 1500 | 300 | 27.07.25 15:33:24 | am 06.08.25 um 16:55 Uhr | ✓ | ✓ | 32% |
| 014 | Testkamera | 7426 / 15000 | 25 | 27.07.25 15:35:09 | am 29.07.25 um 07:52 Uhr | ✓ | ✓ | 49% |
| 007 | Testkamera | 31122 / 50000 | 10 | 27.07.25 15:35:06 | am 28.07.25 um 20:51 Uhr | ✓ | ✓ | 62% |

Laufende Projekte: 5

# □□ Fertige Projekte

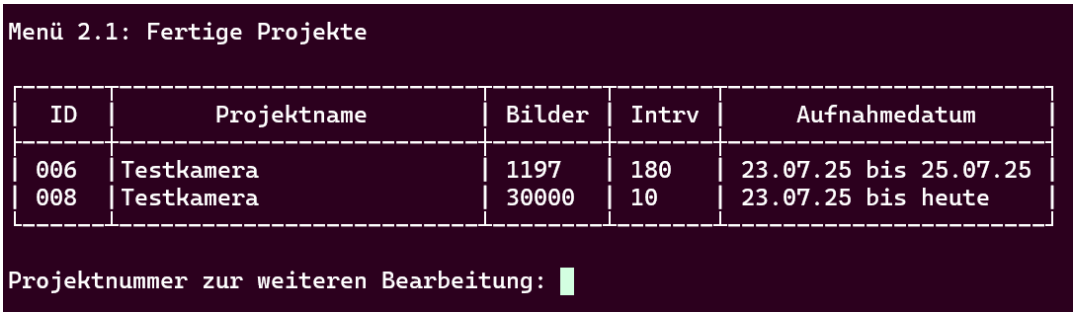
Sobald ein Projekt abgeschlossen ist – entweder durch Erreichen der definierten Bildanzahl oder durch manuellen Abbruch bei aktiviertem unbegrenztem Download – wird es unter dem Menüpunkt **Fertige Projekte** aufgelistet.

Hier werden alle beendeten Projekte zentral gesammelt. Dieser Schritt ist zwingend notwendig für die spätere Videoerstellung - an diesem Menü führt kein Weg vorbei.

**Wichtig:**
Projekte, die per Cron gestartet wurden, gelten technisch als immer abgeschlossen und erscheinen **grundsätzlich** in dieser Übersicht. Sie gelten immer als fertig und können zu jeder Zeit ein weiterverarbeitet werden.

Die Tabelle enthält folgende Informationen:

- Projektnummer (ID)
- Projektname
- Bildanzahl
- Intervall (nur bei manuell gestarteten Projekten sichtbar, also nicht via Cron)
- Aufnahmedatum (als Zeitraum *von–bis*, als *einzelner Tag* oder bis *heute*).

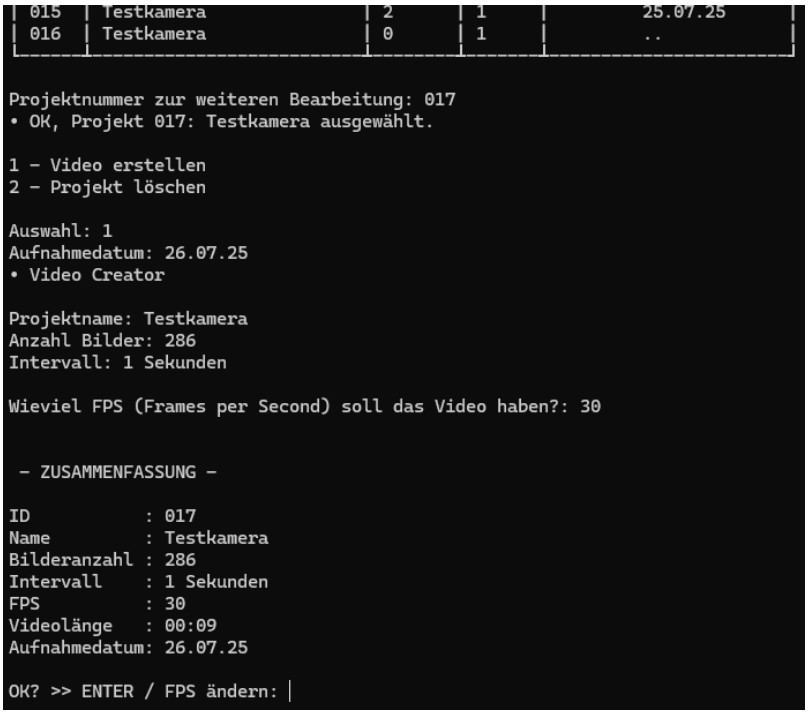



# Projekt abbrechen

Coming soon!

# Videoerstellung

Im oben erwähnten Menü zur Bilderstellung ist die Videoerstellung ebenfalls ein Punkt davon. Darüber lassen sich mit fertigen, abgeschlossenen Projekten ein Video erstellen. Im geführten Ablauf wird zunächst die gewünschte Bildrate (FPS – Frames per Second) abgefragt. Danach folgt eine Zusammenfassung aller relevanten Angaben zur Kontrolle. Da FPS und Bildanzahl maßgeblich die Videolänge beeinflussen, sollte hier sorgfältig geprüft werden. Dieses landet dann ebenfalls im Medienverzeichnis, parallel zu den Bildern. Ist das Video fertig, kann man das Projekt abschließen oder die FPS erneut mit anderem Wert setzen, sollte das Ergebnis des Videos nicht befriedigend sein. Wird das Projekt abgeschlossen, wird das rohe Projekt, also die Bilder als solches, gelöscht. Das Video bleibt selbstverständlich erhalten.




Hier zu sehen ist das Menü für die Videoerstellung bis hin zur □□ Zusammenfassung.

# □ Zeitzonen

Das Script kann mit frei wählbaren Zeitzonen arbeiten. Die eingestellte Zeitzone bestimmt, **in welcher lokalen Zeit** das konfigurierte Zeitfenster ausgewertet wird.

**Beispiel:**
Ist die Zeitzone auf *New York* gesetzt und das Zeitfenster auf **9-16 Uhr**, dann arbeitet das Script zu diesen Zeiten **in New Yorker Ortszeit**. Entsprechend entspricht dies in Deutschland (MEZ/MESZ) etwa **15:00–22:00 Uhr**.

Auf diese Weise können Webcams weltweit zeitgesteuert betrieben werden, ohne Aufnahmen während der Nacht zu erzeugen.

Wird **keine Zeitzone** angegeben, verwendet das Script automatisch die **Systemzeitzone** des Hosts.

Folgene Zeitzonen sind verfügbar, bzw. eingearbeitet worden. Für Vollständigkeit möchte ich nicht garantieren, aber es können Ausweichzeiten wie UTC+X genommen werden, falls wirklich.

Aufgrund der Masse habe ich die Zeitzonen ausgelagert. Zu finden hier: Link (intern). Bitte exakt so verwenden, wie sie in der linken Spalte geschrieben sind.

# ⚠ Haftungsausschluss (Disclaimer)

## Allgemein

Dieses Skript ist zur Ausführung **eigener Kamerasysteme** konzipiert. Fremdquellen wie Wetterkameras, Baustellenkameras etc. dürfen nur nach ausdrücklicher Zustimmung des Rechteinhabers genutzt werden.

**Hinweis zum Datenschutz:**
Die Erfassung und Speicherung von Kamerabildern unterliegt der **DSGVO** und ggf. weiteren gesetzlichen Regelungen.
Insbesondere bei **Personenerkennbarkeit** ist eine vorherige Klärung und ggf. Meldung verpflichtend.
Nutze dieses Tool **verantwortungsvoll** und **transparent** - ausschließlich im Rahmen geltender Gesetze.
Oder mit anderen Worten: Die **Verantwortung** für die rechtmäßige Nutzung **liegt ausschließlich beim Anwender**.

Ich übernehme **keine Haftung** für:

- missbräuchliche Verwendung
- technische Schäden
- Datenschutzverletzungen jeglicher Art

## Öffentliche Kameras

Viele Webcams von Gemeinden, Tourismusportalen, Stauseen oder Skigebieten dürfen ausdrücklich genutzt oder eingebunden werden. Bitte unbedingt bei der betreibenden Gesellschaft der Kamera selbst informieren.

# □□ Script Download

Das Programm kann hier in immer der neuesten Version heruntergeladen werden:

## **Webcamloader Script** (via mariobeh.de)

## Webcamloader auf Github

Ich bitte um Verständnis, wenn ich das Programm selbst vertreiben möchte. Das Script gibt es auch auf Github aber auf dem Server hier ist die Version stets aktuell.

---

*Nur in Deutsch verfügbar, Umbau auf anderen Sprachen auf Anfrage.*

Vielen Dank,
mariobeh.

# Faultnotify □□

> **Faultnotify, Debian (+Derivate)**
> ***Wiki-Stand: 10.04.2026***
> ***Script-Stand: 10.04.2026***
>
> *Sprachdateien-Referenz: 24.01.2026 (mehr...)*

***Download***

---

# □□ Vorwort

Faultnotify ist ein leichtgewichtiges, modular aufgebautes Bash-Script zur Überwachung von Geräten und Diensten.
Es wurde mit dem Ziel entwickelt, Störungen **zuverlässig und frühzeitig** zu erkennen. Dazu überprüft Faultnotify Geräte und Dienste fortlaufend und informiert im Fehlerfall automatisch per Benachrichtigung.

Alle benötigten Daten werden zentral in einem festen Verzeichnis abgelegt. Dort befinden sich unter anderem die Konfiguration, Statusinformationen sowie Steuer- und optional auch Sprachdateien. Dadurch bleibt alles übersichtlich an einem Ort.

Optional kann Faultnotify um zusätzliche Sprachdateien erweitert werden. In diesem Fall ist das Script in der Lage, Benachrichtigungen und Ausgaben in der jeweiligen Sprache bereitzustellen.

Faultnotify eignet sich sowohl für einzelne Systeme als auch für größere, verteilte Umgebungen. Es kommt bewusst ohne zusätzliche Software oder Datenbanken aus und bleibt dadurch einfach, robust und wartungsarm.

In verteilten Umgebungen können Geräte und Dienste zu Gruppen zusammengefasst werden. So wird sichergestellt, dass im Störungsfall nur gezielte und sinnvolle Benachrichtigungen ausgelöst werden.

Für den Dauerbetrieb empfiehlt sich die Einrichtung als Systemdienst. Dadurch läuft Faultnotify permanent im Hintergrund und bleibt auch nach Neustarts zuverlässig aktiv.

# □□ Allgemein und Installation

## Allgemein

Faultnotify arbeitet in einem zentralen Arbeitsverzeichnis unter:

/home/$USER/script-data/faultnotify/

In diesem Verzeichnis werden alle relevanten Dateien abgelegt, darunter die Konfiguration, Status- und Steuerdateien sowie Protokolle. Dadurch bleiben alle Informationen übersichtlich an einem Ort.

Optional kann Faultnotify um Sprachdateien erweitert werden. Diese werden im folgenden Verzeichnis abgelegt:

/home/$USER/script-data/faultnotify/lang/faultnotify-xxx.txt

Deutsch ist die im Script integrierte Standardsprache. Weitere Sprachen können über entsprechende Sprachdateien ergänzt werden. Sprachen können im Downloadbereich heruntergeladen werden.

## Installation und Ersteinrichtung

Faultnotify wird über den Befehl **install** eingerichtet. Das Script führt dabei interaktiv durch die Ersteinrichtung und legt alle grundlegenden Einstellungen an.

Während der Installation prüft Faultnotify, ob alle benötigten Programme auf dem System vorhanden sind. Fehlende Abhängigkeiten werden klar angezeigt und müssen anschließend manuell nachinstalliert werden.

Im nächsten Schritt wird der gewünschte Benachrichtigungsweg festgelegt. Zur Auswahl stehen **E-Mail** oder **Telegram**. Je nach Auswahl werden die erforderlichen Angaben abgefragt und in der Konfiguration gespeichert.

Zum Abschluss der Einrichtung wird ein Verifizierungscode an die angegebene E-Mail-Adresse oder an den ausgewählten Telegram-Account gesendet. Dieser Code muss eingegeben werden, um sicherzustellen, dass Benachrichtigungen im Störungsfall korrekt zugestellt werden.

Wird die Verifizierung übersprungen oder schlägt sie fehl, wird die Installation dennoch fortgesetzt. Die Benachrichtigungseinstellungen können später manuell angepasst werden, jedoch ohne erneute automatische Verifizierung. In diesem Fall liegt die vollständige Verantwortung für eine funktionierende Zustellung beim Nutzer. Ein entsprechender Workaround ist separat beschrieben.

## Gruppenlogik und Abhängigkeiten

Die Gruppenlogik wurde eingeführt, um Benachrichtigungsfluten gezielt zu vermeiden. Ein typisches Anwendungsbeispiel ist die Überwachung einer VPN-Verbindung mit mehreren dahinterliegenden Geräten oder Diensten.

Fällt in einem solchen Szenario die VPN-Verbindung aus, sind alle nachgelagerten Geräte ebenfalls nicht erreichbar. Ohne Gruppenlogik würde dies zu einer Vielzahl gleichzeitiger Störungsmeldungen führen, obwohl die eigentliche Ursache lediglich der VPN-Ausfall ist.

Durch die Gruppenbildung prüft Faultnotify zuerst den übergeordneten Eintrag. Ist dieser nicht erreichbar, werden die Prüfungen der abhängigen Geräte ausgesetzt und es erfolgen keine weiteren Benachrichtigungen für diese Einträge.

Erst wenn der übergeordnete Dienst wieder verfügbar ist, werden die abhängigen Geräte erneut geprüft. Dadurch bleiben Benachrichtigungen übersichtlich, aussagekräftig und auf die tatsächliche Ursache beschränkt.

# □□ Geräteprüfung (TEST)

Das Modul test ist die zentrale Prüffunktion von Faultnotify. Es prüft alle in der Konfiguration hinterlegten Geräte und Dienste und erkennt Störungen sowie Wiederherstellungen.

Bei direktem Aufruf führt test genau einen vollständigen Durchlauf aus. Das bedeutet:

- Alle Geräte und Dienste werden einmal geprüft.
- Im Fehlerfall werden Benachrichtigungen ausgelöst.
- Danach endet der Durchlauf automatisch.

# □□ Automation (RUN)

Dies ist der Dauerschleifen-Modus. Hier werden permanent mit einem Versatz von 5 Sekunden die in der Konfiguration eingerichteten Geräte und Dienste geprüft.

**Hinweis:** Wird `run` direkt im Terminal gestartet, läuft Faultnotify im Vordergrund und blockiert das Terminal. Für den Dauerbetrieb wird die Einrichtung als Systemdienst empfohlen. Dadurch läuft

Faultnotify dauerhaft im Hintergrund und der Status ist jederzeit über `systemctl status faultnotify` sichtbar.

# Geräte/Dienste hinzufügen (ADD)

Mit dem Modul **add** können neue Geräte und Dienste interaktiv zur Überwachung hinzugefügt werden. Der Vorgang ist dialoggeführt aufgebaut und kann mehrfach hintereinander ausgeführt werden, um mehrere Einträge nacheinander anzulegen.

Zu Beginn ermittelt Faultnotify automatisch die nächste freie Geräte-ID. Diese wird fortlaufend vergeben.

Im Anschluss werden Schritt für Schritt folgende Angaben abgefragt:

- **Name des Geräts oder Dienstes**
  Der Anzeigename dient ausschließlich der Übersicht. Kritische Zeichen werden automatisch bereinigt, um die Konfigurationsdatei konsistent zu halten.
- **IP-Adresse oder Domain**
  Es kann entweder eine IPv4-Adresse oder ein gültiger Domainname angegeben werden.
- **Dienstüberwachung (optional)**
  Falls ein Dienst überwacht werden soll, wird der zu prüfende Port abgefragt.
- **Abhängigkeit von einem Master-Gerät (optional)**
  Geräte und Dienste können einem bestehenden Eintrag untergeordnet werden. Dadurch lassen sich Abhängigkeiten abbilden, zum Beispiel:
  - Dienste hängen von einem Server ab
  - Untergeräte hängen von einem Hauptgerät ab

  Ist kein Master angegeben, wird der Eintrag automatisch der Root-Gruppe (G000) zugeordnet.

Nach Abschluss der Eingaben wird der neue Eintrag:

- in die Konfigurationsdatei geschrieben
- im internen Log protokolliert
- sofort für künftige Prüfungen berücksichtigt

Der Hinzufügen-Modus kann jederzeit durch eine leere Eingabe beim Namen beendet werden. Nach dem ersten erfolgreichen Hinzufügen wird Faultnotify als „eingerichtet“ markiert, sodass die Überwachung genutzt werden kann.

**Hinweis zur Gruppenlogik:**

Die Gruppenlogik entstand aus der praktischen Anforderung heraus, Benachrichtigungsfluten zu vermeiden. Ein typisches Beispiel ist die Überwachung einer VPN-Verbindung mit mehreren dahinterliegenden Geräten oder Diensten.

Fällt in einem solchen Szenario die VPN-Verbindung aus, wären die nachgelagerten Geräte zwangsläufig ebenfalls nicht erreichbar. Ohne Gruppierung würde dies zu einer Vielzahl gleichzeitiger Störungsmeldungen führen, obwohl die eigentliche Ursache nur der VPN-Ausfall ist.

Durch die Gruppenbildung wird dieses Problem gezielt verhindert. Wird ein Gerät oder Dienst als abhängig von der VPN-Überwachung angelegt, prüft Faultnotify zuerst den übergeordneten Eintrag. Ist dieser nicht erreichbar, werden die untergeordneten Prüfungen ausgesetzt und es erfolgt keine weitere Benachrichtigung für die abhängigen Geräte.

Erst wenn die VPN-Verbindung wieder verfügbar ist, werden die nachgelagerten Geräte erneut geprüft. Auf diese Weise bleibt die Benachrichtigung aussagekräftig, übersichtlich und auf die tatsächliche Ursache beschränkt.

# ✏ Modifizierung/Bearbeitung (MOD)

Das Modul **mod** dient zur nachträglichen Pflege und Anpassung von Faultnotify. Es ermöglicht sowohl die Verwaltung einzelner Geräte und Dienste als auch die Änderung zentraler Konfigurationseinstellungen.

Nach dem Start von mod wird zunächst abgefragt, **welcher Bereich geändert werden soll**:

- **Geräte und Dienste**
- **Zentrale Konfiguration**

**Geräte und Dienste bearbeiten**

Wird der Geräte-/Service-Bereich gewählt, zeigt Faultnotify zunächst **alle aktuell eingerichteten Einträge** in einer übersichtlichen Liste an. Jeder Eintrag enthält unter anderem:

- Geräte- bzw. Service-ID
- Gruppenzugehörigkeit
- Name
- IP-Adresse
- optional Port und Protokoll

Anschließend wird abgefragt, **welcher Eintrag geändert werden soll**. Die Auswahl kann flexibel erfolgen:

- über die Geräte-ID (z. B. `D006` oder `006`)
- über einen Teil des Namens
- über die vollständige IP-Adresse

Nach eindeutiger Zuordnung des Eintrags stehen zwei Optionen zur Verfügung:

1. **Eintrag löschen und neu definieren**
   Der bestehende Eintrag wird entfernt, zugehörige Status- und Jail-Informationen werden bereinigt und anschließend wird automatisch der add-Dialog gestartet, um das Gerät oder den Dienst neu anzulegen.
2. **Eintrag ersatzlos löschen**
   Der Eintrag wird vollständig aus der Konfiguration entfernt. Auch hier werden zugehörige Status- und Jail-Dateien bereinigt.

Damit lassen sich Geräte und Dienste sauber korrigieren, ersetzen oder dauerhaft entfernen.

**Konfiguration bearbeiten**

Wird die Konfiguration gewählt, zeigt Faultnotify zunächst **alle relevanten aktuellen Einstellungen** an, inklusive erklärender Hinweise. Dazu gehören unter anderem:

- Online-Ping-Prüfung (Referenz für Internet-Erreichbarkeit)
- Art der Benachrichtigung (E-Mail oder Telegram)
- E-Mail-Empfänger
- Telegram-Bot-Token
- Telegram-Chat-ID
- Nächste zu vergebende Geräte-ID

Anschließend kann gezielt ein einzelner Parameter geändert werden. Jede Änderung erfolgt interaktiv:

- Neuer Wert eingeben
- Übersicht anzeigen
- explizite Bestätigung vor dem Übernehmen

Eingaben werden dabei geprüft (z. B. Format von IP-Adressen, Domains, E-Mail-Adressen oder IDs), um fehlerhafte Konfigurationen zu vermeiden. Kommentare in der Konfigurationsdatei bleiben erhalten.

**Ziel des MOD-Moduls**

Das mod-Modul stellt sicher, dass Faultnotify auch im laufenden Betrieb **einfach, kontrolliert und konsistent gepflegt** werden kann - ohne manuelle Eingriffe in die Konfigurationsdatei und ohne das Risiko inkonsistenter Zustände.

# □□ Übersicht (DRAW)

Mit dem Modul draw kann die komplette Geräte- und Dienststruktur übersichtlich als Baum angezeigt werden. Dabei werden alle Abhängigkeiten berücksichtigt, sodass auf einen Blick sichtbar ist, welche Geräte oder Dienste untergeordnet sind und welche als übergeordnete „Master“-Ebene dienen.

Die Darstellung beginnt bei ROOT (G000) und zeigt darunter alle Einträge aus der Konfiguration in einer klaren Hierarchie. Zusätzlich werden die wichtigsten Informationen direkt in der Ausgabe mitgeführt:

- ID des Eintrags
- Name bzw. Beschreibung
- IP-Adresse
- optional Port

Fehlerhinweis bei ungültigen Abhängigkeiten
Falls ein Eintrag auf ein nicht vorhandenes Master-Gerät verweist, gibt draw einen Hinweis aus und listet die betroffenen Einträge auf. Dadurch lassen sich fehlerhafte Abhängigkeiten schnell erkennen und anschließend über mod korrigieren.

```
./faultnotify.sh draw
[Faultnotify Hierarchie]
ROOT
├─ 001  Server 1                                  | 192.168.1.77
├─ 002  VPN Geschäftsstelle 4                     | 10.1.200.10
│   ├─ 003  Server Gechäftsstelle 4               | 10.2.1.1
│   ├─ 004  Plotter                               | 10.2.1.13
│   └─ 005  Terminalserver                        | 10.2.1.100 :3389
├─ 006  VPN Geschäftsstelle 5                     | 10.10.200.10
│   └─ 007  Proxmox 5                             | 10.10.200.20
│       ├─ 008  Webserver                         | 10.10.201.10
│       ├─ 009  Dockerserver                      | 10.10.201.20
│       │   ├─ 011  NPM 80                        | 10.10.201.40 :80
│       │   ├─ 012  NPM 81                        | 10.10.201.40 :81
│       │   └─ 013  Guacamole                     | 10.10.201.40 :10000
│       ├─ 010  Mailserver                        | 10.10.201.30
│       └─ 014  Admin-Terminal                    | 10.10.202.20
└─ 015  Medienserver                              | 192.168.1.78
```

Hier zu sehen: DRAW-Auflistung mit Beispielwerten.

# □□ Fehler / Troubleshooting

Wird während der Installation eine Eingabe wie die Benachrichtigung falsch getägigt oder die Verifizierung der Benachrichtigung übersprungen oder schlägt fehl, wird die Installation trotzdem weitergeführt. Die Verifizierung dient dazu, dass sichergestellt werden kann, dass im Störungsfall auch Benachrichtigungen ankommen können.
In diesem Fall kann im Verzeichnis `script-data/faultnotify` die Datei `.installed` gelöscht werden, um die Installation erneut zu starten und die Erstkonfiguration nochmals vollständig durchzuführen.

# □□ Sprachdateien

Faultnotify ist in der Lage, in verschiedenen Sprachen zu arbeiten und Benachrichtigungen entsprechend auszugeben. Grundlage dafür sind externe Sprachdateien. Im Script wird die **I18N-Übersetzungstechnologie** eingesetzt. Diese ermöglicht eine klare Trennung zwischen Programmlogik und Textausgaben und bildet die Grundlage für die mehrsprachige Ausgabe von Meldungen und Benachrichtigungen.

Deutsch ist die im Script fest integrierte Standardsprache. Zusätzlich steht eine englische Sprachdatei zum Download bereit.

Weitere Sprachen können jederzeit ergänzt werden. Die Sprachdateien sind bewusst einfach aufgebaut, sodass sie von jedermann erstellt oder erweitert werden können. Dadurch ist Faultnotify nicht auf eine feste Anzahl von Sprachen beschränkt und lässt sich flexibel an unterschiedliche Umgebungen anpassen.

Die Sprachdatei muss dann im zentralen Pfad im Ordner *lang/* eingepflegt werden. Liegen mehrere Sprachdateien im Ordner, wird die erste vom Script automatisch ausgewählt. Es ist daher ratsam, nur die benötigte Sprache herunterzuladen.

Faultnotify verwendet zur Einbindung von Sprachdateien einen SHA-256-Referenzcode, der direkt aus dem Script abgeleitet wird. Dieser Referenzcode stellt sicher, dass eine Sprachdatei eindeutig zu genau dieser Script-Version gehört. Diese Referenz ist in der ersten Zeile zu finden. Wenn eine neue Sprache integriert wird, ist dieser Referenzcode einzusetzen.

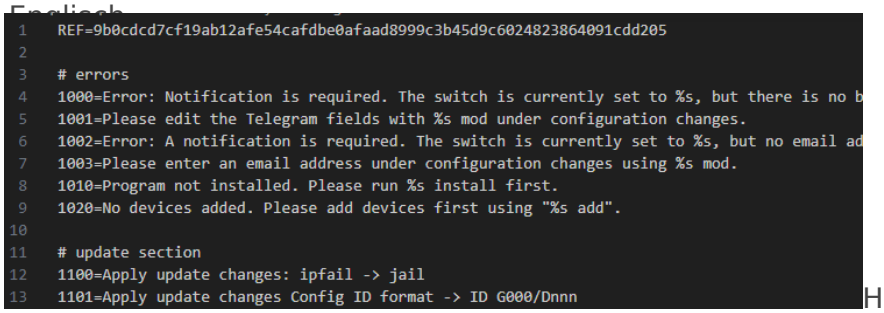
Dadurch wird verhindert, dass versehentlich eine unpassende oder fremde Sprachdatei verwendet wird, zum Beispiel aus einem anderen Script oder einer älteren Version. Stimmen Script und Sprachdatei nicht überein, wird die Sprachdatei nicht geladen.

Der zugehörige Referenzcode lautet:

```
9b0cdcd7cf19ab12afe54cafdbe0afaad8999c3b45d9c6024823864091cdd205
```

Bitte beachten, dass sich dieser Referenzcode mit einer neuen Version ändern kann. Hintergrund ist die Integrität zur neuen Version, falls neue Textabschnitte hinzukommen oder welche abgeändert werden.

Im Falle eines Updates wird darauf hingewiesen mit Link in der Script-Standardsprache (Deutsch) und Englisch.



Hier zu sehen: erste Zeilen der englischen Sprachdatei.

# □□ Download

Das **Programm** kann hier in immer der neuesten Version heruntergeladen werden:

## **Faultnotify Script** (via mariobeh.de)

```
curl -s https://public.mariobeh.de/scripts/faultnotify.sh | bash
```

**Sprachdateien:**

**Englisch** (via mariobeh.de)

```
USER=$(whoami) && mkdir -p /home/$USER/script-data/faultnotify/lang/ && curl -o /home/$USER/script-data/faultnotify/lang/faultnotify-eng.txt https://public.mariobeh.de/scripts/faultnotify-eng.txt && curl -s https://public.mariobeh.de/scripts/faultnotify.sh | bash
```

*(um gleich in Englisch zu starten, ist dieser Befehl erfoderlich. Er erstellt die Verzeichnisstruktur und legt die Sprachdatei an passender Stelle ab. Dieser Befehl lädt nicht nur die Sprachdatei, sondern startet direkt das Script.)*

**Hinweis:** Deutsch ist die im Script fest integrierte Standardsprache und benötigt keine separate Sprachdatei. Daher wird Deutsch hier nicht separat angeboten. Weitere Sprachen werden folgen.

# □□ Update

Über ein externes Updater-Script kann Faultnotify immer geupdated werden. Link hier: Updater-Script (COMING SOON!)

Für die Sprachdatei gilt ein direktes Verfahren, einfach den Link einfügen, Sprachdatei wird automatisch an Ort und Stelle geladen. Alternativ kann oben die Datei manuell ins System eingepflegt werden.

```
USER=$(whoami) && mkdir -p /home/$USER/script-data/faultnotify/lang/ && curl -o /home/$USER/script-data/faultnotify/lang/faultnotify-eng.txt https://public.mariobeh.de/scripts/faultnotify-eng.txt
```

# IPlogger □□

> **IPlogger, Debian (+Derivate)**
> ***Wiki-Stand: 12.05.2026***
> ***Script-Stand: 12.05.2026***
>
> ***Download***

# IPLogger DB – IP- und Verfügbarkeitsüberwachung

## Überblick

Der `iplogger-db.sh` ist ein datenbankgestützter IP-Logger zur dauerhaften Überwachung der eigenen Internetverbindung und der öffentlichen WAN-IP-Adresse.
Das Script wurde dafür entwickelt, langfristig und nachvollziehbar festzuhalten:

- welche öffentliche IP-Adresse zu welchem Zeitpunkt aktiv war
- ob eine Internetverbindung vorhanden war
- ob lediglich die IP-Ermittlung fehlgeschlagen ist
- welche Prüfdienste funktioniert oder versagt haben
- wie lange einzelne IPs aktiv waren
- wann Provider oder Verbindungen ausgefallen sind

Der Fokus liegt nicht auf einer simplen „Wie ist meine IP“-Abfrage, sondern auf einer historisch nachvollziehbaren Langzeitaufzeichnung mit Auswertung und Zeiträumen.

Das System arbeitet vollständig datenbankbasiert und speichert jede einzelne Messung dauerhaft ab.

---

# Ziel des Systems

Der Logger beantwortet unter anderem folgende Fragen:

- Welche öffentliche IP war am 12.03.2024 um 14:00 Uhr aktiv?
- Wie lange blieb eine bestimmte IP bestehen?
- Wann erfolgte ein DSL-/WAN-Reconnect?
- War die Leitung tatsächlich offline oder nur ein Prüfdienst gestört?
- Welche Provider lieferten fehlerhafte Antworten?
- Welche DNS-/Connectivity-Ziele waren erreichbar?
- Wie oft kam es zu Unterbrechungen?
- Wie stabil arbeitet die Internetanbindung?

Gerade bei:

- DSL-Zwangstrennungen
- CGNAT-/Providerwechseln
- VPN-/Tunnelproblemen
- Routingfehlern
- sporadischen Ausfällen
- Providerstörungen

liefert die Historie eine klare technische Nachvollziehbarkeit.

---

# Architektur

Das System besteht aus mehreren Komponenten:

| Komponente | Aufgabe |
|---|---|
| `providers` | Öffentliche IP-Abfragedienste |
| `connectivity_checks` | Prüft, ob Internet grundsätzlich erreichbar ist |
| `state` | Speichert Rotationszustände |
| `iplog` | Historische Hauptdatenbank |
| `summary_*` | Auswertung und Zusammenfassung |
| PDF-/CSV-Export | Archivierung und Dokumentation |

---

# Grundlogik

Der Ablauf einer Messung sieht vereinfacht so aus:

1. Connectivity prüfen
2. Wenn Internet vorhanden:
3. Öffentliche IP ermitteln
4. Ergebnis speichern
5. Statistiken aktualisieren

Wichtig dabei:

Das Script unterscheidet sauber zwischen:

| Zustand | Bedeutung |
|---|---|
| `OFFLINE` | Keine Internetverbindung |
| `CHECK_FAILED` | Internet vorhanden, aber IP nicht ermittelbar |
| `OK` | IP erfolgreich ermittelt |
| `INVALID_RESPONSE` | Provider lieferte ungültige Antwort |
| `NO_RUN` | Reservierter Status |

Dadurch wird verhindert, dass ein einfacher Fehler eines einzelnen IP-Dienstes fälschlich als Internetausfall interpretiert wird.

---

# Initialisierung

# Einstiegspunkt: `init`

```
./iplogger-db.sh init
```

Dieser Befehl erstellt die komplette Datenbankstruktur.

Dabei werden automatisch angelegt:

- Tabellen
- Standardprovider
- Connectivity-Ziele
- Rotationsstatus

---

# Datenbanktabellen

Es muss vorab ein Benutzer und eine Datenbank erstellt sein. Die init-Datenbankinitialisierung erstellt dann die Tabellen.

# providers

Enthält alle IP-Ermittlungsdienste.

Standardmäßig:

- icanhazip.com
- ifconfig.me
- api.ipify.org
- checkip.amazonaws.com

Gespeichert werden zusätzlich:

- Erfolgszähler
- Fehlerzähler
- letzter Erfolg
- letzter Fehler
- Timeoutwerte

Beispiel:

| Name | URL |
| --- | --- |
| icanhazip.com | https://icanhazip.com |

| Name | URL |
| --- | --- |
| api.ipify.org | https://api.ipify.org |

---

# connectivity_checks

Prüft die generelle Internetverfügbarkeit.

Standardmäßig:

- Google DNS (`8.8.8.8`)
- Quad9 (`9.9.9.9`)
- Cloudflare (`1.1.1.1`)

Die Prüfung erfolgt per Ping.

---

# state

Speichert interne Rotationsinformationen.

Dadurch wird nicht immer derselbe Provider zuerst verwendet.

Beispiele:

| Name | Wert |
| --- | --- |
| next_provider_index | 2 |
| next_connectivity_index | 1 |

---

# iplog

Die zentrale Historientabelle.

Hier wird jede Messung dauerhaft gespeichert.

Gespeichert werden unter anderem:

- Messzeitpunkt
- öffentliche IP
- Status
- verwendeter Provider
- Anzahl Versuche
- Fehlermeldungen
- Rohantworten
- Connectivity-Status

---

# Rotationssystem

## Warum Rotation?

Würde immer derselbe Provider zuerst verwendet werden:

- wäre dieser stärker belastet
- Fehler würden schwerer auffallen
- andere Provider würden kaum genutzt

Deshalb arbeitet das System rotierend.

---

# Beispiel

## Durchlauf 1

1. icanhazip.com
2. ifconfig.me
3. ipify

## Durchlauf 2

1. ifconfig.me
2. ipify
3. icanhazip.com

## Durchlauf 3

1. ipify
2. icanhazip.com
3. ifconfig.me

Dadurch verteilt sich die Last gleichmäßig.

---

# Connectivity-Prüfung

## Einstiegspunkt: `check` oder `run`

```
./iplogger-db.sh check
```

oder:

```
./iplogger-db.sh run
```

Beides startet dieselbe Messlogik.

---

# Ablauf der Connectivity-Prüfung

Vor der eigentlichen IP-Ermittlung wird geprüft:

- besteht überhaupt Internet?
- ist Routing vorhanden?
- funktionieren externe Ziele?

Dazu werden mehrere bekannte Ziele angepingt.

Erst wenn mindestens eines erreichbar ist, wird die IP-Abfrage gestartet.

---

# Provider-Prüfung

Nach erfolgreicher Connectivity-Prüfung:

```
curl -> Provider -> Antwort validieren
```

Dabei wird geprüft:

- ist die Antwort leer?
- ist die Antwort eine gültige IPv4?
- ist die Antwort eine gültige IPv6?

Nur dann gilt der Durchlauf als erfolgreich.

---

# Beispielabläufe

# Erfolgreicher Lauf

Connectivity OK

Provider erreichbar

IP gültig

→ Status OK

Gespeichert wird beispielsweise:

2026-05-12 22:00:00

IP: 93.192.117.44

Status: OK

---

# Internet vorhanden, aber Provider kaputt

Connectivity OK

Provider liefert Müll

→ CHECK_FAILED

Beispiel:

Provider https://api.example liefert:

ERROR 500

Das Internet funktioniert trotzdem.

---

# Komplett offline

```
Google DNS nicht erreichbar
Cloudflare nicht erreichbar
Quad9 nicht erreichbar
→ OFFLINE
```

---

# Summary-System

Das eigentliche Kernstück ist die Verlaufsanalyse.

Einzelne Messungen werden zu Zeiträumen zusammengefasst.

---

# Einstiegspunkt: `summary`

# Tabellenansicht

```
./iplogger-db.sh summary tabelle
```

Beispiel:

```
BEGIN_AT         END_AT           IP              SAMPLES OFFLINE FAILS
2026-05-01 10:00  2026-05-03 02:00   93.192.117.44       91      0     0
```

# Textansicht

```
./iplogger-db.sh summary text
```

Beispiel:

```
01.05.2026 10:00 Uhr - 03.05.2026 02:00 Uhr:
IP 93.192.117.44 (91x online)
```

# Wie die Periodenerkennung funktioniert

Die Funktion `summary_rows()` analysiert die komplette Historie.

Sobald sich die IP ändert:

```
alte Periode schließen
neue Periode beginnen
```

Zusätzlich werden innerhalb der Periode gezählt:

- Offline-Ereignisse

- Fehler
- erfolgreiche Samples

---

# Detailmodus

# Einstiegspunkt

```
./iplogger-db.sh summary export pdf details
```

Der Detailmodus erzeugt eine vollständige Zeitauflistung aller erfolgreichen Messungen innerhalb einer Periode.

---

# Beispiel

```
16.02.2020 05:00 Uhr - 16.02.2020 11:00 Uhr:
IP 93.192.117.44 (24x online)

- 16.02.2020 05:00 06:00 07:00 08:00
- 16.02.2020 09:00 10:00 11:00
```

Oder bei Tageswechsel:

```
07.02.2021 04:00 05:00 06:00
08.02.2021 00:00 01:00 02:00
```

Dadurch werden lange Zeiträume kompakt lesbar dargestellt.

# PDF-Export

# Einstiegspunkt

```
./iplogger-db.sh summary export pdf
```

oder:

```
./iplogger-db.sh summary export pdf details
```

Der Export erzeugt automatisch:

- TXT-Zwischendatei
- PDF-Datei
- Überschrift
- Datum
- Seitenzahlen

Verwendet werden bevorzugt:

- `enscript`
- `ps2pdf`

Alternativ:

- `pandoc`

# CSV-Export

## Einstiegspunkt

```
./iplogger-db.sh summary export csv
```

Exportiert die zusammengefassten Perioden in maschinenlesbarer Form.

Beispiel:

```
begin_at;end_at;ip;samples;offline;fails
2026-05-01 10:00;2026-05-03 02:00;93.192.117.44;91;0;0
```

---

# Historischer CSV-Import

## Einstiegspunkt

```
./iplogger-db.sh import ipdb.csv ";"
```

Der CSV-Import dient primär zur Übernahme historischer Daten aus der ersten Generation des IPLoggers.

Dabei werden ältere Aufzeichnungen in die neue Datenbankstruktur übernommen.

Unterstützt werden:

- erfolgreiche IPs
- Offline-Einträge
- Fehlerzustände

---

# Typische Einsatzszenarien

## WAN-Reconnects nachvollziehen

IP 1:
01:00 - 12:00

IP 2:
12:05 - aktuell

→ DSL-Zwangstrennung klar sichtbar.

---

## Providerprobleme erkennen

15x CHECK_FAILED
aber keine OFFLINE-Zustände

→ Internet vorhanden, aber IP-Dienst gestört.

---

## VPN-/Tunnelprobleme analysieren

Wenn:

- Connectivity vorhanden
- aber bestimmte Ziele nicht erreichbar

kann nachvollzogen werden:

- wann Routingprobleme begannen
- wie lange sie bestanden
- ob WAN-Wechsel beteiligt waren

---

# Besonderheiten

## IPv4 und IPv6

Das System akzeptiert beide Varianten automatisch.

---

## Zeitbasierte Rotation

Provider und Connectivity-Checks werden nicht statisch verwendet.

Das verhindert:

- Single-Point-Abhängigkeiten
- einseitige Last
- verfälschte Statistiken

---

## Dauerhafte Historie

Die Datenbank arbeitet append-only.

Neue Einträge werden ausschließlich ergänzt.

Dadurch bleibt die komplette Historie nachvollziehbar erhalten.

---

# Beispiel für Cronjob

```
0 * * * * /srv/scripte/iplogger-db.sh check >/dev/null 2>&1
```

Dadurch erfolgt stündlich eine neue Messung.

---

# Fazit

`iplogger-db.sh` ist kein einfacher „Wie ist meine IP“-Checker, sondern ein vollständiges Langzeit-Überwachungs- und Auswertungssystem für:

- öffentliche WAN-IPs
- Internetverfügbarkeit
- Providerstabilität
- Fehleranalysen
- historische Zeiträume
- dokumentierbare Verbindungsverläufe

Besonders wertvoll wird das System durch:

- die periodische Zusammenfassung
- die saubere Statusunterscheidung
- die rotierenden Prüfmechanismen
- die nachvollziehbare Langzeithistorie
- die Exportfunktionen für Archivierung und Dokumentation

# □□ Download

Das **Programm** kann hier in immer der neuesten Version heruntergeladen werden:

**IPlogger Script** (via mariobeh.de)