BEH:Wiki

Linux Nachschlagwissen in dem Groß-Distributionsbereich Debian/Ubuntu mit eigenen Programmen und Hilfestellungen, die mir und Kollegen dienen. Wenn Fremde Leute sich hier Wissen abholen, ist dies ebenfalls im Sinne des Erfinders.

- Headless sourcing-fähige Snippets
  - mailer (Bash)
  - mailer (Python3)
  - fillortrim.sh (Bash)
  - fillortrim.sh (Python3)
  - dabch2hz.sh
  - fm2hz.sh
  - checkpackage.sh
  - picorvid.sh
  - port.sh
  - rdm.sh
  - rdmfreqfm.sh
  - rdsps.sh
  - runtxt.sh
  - tvch2hzdigeu.sh
- Proxmox: VM/CT gesperrt
- Archive unter Linux
  - zip
  - bz2, bzip2
  - gz, gzip
  - tar
- VPN-Server mit Wireguard / Side-to-Side-VPN / alles tunneln
- Zeitumstellung via timedatectl
- Systemdienst einrichten

- SSH Forwarding (Portweiterleitung via SSH)
- Root-Login verbieten
- PhpMyAdmin - Installation
- PhpMyAdmin - Fehlermeldung Konfigurationsspeicher
- Mount - HDD/SSD mounten
- Benutzer zum sudo (sudoers) hinzufügen
- Apt-get Autocomplete / Autocompletion
- eigene Programme, headless
    - IP Logger (Debian / 04.01.2024)
    - M4A zu MP3 Massenumwandlung (Debian / 09.04.2025)
    - DynDNS Updater (Debian / 10.07.2024)
    - Webcamloader □□
    - Faultnotify □□
    - IPlogger □□

- Telegram Benachrichtigungsdienst mit einer einzigen PHP
- Telegram Benachrichtigungsdienst mit einer einzigen Bash
- Proxmox - Virtuelle Festplatte vergrößern und in der VM nutzbar machen (LVM + ext4)
- webcamloader-timezones
- Webcamloader (□□Debian / 08.25)
- Versioner □□ (Git Alternative Lite)

Headless sourcing-fähige Snippets

Hier werden kleine Helferscripte vorgeführt, die in Scripten included oder per if-then abgefragt werden können.

mailer (Bash)

Dieser Mailer schickt einfach eine Mail vom System aus. Dabei muss darauf geachtet werden, dass der Server Mails versenden kann mit "mail". Es schickt je nach Einstellung eine Text-Mail oder Mail mit Anhang. Der Rückgabewert besagt, ob die Mail erfolgreich versendet wurde - oder nicht. Eigener Anwendungsfall: Einfacher Text, Bestätigungscode, Link, Nachricht.

HOWTO:

```
$0 -f "/var/file.png" -e mail@domain.tld -s "Betreff"
$0 -t "Text" -m mail@domain.tld -s "Betreff"
# -e ODER -m = Mailadresse
```

Beispiel:

```
# Direktaufruf (Tests)
mailer.sh -e "mail@domain.tld" -s "Testbetreff" -t "Hallo, ich bin ein Test!"

# im Script

empfaenger="mail@domain.tld"
betreff="Testbetreff"
nachricht="Hallo, ich bin ein Test!"

if bash lib/mailer.sh -e "$empfaenger" -s "$betreff" -t "$nachricht"; then
    echo "Mail erfolgreich gesendet"
else
    echo "Fehler beim Mailversand"
fi
```

Script:

```
#!/bin/bash

while [[ $# -gt 0 ]]; do
```

```
    case "$1" in
        -f) file="$2"; shift 2 ;;
        -t) text="$2"; shift 2 ;;
        -e|-m) mail="$2"; shift 2 ;;
        -s) subj="$2"; shift 2 ;;
    esac
done

if [ -z "$mail" ] || [ -z "$subj" ]; then
    echo "ERROR"
    exit 1
fi

if [ -n "$file" ] && [ -z "$text" ]; then
    cat "$file" | mail -s "$subj" "$mail"

elif [ -n "$text" ] && [ -z "$file" ]; then
    echo "$text" | mail -s "$subj" "$mail"

else
    echo "ERROR"
    exit 1
fi

exit 0
```

mailer (Python3)

Dieser Mailer schickt einfach eine Mail vom System aus. Dabei muss darauf geachtet werden, dass der Server Mails versenden kann mit "mail". Es schickt je nach Einstellung eine Text-Mail oder Mail mit Anhang. Der Rückgabewert besagt, ob die Mail erfolgreich versendet wurde - oder nicht. Eigener Anwendungsfall: Einfacher Text, Bestätigungscode, Link, Nachricht.

HOWTO:

```
python3 $0 -f "/var/file.png" -e mail@domain.tld -s "Betreff"
python3 $0 -t "Text" -m mail@domain.tld -s "Betreff"
# -e ODER -m = Mailadresse
```

Beispiel:

```
import subprocess

empfaenger = "mail@domain.tld"
betreff = "Testbetreff"
nachricht = "Hallo, ich bin ein Test!"

try:
    result = subprocess.run(
        ['python3', 'lib/mailer.py', '-e', empfaenger, '-s', betreff, '-t', nachricht],
        check=True
    )
    print("Mail erfolgreich gesendet")
except subprocess.CalledProcessError:
    print("Fehler beim Mailversand")
```

Script:

```
#!/usr/bin/env python3

import sys
```

```
import subprocess

args = sys.argv[1:]

file = None
text = None
mail = None
subj = None

# Argument-Parsing
i = 0
while i < len(args):
    if args[i] == '-f':
        file = args[i + 1]
        i += 2
    elif args[i] == '-t':
        text = args[i + 1]
        i += 2
    elif args[i] in ['-e', '-m']:
        mail = args[i + 1]
        i += 2
    elif args[i] == '-s':
        subj = args[i + 1]
        i += 2
    else:
        i += 1

# Fehlerprüfung
if not mail or not subj:
    print("ERROR")
    sys.exit(1)

# Mailversand
if file and not text:
    try:
        subprocess.run(['mail', '-s', subj, mail], input=open(file, 'rb').read(), check=True)
    except Exception as e:
        print("Fehler beim Senden:", e)
        sys.exit(1)
```

```
elif text and not file:
    try:
        subprocess.run(['mail', '-s', subj, mail], input=text.encode(), check=True)
    except Exception as e:
        print("Fehler beim Senden:", e)
        sys.exit(1)

else:
    print("ERROR")
    sys.exit(1)
```

Headless sourcing-fähige Snippets

fillortrim.sh (Bash)

Auffüllen oder trimmen von einem Text auf X Zeichen. Es erzeugt kein typischer Rückgabewert, sondern gibt den Text bearbeitet aus.
Klassischer Anwendungsfall: textbasierte Tabelle, bei der ein Text immer die gleiche Länge haben soll, um eine sinnvolle, visuelle Tabelle zu bilden.

HOWTO:

```
$0 $var $length
$0 "Hallo ich bin ein Beispiel" 24
# immer 24 Zeichen
```

Beispiel:

```
var="Hallo ich bin ein Beispiel"
length="24"

filltext=$(bash lib/fillortrim.sh "$var" "$length")

echo "|...|$formfilltext|...|"
```

Script:

```
#!/bin/bash

var="$1"
length=${#var}
while [ $length -lt "$2" ]; do
    var="$var "
    ((length++))
done
var="${var:0:$2}"

echo "$var"
```

```
exit 0
```

fillortrim.sh (Python3)

Auffüllen oder trimmen von einem Text auf X Zeichen. Es erzeugt kein typischer Rückgabewert, sondern gibt den Text bearbeitet aus.
Klassischer Anwendungsfall: textbasierte Tabelle, bei der ein Text immer die gleiche Länge haben soll, um eine sinnvolle, visuelle Tabelle zu bilden.

HOWTO:

```
python3 $0 $var $length
python3 $0 "Hallo ich bin ein Beispiel" 24
# immer 24 Zeichen
```

Beispiel:

```
var = "Hallo ich bin ein Beispiel"
length = 24

formfilltext = var.ljust(length)[:length]

print(f"|...|{formfilltext}|...|")
```

Script:

```
import sys

def fill_or_trim(var, length):
    # Füllen bis zur gewünschten Länge
    while len(var) < length:
        var += " "
    # Abschneiden auf die gewünschte Länge
    var = var[:length]
    return var

if __name__ == "__main__":
```

```
if len(sys.argv) != 3:
    print("Usage: python fillortrim.py <string> <length>")
    sys.exit(1)

var = sys.argv[1]
length = int(sys.argv[2])

result = fill_or_trim(var, length)
print(result)
```

Headless sourcing-fähige Snippets

dabch2hz.sh

Ausgabe von DAB-Kanal in Hertz. Ideal für Scripte, die einen Kanal übergeben und den Rückgabewert verarbeiten.
Eigener Anwendungsfall: Frequenzumrechnung in Verbindung mit einem HackRF.

Der Rückgabewert ist Hertz.

HOWTO:

```
$0 12D
```

Beispiel:

```
ch="12D"


freq=$(bash lib/dabch2hz.sh "$ch")
# Hier kann mit $freq weitergearbeitet werden
```

Script:

```
#!/bin/bash


ch=$(echo "$1" | sed 's/\([A-Z]\)/\L\1/g')
□
case "$ch" in
  5a) freq="174928000" ;;
  5b) freq="176640000" ;;
  5c) freq="178352000" ;;
  5d) freq="180064000" ;;
  6a) freq="181936000" ;;
  6b) freq="183648000" ;;
  6c) freq="185360000" ;;
  6d) freq="187072000" ;;
  7a) freq="188928000" ;;
```

7b) freq="190640000" ;;

7c) freq="192352000" ;;

7d) freq="194064000" ;;

8a) freq="195936000" ;;

8b) freq="197648000" ;;

8c) freq="199360000" ;;

8d) freq="201072000" ;;

9a) freq="202928000" ;;

9b) freq="204640000" ;;

9c) freq="206352000" ;;

9d) freq="208064000" ;;

10a) freq="209936000" ;;

10n) freq="210096000" ;;

10b) freq="211648000" ;;

10c) freq="213360000" ;;

10d) freq="215072000" ;;

11a) freq="216928000" ;;

11n) freq="217088000" ;;

11b) freq="218640000" ;;

11c) freq="220352000" ;;

11d) freq="222064000" ;;

12a) freq="223936000" ;;

12n) freq="224096000" ;;

12b) freq="225648000" ;;

12c) freq="227360000" ;;

12d) freq="229072000" ;;

13a) freq="230784000" ;;

13b) freq="232496000" ;;

13c) freq="234208000" ;;

13d) freq="235776000" ;;

13e) freq="237488000" ;;

13f) freq="239200000" ;;

la) freq="1452960000" ;;

lb) freq="1454672000" ;;

lc) freq="1456384000" ;;

ld) freq="1458096000" ;;

le) freq="1459808000" ;;

lf) freq="1461520000" ;;

lg) freq="1463232000" ;;

lh) freq="1464944000" ;;

```
  li) freq="1466656000" ;;
  lj) freq="1468368000" ;;
  lk) freq="1470080000" ;;
  ll) freq="1471792000" ;;
  lm) freq="1473504000" ;;
  ln) freq="1475216000" ;;
  lo) freq="1476928000" ;;
  lp) freq="1478640000" ;;
  2a) freq="47936000" ;;
  2b) freq="49648000" ;;
  2c) freq="51360000" ;;
  2d) freq="53072000" ;;
  3a) freq="54928000" ;;
  3b) freq="56640000" ;;
  3c) freq="58352000" ;;
  3d) freq="60064000" ;;
  4a) freq="61936000" ;;
  4b) freq="63648000" ;;
  4c) freq="65360000" ;;
  4d) freq="67072000" ;;
esac

echo "$freq"
exit 0
```

Headless sourcing-fähige Snippets

fm2hz.sh

Ausgabe einer Frequenz von FM in Hertz. Ideal für Scripte, die eine Frequenz übergeben und den Rückgabewert verarbeiten. Es ist egal, ob die Frequenz mit Punkt oder Komma geschrieben wird. Eigener Anwendungsfall: Frequenzumrechnung in Verbindung mit einem HackRF.

Der Rückgabewert ist Hertz.

HOWTO:

```
$0 89.7
```

Beispiel:

```
# Eingabe-Beispiele
eingabe="89.7"
eingabe="106,4"
eingabe="102.70"

freq=$(bash lib/fm2hz.sh "$eingabe")
# Hier kann mit $freq weitergearbeitet werden
```

Script:

```
#!/bin/bash

input=$(echo "$1" | sed 's/,/./g')

if [[ "$input" == *.* ]]; then
    integer_part=$(echo "$input" | cut -d'.' -f1)
    decimal_part=$(echo "$input" | cut -d'.' -f2)
else
    integer_part=$input
    decimal_part=""
fi
```

```
freq="${integer_part}${decimal_part}"
zeros_to_add=$((6 - ${#decimal_part}))

while [ $zeros_to_add -gt 0 ]; do
    freq="${freq}0"
    ((zeros_to_add--))
done

echo "$freq"
exit 0
```

Headless sourcing-fähige Snippets

checkpackage.sh

Es wird geprüft, ob übergebene System-Packages installiert sind. Kann auf mehreren Distros verwendet werden: Debian, Red Hat, Arch, OpenSUSE und alle Derivate davon.

Der Rückgabewert ist 0 (true) oder 1 (false).

HOWTO:

```
$0 wget
```

Beispiel:

```
if [ "$(bash lib/checkpackage.sh wget)" = "0" ]; then
    echo "installiert"
else
    echo "nicht installiert"
fi
```

Script:

```
#!/bin/bash


if [ -f /etc/debian_version ]; then
    # Debian-based
    if dpkg -s "$1" &> /dev/null; then
        echo "0"
    else
        echo "1"
    fi
elif [ -f /etc/redhat-release ]; then
    # Red Hat-based
    if rpm -q "$1" &> /dev/null; then
        echo "0"
    else
```

```
        echo "1"
    fi
elif [ -f /etc/arch-release ]; then
    # Arch-based
    if pacman -Qi "$1" &> /dev/null; then
        echo "0"
    else
        echo "1"
    fi
elif [ -f /etc/SuSE-release ]; then
    # openSUSE
    if zypper se --installed-only "$1" &> /dev/null; then
        echo "0"
    else
        echo "1"
    fi
else
    echo "Unsupported Linux distribution"
    exit 1
fi
exit 0
```

picorvid.sh

Ausgabe des Typs einer Webcam/Kamera. Ist es ein Bild oder Video - der Rückgabewert ist zur Weiterverarbeitung.
Dabei ist der Rückgabewert wie folgt:
0 = Bild
1 = Video
2 = nicht unterstützt
3 = Fehler/ungültig.
Die Weiterverarbeitung ist mit ffmpeg oder wget/curl ideal, wenn man vorher einsortieren muss, um welchen Typ es sich handelt. Bilder lassen sich wie gewohnt herunterladen und ffmpeg kann Einzelbilder aus Streams extrahieren.

HOWTO:

```
$0 URL
```

Beispiel:

```
url="http://192.168.1.10/video.cgi"
typ=$(bash lib/picorvid.sh "$url")

# VARIANTE 1
case "$typ" in
    1)
        echo "MJPEG-Stream erkannt"
        ;;
    0)
        echo "Einzelbild erkannt"
        ;;
    2)
        echo "Bekannt, aber nicht unterstützt"
        ;;
    3)
        echo "Ungültige oder unbekannte URL"
        ;;
esac
```

```
# VARIANTE 2
if [ "$typ" = "1" ]; then
    echo "Video/Stream"
elif [ "$typ" = "0" ]; then
    echo "Bild"
fi
```

Script:

```
#!/bin/bash

# Video
if echo "$1" | grep -qE
'\.mjpg|\.mjpeg|faststream|video\.cgi|GetOneShot|mjpg\.cgi|videostream\.cgi|\/image|\?action\=stream|\/cam_.\.c
gi|\.r-kom\.de'; then
echo "1"
exit 0

# Bild
elif echo "$1" | grep -qE 'snapshot\.cgi|SnapshotJPEG|\.jpg|api\.cgi|cgi-
bin\/camera|alarmimage|oneshotimage|image\/Index|CGIProxy\.fcgi|nph-
jpeg\.cgi|onvif\/snapshot|GetImage\.cgi'; then
echo "0"
exit 0

# nicht unterstützt
elif echo "$1" | grep -qE 'GetData\.cgi|mjpeg\.cgi|\.png'; then
echo "2"
exit 0
else
# ansonsten ungültig
echo "3"
exit 0
fi
```

port.sh

Gibt aus der übergebenen URL den Port aus. Falls keiner übergeben wurde, bezieht man sich auf den Standardport je nach HTTP oder HTTPS.
http://abc.de:8080/test --> 8080
https://abc.de/test --> 443
http://abc.de/test --> 80

HOWTO:

```
$0 URL
```

Beispiel:

```
url="http://beispiel.de:8080/snapshot.cgi"
port=$(bash src/port.sh "$url")
echo "Verwendeter Port: $port"
# --> 8080
```

Script:

```
#!/bin/bash

url=$(echo "$1" | sed ':a; s/\([^ ]\) /\1%20/g; ta; s/^%20//; s/%20$//' | sed s'/ //'g)

#zerlege URL in Adresse und Port
addr=$(echo "$url" | grep -oP '^https?://\K[^:/]+')
port=$(echo "$url" | grep -oP ':\K[0-9]+')

#setze Standard-Port, wenn nicht anders angegeben
if [ -z "$port" ]; then
	if echo "$url" | grep -q "https"; then
		port="443"
	else
		port="80"
```

```
fi
fi

echo "$port"
exit 0
```

rdm.sh

Ausgabe einer random Zahl zwischen $1 und $2. Obergrenze: 32767, da RANDOM nicht mehr verarbeiten kann.

HOWTO:

```
$0 100 199
```

Beispiel:

```
min="100"
max="199"

zahl=$(bash lib/rdm.sh "$min" "$max")
```

Script:

```
#!/bin/bash

min=$1
max=$2

echo $((RANDOM % (max - min + 1) + min))
```

Headless sourcing-fähige Snippets

rdmfreqfm.sh

Es entstehen $1 random Frequenzen in einer vordefinierten Range (FM). Es kann auch ein Abstand $2 zwischen den Frequenzen eingegeben werden. Dies eignet sich gut für Frequenzplanlogistik oder Experimente.

HOWTO:

```
$0 3 5
```

Ausagabe:
88.1
107.5
97.7

Der Abstand $2 besagt, dass bei Eingabe von "2" bei einer Frequenz von 90.0 MHz NICHT 89.8, 89.9 und 90.1, 90.2 generiert werden darf.

Beispiel:

```
anzahl="3"
abstand="5"

# BEISPIEL 1
frequenzen=$(bash lib/rdmfreqfm.sh "$anzahl" "$abstand")

# BEISPIEL 2
mapfile -t freqenzen < <(bash inc/gen_frequencies.sh "$anzahl" "$abstand")

for f in "${frequenzen[@]}"; do
    echo "Frequenz: $f MHz"
done
```

Script:

```
#!/bin/bash

count="$1"
scope="$2"
min="876"
max="1079"
max_attempts="500"  # Maximale Anzahl von Versuchen, um eine nicht überlappende Zufallszahl zu finden

if [ -z "$2" ]; then
    scope="3"
fi

generated_numbers=()

function is_nearby {
    local number=$1
    for n in "${generated_numbers[@]}"; do
        if (( number >= n - $scope && number <= n + $scope )); then
            return 1
        fi
    done
    return 0
}

while [ ${#generated_numbers[@]} -lt $count ]; do
    attempts=0
    while true; do
        if (( attempts >= max_attempts )); then
            echo "ERROR"
            exit 1
        fi

        random_number=$(shuf -i ${min}-${max} -n 1)

        if is_nearby $random_number; then
            generated_numbers+=($random_number)
            formatted_number=$(echo "$random_number" | sed 's/\(.*\)\(.\)$/\1.\2/')
            echo "$formatted_number"
            break
        else
```

```
            (( attempts++ ))
        fi
    done
    attempts=0
done
```

Headless sourcing-fähige Snippets

rdsps.sh

RDS wird generiert und fix auf 8 Zeichen gesetzt. Ideal für die Weitergabe an einen RDS-Decoder. Hier kann der Name/Text an $1 übergeben werden und ein Modus für Großbuchstaben eingeschaltet werden:
$2=0 --> Großbuchstaben,
$2=1 --> normal, wie Input.
Bei kurzen Namen/Texten wird das RDS zentriert. Bei langen Namen/Texten wird das RDS gekapert auf 8 Zeichen. Untypische Zeichen fürs RDS werden ersetzt.

HOWTO:

```
input="Regio 8"
mode=1  # oder 0 für erzwungene Großbuchstaben

rds=$(bash lib/rdsps.sh "$input" "$mode")
echo "RDS-PS: [$rds]"
```

Script:

```
#!/bin/bash

if [ "$2" = "1" ]; then
    # Entferne alle Zeichen außer Großbuchstaben, Zahlen und bestimmten Satzzeichen
    var=$(echo "$1" | sed 's/[^A-Za-z0-9.,!?*-]/_/g; s/ /_/g; s/Ä/A/g; s/Ö/O/g; s/Ü/U/g; s/ä/a/g; s/ö/o/g; s/ü/u/g')
elif [ "$2" = "0" ]; then
    # Ersetze Kleinbuchstaben durch Großbuchstaben und entferne alle anderen nicht gewünschten Zeichen
    var=$(echo "$1" | sed 's/[a-z]/\U&/g' | sed 's/[^A-Z0-9.,!?*-]/_/g; s/ /_/g; s/Ä/A/g; s/Ö/O/g; s/Ü/U/g; s/ä/a/g;
s/ö/o/g; s/ü/u/g')
fi

length=${#var}

if [ "$length" = "1" ]; then
    echo "___${var}____"
```

```
elif [ "$length" = "2" ]; then
    echo "___${var}___"

elif [ "$length" = "3" ]; then
    echo "__${var}___"

elif [ "$length" = "4" ]; then
    echo "__${var}__"

elif [ "$length" = "5" ]; then
    echo "_${var}__"

elif [ "$length" = "6" ]; then
    echo "_${var}_"

elif [ "$length" = "7" ]; then
    echo "${var}_"

elif [ "$length" = "8" ]; then
    echo "${var}"

elif [ "$length" -ge "9" ]; then
    var="${var:0:8}"
    echo "$var"
fi

exit 0
```

Headless sourcing-fähige Snippets

runtxt.sh

Hier kann ein Text $1 in einem textbasierten Programm durchlaufen. Hierbei lässt sich die Länge der anzuzeigenden Zeichen mit $2 bestimmen und die Schnelligkeit in ms in $3. Ideal für z. B. Displays mit begrenzter Ausgabelänge. Hier kann der Text ganz einfach durchlaufen.
$1 --> Text, der gescrollt wird
$2 --> Blockgröße (sichtbare Länge)
$3 --> Pausenzeit pro Schritt (in ms)

HOWTO:

```
$0 "Hallo Welt!" 8 100
```

Beispiel:

```
text="System läuft normal, keine Vorkommnisse"
display="16"
ms="120"

bash src/runtxt.sh "$text" "$display" "$ms"
```

Script:

```
#!/bin/bash

if [ -z "$3" ]; then
    exit 1
fi

text="$1"
block_size="$2"
sleep_time=$(echo "scale=3; $3 / 1000" | bc)

# Den Text erweitern, um eine ausreichende Anzahl von Leerzeichen für nahtloses Scrollen hinzuzufügen
padded_text="$(printf '%*s' $block_size)${text}"
padded_length=${#padded_text}
```

```
while true; do
    for (( i=0; i<padded_length; i++ )); do
        # Substring von i bis i+block_size Zeichen
        if (( i + block_size <= padded_length )); then
            substring="${padded_text:i:block_size}"
        else
            substring="${padded_text:i}"
            remaining_length=$((block_size - ${#substring}))
            substring="${substring}${padded_text:0:remaining_length}"
        fi
        echo -ne "$substring\r"
        sleep "$sleep_time"
    done
done

exit 0
```

tvch2hzdigeu.sh

Ausgabe von TV-Kanal digital in Hertz. Ideal für Scripte, die einen Kanal übergeben und den Rückgabewert verarbeiten.
Eigener Anwendungsfall: Frequenzumrechnung in Verbindung mit einem HackRF.
Dieses Snippet ist ausgelegt auf Europa, da andererorts die Frequenzen hinsichtlich der Zentralfrequenz etwas verschoben sein können.
Die Eingaben können mit Kxx, Exx, Cxx für die normalen Kanäle, Sxx für Sonderkanäle und Dxx für die Digitalkanäle (Kabelanschluss) erfolgen.
Range: K21-K69, D73-D858, S3-S41. Die Ausgabe erfolgt auf der Grundlage der Mittenfrequenz für DVB-T und DVB-C. Keine Berücksichtigung von analogen Signalen.

Hinweis: Frequenzen oberhalb von 694 MHz dürfen nicht mehr für Rundfunkausstrahlung (z. B. DVB-T) genutzt werden, da dieser Bereich durch die sogenannte digitale Dividende II für den Mobilfunk (LTE/5G) freigegeben wurde.
Die nachfolgenden Frequenzen sind daher ausschließlich aus historischen und dokumentarischen Gründen aufgeführt und dürfen nicht mehr für die Ausstrahlung verwendet werden außer in geschlossenen Systemen wie ein eigener Kabelanschluss.

HOWTO:

```
$0 K24
```

Beispiel:

```
ch="K24"


freq=$(bash lib/tvch2hzeu.sh "$ch")
# Hier kann mit $freq weitergearbeitet werden
```

Script:

```
#!/bin/bash


ch=$(echo "$1" | sed 's/\([A-Z]\)/\L\1/g' | sed -e 's/0*\([0-9]\)/\1/g')


case "$ch" in
```

```
d73) freq="73000000" ;;
d81) freq="81000000" ;;
d114) freq="114000000" ;;
d122) freq="122000000" ;;
d130) freq="130000000" ;;
d138) freq="138000000" ;;
d146) freq="146000000" ;;
d154) freq="154000000" ;;
d162) freq="162000000" ;;
d170) freq="170000000" ;;
d178) freq="178000000" ;;
d186) freq="186000000" ;;
d194) freq="194000000" ;;
d202) freq="202000000" ;;
d210) freq="210000000" ;;
d218) freq="218000000" ;;
d226) freq="226000000" ;;
c5|k5|e5) freq="177500000" ;;
c6|k6|e6) freq="184500000" ;;
c7|k7|e7) freq="191500000" ;;
c8|k8|e8) freq="198500000" ;;
c9|k9|e9) freq="205500000" ;;
c10|k10|e10) freq="212500000" ;;
c11|k11|e11) freq="219500000" ;;
c12|k12|e12) freq="226500000" ;;
s2) freq="114000000" ;;
s3) freq="122000000" ;;
s4) freq="130000000" ;;
s6) freq="138000000" ;;
s7) freq="146000000" ;;
s8) freq="154000000" ;;
s9) freq="162000000" ;;
s10) freq="170000000" ;;
s11|d234) freq="234000000" ;;
s13|d242) freq="242000000" ;;
s14|d250) freq="250000000" ;;
s15|d258) freq="258000000" ;;
s16|d266) freq="266000000" ;;
s17|d274) freq="274000000" ;;
s18|d282) freq="282000000" ;;
```

s19|d290) freq="290000000" ;;

s20|d298) freq="298000000" ;;

s21) freq="306000000" ;;

s22) freq="314000000" ;;

s23) freq="322000000" ;;

s24) freq="330000000" ;;

s25) freq="338000000" ;;

s26) freq="346000000" ;;

s27) freq="354000000" ;;

s28) freq="362000000" ;;

s29) freq="370000000" ;;

s30) freq="378000000" ;;

s31) freq="386000000" ;;

s32) freq="394000000" ;;

s33) freq="402000000" ;;

s34) freq="410000000" ;;

s35) freq="418000000" ;;

s36) freq="426000000" ;;

s37) freq="434000000" ;;

s38) freq="442000000" ;;

s39) freq="450000000" ;;

s40) freq="458000000" ;;

s41) freq="466000000" ;;

c21|k21|e21|d474) freq="474000000" ;;

c22|k22|e22|d482) freq="482000000" ;;

c23|k23|e23|d490) freq="490000000" ;;

c24|k24|e24|d498) freq="498000000" ;;

c25|k25|e25|d506) freq="506000000" ;;

c26|k26|e26|d514) freq="514000000" ;;

c27|k27|e27|d522) freq="522000000" ;;

c28|k28|e28|d530) freq="530000000" ;;

c29|k29|e29|d538) freq="538000000" ;;

c30|k30|e30|d546) freq="546000000" ;;

c31|k31|e31|d554) freq="554000000" ;;

c32|k32|e32|d562) freq="562000000" ;;

c33|k33|e33|d570) freq="570000000" ;;

c34|k34|e34|d578) freq="578000000" ;;

c35|k35|e35|d586) freq="586000000" ;;

c36|k36|e36|d594) freq="594000000" ;;

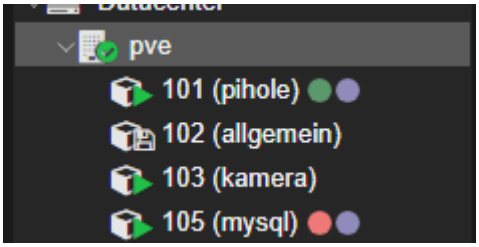
c37|k37|e37|d602) freq="602000000" ;;

```
    c38|k38|e38|d610) freq="610000000" ;;
    c39|k39|e39|d618) freq="618000000" ;;
    c40|k40|e40|d626) freq="626000000" ;;
    c41|k41|e41|d634) freq="634000000" ;;
    c42|k42|e42|d642) freq="642000000" ;;
    c43|k43|e43|d650) freq="650000000" ;;
    c44|k44|e44|d658) freq="658000000" ;;
    c45|k45|e45|d666) freq="666000000" ;;
    c46|k46|e46|d674) freq="674000000" ;;
    c47|k47|e47|d682) freq="682000000" ;;
    c48|k48|e48|d690) freq="690000000" ;;
    c49|k49|e49|d698) freq="698000000" ;;
    c50|k50|e50|d706) freq="706000000" ;;
    c51|k51|e51|d714) freq="714000000" ;;
    c52|k52|e52|d722) freq="722000000" ;;
    c53|k53|e53|d730) freq="730000000" ;;
    c54|k54|e54|d738) freq="738000000" ;;
    c55|k55|e55|d746) freq="746000000" ;;
    c56|k56|e56|d754) freq="754000000" ;;
    c57|k57|e57|d762) freq="762000000" ;;
    c58|k58|e58|d770) freq="770000000" ;;
    c59|k59|e59|d778) freq="778000000" ;;
    c60|k60|e60|d786) freq="786000000" ;;
    c61|k61|e61|d794) freq="794000000" ;;
    c62|k62|e62|d802) freq="802000000" ;;
    c63|k63|e63|d810) freq="810000000" ;;
    c64|k64|e64|d818) freq="818000000" ;;
    c65|k65|e65|d826) freq="826000000" ;;
    c66|k66|e66|d834) freq="834000000" ;;
    c67|k67|e67|d842) freq="842000000" ;;
    c68|k68|e68|d850) freq="850000000" ;;
    c69|k69|e69|d858) freq="858000000" ;;
esac

echo "$freq"
exit 0
```

Proxmox: VM/CT gesperrt

Manchmal kommt es in Proxmox vor, dass bei einem unterbrochenen oder fehlerhaftem Update die Maschine sich nicht mehr entsperrt. Die Maschine bleibt gesperrt und es sind keine Handlungen wie Konfiguration oder Backup möglich.
Bemerkbar macht sich dies durch das kleine Schloss auf der Verwaltungsoberfläche.



Demnach muss die Maschine händisch entsperrt werden. Die Vorgehensweise ist bei VMs und CTs jeweils unterschiedlich, aber nicht gravierend.

Für VMs folgenden Befehl in der Proxmox-Shell absetzen:

```
qm unlock ID
```

Für CTs:

```
pct unlock ID
```

Fertig.

Archive unter Linux

zip

Das Tool zip ist vor allem aus der Windows-Welt bekannt, ist aber ebenso bei Linux-basierten Betriebssystemen verfügbar.

Einzelne Dateien in einem komprimierten Archiv zusammenfassen:

```
zip archiv.zip inhalt1 inhalt2
```

Komplette Ordner in einem komprimierten Archiv zusammenfassen:

```
zip -r archiv.zip ordner1 ordner2 ordner3
```

Komprimiertes Archiv entpacken:

```
unzip archiv.zip
```

Inhalt eines komprimierten Archivs anzeigen:

```
unzip -l archiv.zip
```

bz2, bzip2

Neben gzip gibt es noch bzip2, es ist gzip sehr ähnlich, verwendet aber einen anderen Algorithmus.

Eine Datei komprimieren:

```
bzip2 file
```

Datei dekomprimieren:

```
bunzip2 file.bz2
```

Dateien in einem komprimierten Archiv zusammenfassen:

```
tar cfvj archiv.tar.bz2 inhalt1 inhalt2
```

Archiv dekomprimieren und auspacken:

```
tar xfvj archiv.tar.bz2
```

gz, gzip

Da ein tar-Archiv, wie oben erwähnt, ohne Zusatzoptionen nicht komprimiert ist, kann dies mit der Zusatzoption gzip geschehen. gzip steht für GNU zip und dem tar-Archiv wird die Dateiendung .gz angehängt.

Eine Datei komprimieren:

```
gzip file
```

Datei dekomprimieren:

```
gunzip file
```

Dateien in einem komprimierten Archiv zusammenfassen:

```
tar cfvz archiv.tar.gz inhalt1 inhalt2
```

Archiv dekomprimieren und auspacken:

```
tar xfvz archiv.tar.gz
```

tar

Das Programm tar steht ursprünglich für **T**ape **Ar**chiver, es wurde verwendet um Daten auf Bandlaufwerken zu sichern. Es ist heute noch sehr beliebt und verbreitet.
Ein reines tar-Archiv ist **nicht** komprimiert.

Entpacken eines Archivs:

```
tar xfv archiv.tar
```

Dateien/Ordner in ein Archiv packen:

```
tar cfv archiv.tar inhalt1 inhalt2 inhalt3
```

Komprimierte Archive erstellen:

```
tar cfzv archiv.tar inhalt1 inhalt2 inhalt3
```

Inhalt eines Archivs auflisten:

```
tar tfv archiv.tar
```

Legende:

x = entpacken (extract)
f = Datei (file)
v = Details anzeigen (verbose)
c = erstellen (create)

VPN-Server mit Wireguard / Side-to-Side-VPN / alles tunneln

Es kann mehrere Gründe geben, einen VPN-Tunnel zu nutzen. Sei es zwecks der Anonymität, weil man geogeblockte Seiten öffnen möchte oder die normale Praxis - einfach nur, weil man auf der Gegenstelle einen Server hat, deren Dienste man sicher erreichen möchte.

In unserem Fall möchten wir als IP gerne diese externe IP haben, sodass wir geogeblockte Seiten öffnen können.

Als Server kommt ein V-Server in einem Rechenzentrum zum Einsatz (Debian), als Client ist es lokal ein LXC-Container in Proxmox (Ubuntu). Dies kann natürlich auch der lokale Rechner sein oder ein lokaler Laptop mit Linux drauf.

Wir gehen davon aus, dass wir auf beiden Systemen root sind, ansonsten ist - in diesem Fall - vor **jedem** Befehl ein sudo zu setzen.

Allgemein/Vorbereiten

Dieser Abschnitt ist auf beiden Systemen auszuführen.

Sollte das System recht frisch sein, ist unbedingt ein Aktualisieren der apt-Datenbank von nöten, in allen anderen Fällen kann es nicht schaden:

```
apt update -y && apt upgrade -y
```

Ggf. Wireguard selbst installieren - sollte es nicht vorhanden sein:

```
apt install wireguard
```

Als nächstes muss in der Datei */etc/sysctl.conf* unbedingt die Zeile *#net.ipv4.ip_forward=1* auskommentiert werden:

```
nano /etc/sysctl.conf
```

aus

```
#net.ipv4.ip_forward=1
```

wird

```
net.ipv4.ip_forward=1
```

Als nächstes generieren wir die Schlüssel für den gegenseitigen Austausch:

```
umask 077; wg genkey | tee /etc/wireguard/privatekey | wg pubkey > /etc/wireguard/publickey
```

Diesen lassen wir uns jeweils auf beiden Systemen anzeigen für die Konfigurationsdatei nachher:

```
cat /etc/wireguard/privatekey && cat /etc/wireguard/publickey
```

Serverkonfiguration

Als erstes fragen wir ab, welches Interface hardwareseitig benutzt wird. Dies ist meist eth0. Über dieses Interface soll nachher alles laufen. Dieses ist mit

```
ip a
```

abzufragen.

Nun beginnen wir mit der Erstellung der Konfigurationsdatei auf dem Server. Folgender Inhalt muss in die Wireguard-Konfiguration *wg0.conf*, wie wir sie hier nennen. Wenn ein anderer Name als wg0 verwendet wird, muss in allen Fällen, wo wir wg0 verwenden, natürlich der individuelle Name stehen.

```
nano /etc/wireguard/wg0.conf
```

```
[Interface]
PrivateKey = SERVER-PRIVATEKEY
Address = 172.31.0.1/32
SaveConfig = true
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT; iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE
ListenPort = 51820


[Peer]
```

```
PublicKey = CLIENT-PUBLICKEY
AllowedIPs = 172.31.0.2/32, X.X.X.X/XX
```

Hier gibt es einiges zu beachten. Der SERVER-PRIVATEKEY und CLIENT-PUBLICKEY muss mit den Keys vom Server und Clienten ersetzt werden. Das Interface ist zu beachten, wie oben beschrieben. Die IP-Adresse 172.31.0.1/32 suggeriert ein Netz, deren /32-Subnetz technischer Weise nur 2 IP-Adressen erlaubt. Nämlich Server (172.31.0.1) und Client (172.31.0.2). Diese beiden benutzen dieses Netz und kommunizieren über den Tunnel. Der Port 51820 ist der Wireguard-Standard-Port, kann aber geändert werden, muss aber dann natürlich auf beiden Seiten gleich sein. Die PostUp und PostDown-Zeilen bilden einfach die IPtables, wie wo was geroutet wird.

Im Abschnitt Peer bei den Allowed IPs muss unbedingt die IP des Clienten enthalten sein, in diesem Fall eben 172.31.0.2/32. Für unseren Fall des Gateways ist es unbedingt erforderlich, das eigene Netz, welches wir daheim haben, dort einzusetzen.

Haben wir die 192.168.1.6 als IP in einem /24er-Netz, ist, sofern das ganze Netz geroutet werden darf, die 192.168.1.0/24 stehen. Im Falle eines 172.20.2.86 als IP bei einem /16er-Netz, ist dort auch die 172.20.0.0/16 zu wählen. Einzelne IPs können auch gewählt werden, wenn nur ein Gerät durch den Tunnel sollte.

Clientkonfiguration

Wir legen auch auf dem Client die Konfigurationsdatei mit gleichem Namen wg0 an:

```
nano /etc/wireguard/wg0.conf
```

```
[Interface]
PrivateKey = CLIENT-PRIVATEKEY
Address = 172.31.0.2/32

[Peer]
PublicKey = SERVER-PUBLICKEY
Endpoint = Adresse:51820
AllowedIPs = 0.0.0.0/0
PersistentKeepalive = 25
```

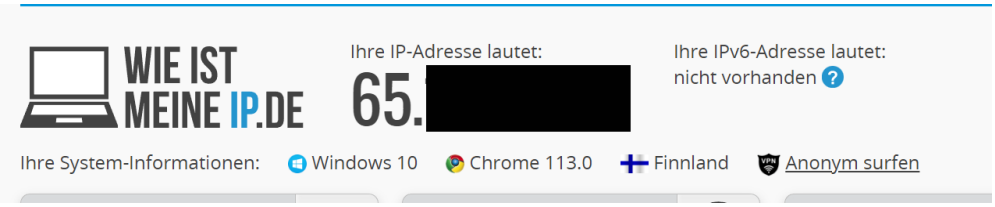
Hier ist natürlich wieder zu beachten, dass die Keys entsprechend eingesetzt werden. Bei Adresse kann die IP des Servers oder auch eine Domain stehen. Die AllowedIPs mit dem Wert 0.0.0.0/0 sagt aus, dass alles über den VPN-Server geroutet wird, also auch Internet.

Und da wir hier mit Wireguard im UDP-Protokoll unterwegs sind, gibt es keine Bestätigung der Verbindung. Demzufolge muss permanent der PersistentKeepalive abgefragt werden, hier alle 25

Sekunden.

Abschluss

Sollte der Dienst nach einem Neustart eines der beiden Systemen automatisch wieder starten, ist folgender Befehl abzusetzen:



War man erfolgreich, sollte dies bei https://wieistmeineip.de erscheinen.

```
systemctl enable wg-quick@wg0
```

Zeitumstellung via timedatectl

Nach dem Installieren einiger Distributionen, insbesondere die des Raspberry Pi, kann es immer mal wieder vorkommen, dass sich die Zeitzone nicht von selbst einstellt.

Abhilfe schafft folgender Befehl:

```
sudo timedatectl set-timezone Europe/Berlin
```

Wobei "Europe/Berlin" dementsprechend anzupassen ist. Mit diesem Befehl

```
timedatectl list-timezones
```

werden alle möglichen Timezones gelistet.

Sonstige Tags: timezone

Systemdienst einrichten

Um einen Systemdienst auf Linux einzurichten, um ihn als "service XYZ start" starten zu können, sind folgende Schritte nötig:

Es wird davon ausgegangen, dass diese Schritte als sudo-Mitglied gemacht werden.

`sudo nano /etc/systemd/system/dienst.service`

Im einfachsten Fall reicht folgendes aus:

```
[Unit]
Description=Eigener Dienst

[Service]
WorkingDirectory=/srv
ExecStart=/srv/script.sh arg1 arg2
User=scripter
Restart=always

[Install]
WantedBy=multi-user.target
```

In einigen Fällen, vor allem wenn while-Schleifen zum Einsatz kommen, kann dies auch komplexer werden:

```
[Unit]
Description=Eigener dienst
After=network.target
Restart=always
RestartSec=5

[Service]
ExecStart=/srv/script.sh arg1 arg2
WorkingDirectory=/srv
User=scripter
Group=scripter
Restart=always
StandardOutput=journal
```

```
StandardError=journal

[Install]
WantedBy=multi-user.target
```

Zum Schluss muss der Dienst noch aktiviert werden:

`sudo systemctl enable dienst.service`

Und anschließend kann der dienst gestartet und auf den Status geschaut werden:

`sudo service dienst start` `sudo service dienst status` `sudo service dienst stop`

SSH Forwarding (Portweiterleitung via SSH)

Oft ist man gezwungen, z. B. auf Proxmox zuzugreifen aber aus Sicherheitsgründen wird von einer öffentlichen Portfreigabe abgesehen. Demnach wäre eine Portweiterleitung nur über SSH interessant. Voraussetzung ist hier natürlich, dass SSH öffentlich freigegeben wurde. Dabei muss es nicht einmal direkt der Proxmox-Server, sondern es kann auch ein Zwischenserver, wie hier in diesem Beispiel erklärt wird, sein.

Ein SSH Forwarding von Client zu Server funktioniert hier im Beispiel wie folgt.

Der Client (localhost) möchte einen Port auf dem Server (lokal) öffnen. Hier wird zunächst ein SSH-Tunnel aufgebaut, der den Verkehr entsprechend portiert.

```
ssh -N -L 443:localhost:8006 user@beispielserver.de -p 2222
```

Hier wird der lokale Port 443 (HTTPS) auf dem am SERVER.DE befindlichen Port 8006 über SSH Port 2222 weitergeleitet. In diesem konkreten Beispiel lässt sich auf den Proxmox-Server zugreifen, ohne den Port nach außen hin öffnen zu müssen.

Sollte ein Zwischenserver vorhanden sein, der vor Proxmox agiert, sodass Proxmox keinen SSH-Port öffentlich geöffnet haben muss, muss zudem sichergestellt sein, dass der Zwischenserver mit dem gleichen SSH Forwarding versehen ist, sodass am Zwischenserver der Port 8006 auch zum Proxmox-Server auf Port 8006 führt. Dieser ist mit

```
ssh -N -L 8006:localhost:8006 sshuser@172.16.99.10 -p 22
```

einzureichen. In diesem konkreten Beispiel wird eben der Port 8006 auf dem Zwischenserver auf die IP und ebenfalls Port 8006 weitergeleitet.

Um diesen SSH Forwarding am Zwischenserver nicht immer aufmachen zu müssen, kann alternativ eine screen-Session mit genau dieser Portweiterleitung geschaffen werden oder idealer, die SSH-Verbindung im Hintergrund auszuführen mit

```
ssh -f -N -L 8006:localhost:8006 sshuser@172.16.99.10 -p 22
```

Auf dem Client kann auf die gleiche Weise agiert werden.

Hinweis: Beim Standard-Port 22 kann diese Port-Angabe auch weggelassen werden.

-L: local port forwarding

-f: Hintergrund

-N: kein einloggen erforderlich.

Demzufolge kann auf dem Client mit folgender Adresse nun entfernt der Port 8006 aufgerufen werden:

```
https://localhost
```

```
https://127.0.0.1
```

Den HTTPS-Standardport :443 schenken wir uns in diesem Fall.

Root-Login verbieten

Nach dem Installieren einiger Distributionen, insbesondere die des Raspberry Pi oder im Proxmox bei den LXC-Containern, ist es sicherheitsrelevant und deshalb erforderlich, wenn die Maschinen ans Netz gehen, dass der root-Login unterbunden wird.

Dies wird in der Datei **/etc/ssh/sshd_config** möglich gemacht.

```
sudo nano /etc/ssh/sshd_config
```

Aus

```
PermitRootLogin yes
```

wird

```
PermitRootLogin no
```

und fertig.

Anschließend den SSH-Dienst neustarten mit

```
sudo service sshd restart
```

sshd = ssh daemon.

Der root-Login ist somit nicht mehr möglich, es kann aber weiterhin als Benutzer mit

```
su
```

in den root-Account gewechselt werden. Unabhängig davon kann in einigen Distributionen mit

```
sudo befehl
```

interagiert werden, sofern der Benutzer in der sudoers-Tabelle gelistet ist.

PhpMyAdmin - Installation

Verwendet man eine Datenbank und möchte nicht alles mit dem Terminal regeln, ist es oft sinnvoller, eine Datenbankkonfigurationssoftware herbei zu ziehen. Es bietet sich PhpMyAdmin an.

In unserem Fall verwenden wir PhpMyAdmin mit MySQL8.0-Server, PHP8.3 und Apache2 als Webserver. Dateispeicherort wird der hauseigene Web-Pfad */var/www/html* verwendet.

Ferner gibt es mehrere Möglichkeiten, das ganze auf Linux mit den Rechten zu verbinden, in unserem Beispielfall läuft PhpMyAdmin unter demselben Benutzer wie der Webserver.

PHP, MySQL und Apache2 werden zuerst heruntergeladen, wenn noch nicht vorhanden:

```
sudo apt install mysql-server php8.3 apache2
```

Danach wechseln wir in unser Webverzeichnis:

```
cd /var/www/html
```

In diesem könnte es Rechteprobleme geben. Dann einfach das Verzeichnis für den Web-Benutzer freigeben:

```
sudo chown webuser:webuser -R /var/www/html
```

chown für *change owner* vom Verzeichnis */var/www/html*, das ganze rekursiv auf alle Unterverzeichnisse von dort, an den Benutzer *webuser* in der gleichnamigen Gruppe.

Danach laden wir uns PhpMyAdmin auf der offiziellen Seite *phpmyadmin.net* herunter, direkt auf den Server. Bitte die Versionsnummer beachten.

```
wget https://files.phpmyadmin.net/phpMyAdmin/5.2.1/phpMyAdmin-5.2.1-all-languages.zip
```

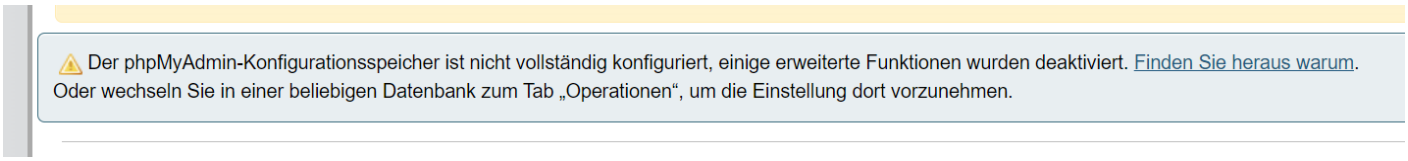
Dies entpacken wir dann an dieser Ort und Stelle.

...

PhpMyAdmin - Fehlermeldung Konfigurationsspeicher

Bei einer recht frischen Installation von phpMyAdmin kann es vorkommen, dass folgender Fehler im Webinterface angezeigt wird:

Der phpMyAdmin-Konfigurationsspeicher ist nicht vollständig konfiguriert, einige erweiterte Funktionen wurden deaktiviert. Finden Sie heraus warum. Oder wechseln Sie in einer beliebigen Datenbank zum Tab "Operationen", um die Einstellung dort vorzunehmen.



Screenshot der Fehlermeldung (v5.2.1)

Dieser Konfigurationsspeicher von phpMyAdmin benötigt eine eigene Datenbank oder eine eigene Tabelle auf dem MySQL-Server. Dies kann zur Abhilfe führen oder aber auch einfacher, diese Meldung ignorieren. Hierzu muss die Datei config.inc.php im root-Verzeichnis von phpMyAdmin editiert und am Ende davon folgendes eingefügt werden:

```
$cfg['PmaNoRelation_DisableWarning'] = true;
```

Ist die individuelle Konfigurationsdatei *config.inc.php* nicht vorhanden, dann muss im ersten Schritt die *config.sample.inc.php* umbenannt, bzw. kopiert und dann editiert werden.

Umbenennen:

```
mv config.sample.inc.php config.inc.php
```

..oder Kopieren:

```
cp config.sample.inc.php config.inc.php
```

..in allen Fällen bearbeiten:

```
nano config.inc.php
```

Mount - HDD/SSD mounten

Um eine Festplatte/SSD im Terminal zu mounten, muss eine Voraussetzung gegeben sein.

Als erstes muss die Platte ermittelt werden, wo sie zu finden ist in */dev/*.

```
fdisk -l
```

bzw. - einfacher ist bei fortgeschrittenen Systemkenntnisse auch das hier:

```
ls /dev/ | grep sd
```

Wir gehen davon aus, dass die Platte neu ist, bzw. neu erstellt wurde und es noch keine Formatierung gegeben hat. Ansonsten kann dieser Schritt übersprungen werden.

Die Platte kann in ext4 (Linux), NTFS (Windows) oder andere Optionen formatiert werden. Hier sind zwei Beispiele:

```
sudo mkfs.ext4 /dev/sdX
```

```
sudo mkfs.ntfs /dev/sdX
```

...wobei das X in sdX ersetzt werden muss auf den Laufwerksbuchstaben, der vorher ermittelt wurde.

Jetzt kann die Platte gemountet werden:

```
sudo mount /dev/sdX /mnt/hdd
```

Der Zielpfad, hier */mnt/hdd*, kann natürlich individuell sein. Das Verzeichnis muss aber im Vorfeld bereits erstellt worden sein.

```
sudo mkdir /mnt/hdd
```

Die Verwendung von sudo im letzten Schritt ist nicht immer erforderlich, da es auch in einem Benutzerordner stattfinden kann. Hier, bei */mnt/* wird aber sudo vorausgesetzt.

Benutzer zum sudo (sudoers) hinzufügen

Um einen Benutzer in die sudo-Administration hinzuzufügen, reicht nach dem Erstellen des Benutzers mit

```
sudo adduser BENUTZER
```

folgender Befehl aus:

```
sudo usermod -aG sudo BENUTZER
```

Apt-get Autocomplete / Autocompletion

Teilweise kommt es unter einem frisch installiertem Linux vor, dass bei *apt* bzw. *apt-get* die Autokomplettierung mit der Tab-Taste nicht funktioniert. Um dies zu aktivieren, sind folgende Schritte nötig.

Sudo wird ggf. benötigt, wenn kein root.

```
sudo apt install bash-completion
```

Danach ist diese Datei ohne root-Rechte zu editieren. Diese Datei ist benutzerspezifisch und erfordert in keinem Fall sudo:

```
nano ~/.bashrc
```

Inhalt am Ende der Datei einfügen:

```
if [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
fi
```

Fertig. Normalerweise funktioniert die Autokomplettierung ohne erneutes Einloggen.

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.

eigene Programme, headless

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
```

eigene Programme, headless

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
```

eigene Programme, headless

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
```

eigene Programme, headless

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.

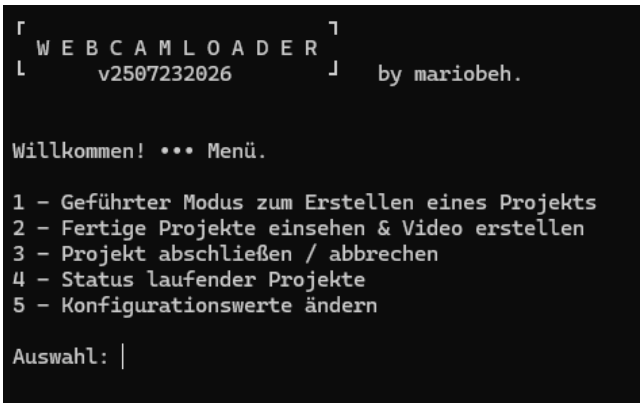
Das Script bietet drei Modi zum Besorgen der Bilder, die weiter unten genauer erklärt werden:
• **Menü**: Interaktives Menü mit allen Funktionen.
• **Quicky**: Komplette Steuerung via Argumente, kein Menü.
• **Cron**: Einmalaufrufe, zeitgesteuert z. B. über Crontab.
Für die **Videoerstellung** steht 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.

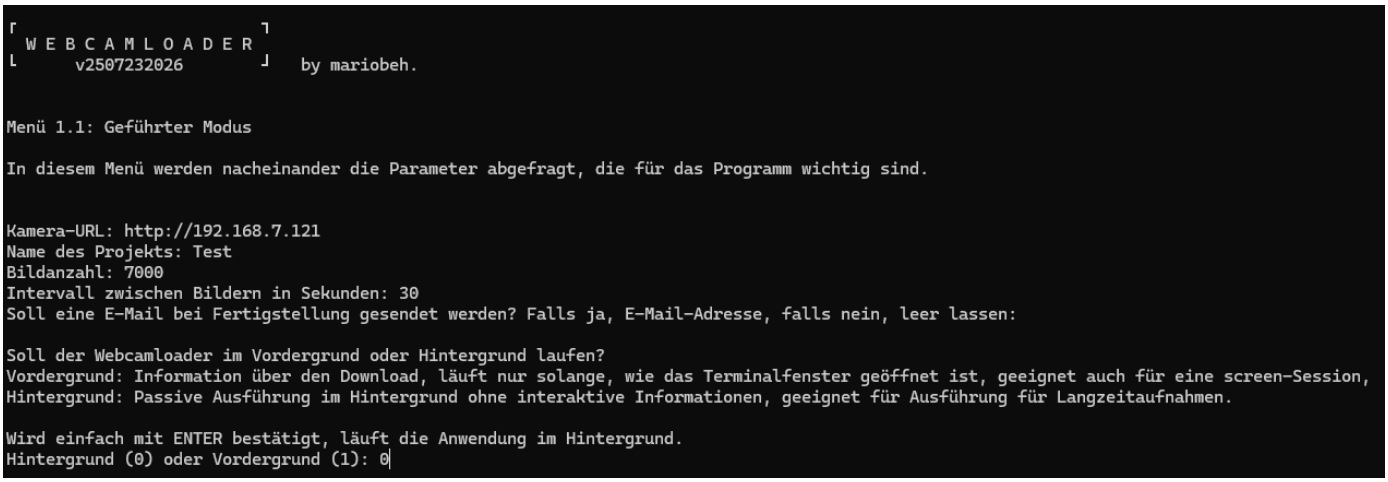
Im Menü kann zwischen verschiedenen Aktionen gewählt werden: Ein neues Projekt starten, ein bestehendes Projekt fortsetzen, Einstellungen ändern, den Status prüfen oder das Projekt abbrechen. Auch ein Video lässt sich hier erstellen, diese Funktion ist im Videosektor genauer erklärt. Der Einstieg erfolgt 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.

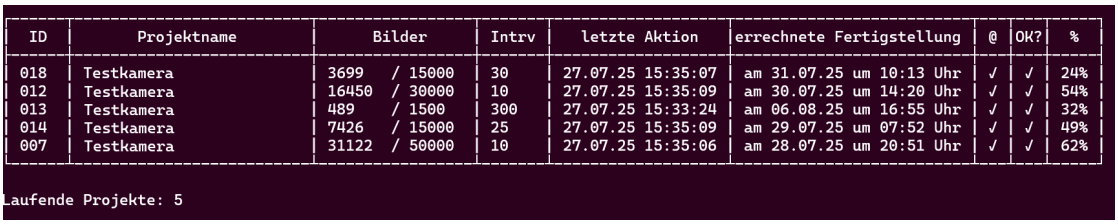
Der Status wird visuell durch ein ✔ (alles in Ordnung) oder ein ✖ (Problem erkannt) angezeigt. Zusätzlich ist ersichtlich, ob beim Start des Projekts eine Benachrichtigungs-E-Mail aktiviert wurde.

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

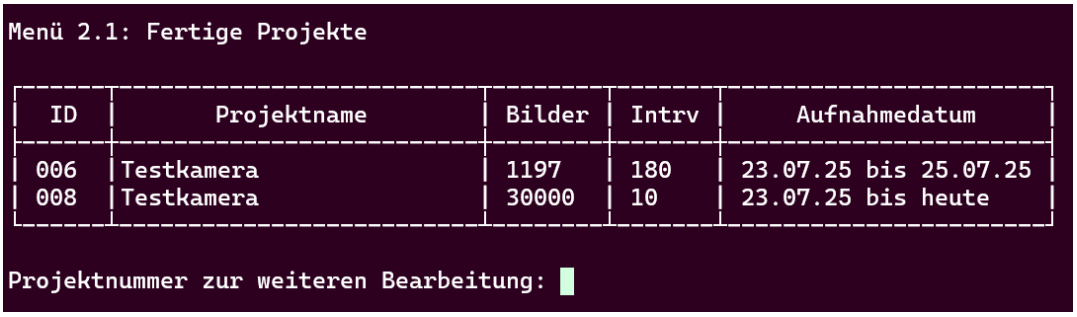
Sobald ein Projekt abgeschlossen ist – entweder durch Erreichen der definierten Bildanzahl oder durch manuellen Abbruch bei aktiviertem unbegrenztem Download – wird es unter dem Menüpunkt **Fertige Projekte** aufgelistet.

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*).

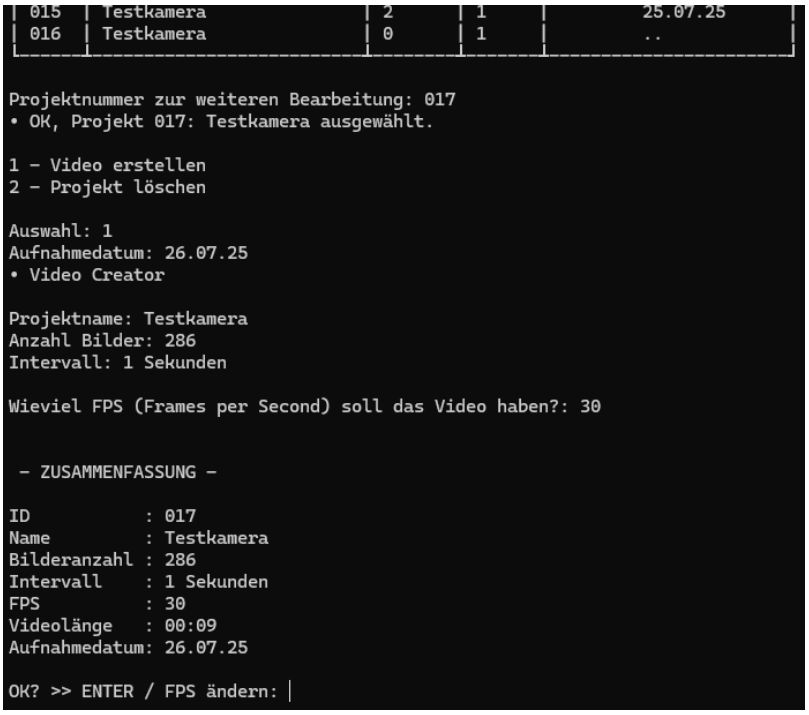


Projekt abbrechen

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.

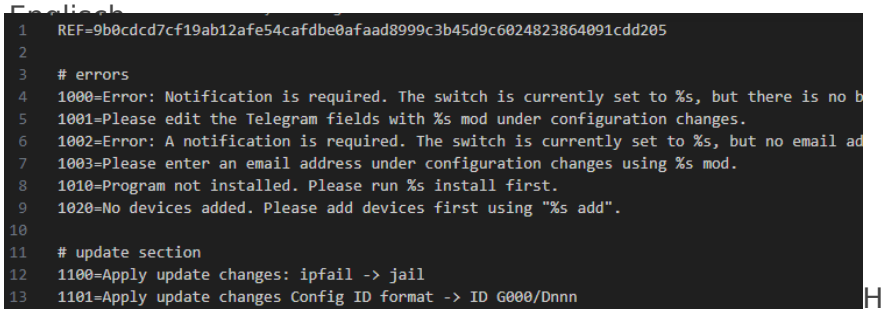
Dadurch wird verhindert, dass versehentlich eine unpassende oder fremde Sprachdatei verwendet wird, zum Beispiel aus einem anderen Script oder einer älteren Version. Stimmen Script und Sprachdatei nicht überein, wird die Sprachdatei nicht geladen.

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
```

eigene Programme, headless

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)

Telegram Benachrichtigungsdienst mit einer einzigen PHP

Ein Benachrichtigungsansatz, der schnell und einfach umzusetzen ist - in nur einer Datei, mit Ausnahme einer Konfiguration, wäre ein Träumchen. Das habe ich mir gedacht und habe das telegram.php ins Leben gerufen. Die Einstellungen zum Bot und zum Chat, sowie die Nachricht selbst, werden der PHP alle übergeben, sodass es eine Konfigurationsdatei überflüssig macht.

Diese Datei ist universell ausgelegt, betriebssystemunabhängig, also rein PHP - logisch, wie der Dateiname verrät. Alles, was benötigt wird, ist PHP selbst. Falls die PHP scripttechnisch aufgerufen wird, nicht nur im Browser, dann wird auch z. B. curl benötigt.

Der PHP müssen sämtliche Daten mit GET mitübergeben werden, damit das PHP diese intern verarbeiten kann.

Die zu übergebenden Parameter **bot**, **chatid** und **msg** werden hier erklärt:

• bot=... ist der BOT:TOKEN des Botfathers von Telegram,
• chatid=... ist der Chat, an dem die Nachricht gehen soll,
• msg=... ist die Nachricht, die übergeben wird.

Beispielaufruf:

```
http(s)://sub.domain.tld/telegram.php?bot=123456789:ABCdefGhIjKlMnOpQrStUvWxYz&chatid=987654321&msg=Hallo ich bins
curl "http(s)://sub.domain.tld/telegram.php?bot=123456789:ABCdefGhIjKlMnOpQrStUvWxYz&chatid=987654321&msg=Hallo ich bins"
php -f /srv/scripts/telegram.php "?bot=123456789:ABCdefGhIjKlMnOpQrStUvWxYz&chatid=987654321&msg=Hallo ich bins"
```

Zeile 1 zeigt einen Aufruf direkt im Browser. Macht an sich keinen Sinn, nur zur Veranschaulichung und eventuell zum Testen.
Zeile 2 zeigt einen CLI-Aufruf via curl. Ideal für Scripte.
Zeile 3 zeigt einen direkten Aufruf via php -f - hier wird die Leier danach übergeben. Technisch okay, aber irgendwie unsauber.

Ich empfehle ganz klar Zeile 2 zur Automation, Zeile 1 und 3 sind eher zum Testen da.

Hier das Script:

```
<?php
// Telegram Send Script - Funktioniert per CLI mit ?param=... und per HTTP-Aufruf

// Funktion zum Parsen der Query bei CLI
function getParams() {
    global $argc, $argv;

    // Wenn über HTTP aufgerufen
    if (php_sapi_name() !== 'cli') {
        return $_GET;
    }

    // CLI-Aufruf mit "?bot=...&chatid=...&msg=..."
    if ($argc >= 2) {
        parse_str(ltrim($argv[1], "?"), $params);
        return $params;
    }

    // Keine Parameter
    return [];
}

$params = getParams();

// Parameter prüfen
if (empty($params['bot']) || empty($params['chatid']) || empty($params['msg'])) {
    echo "[FEHLER] Bot, ChatID oder Nachricht fehlt.\n";
    exit(1);
}

$bot = $params['bot'];
$chatid = $params['chatid'];
$msg = urldecode($params['msg']);

// Telegram API URL
$url = "https://api.telegram.org/bot$bot/sendMessage";
```

```
// POST-Daten
$data = array(
    'chat_id' => $chatid,
    'text' => $msg
);

// cURL-Request ausführen
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$result = curl_exec($ch);
curl_close($ch);

echo $result . "\n";
?>
```

Telegram Benachrichtigungsdienst mit einer einzigen Bash

Das Pendant zur telegram.php ist hier die telegram.sh - eine Bash, die genau das gleiche macht wie die PHP, ideal zum Scripten und zur Automation.

Ein Benachrichtigungsansatz, der schnell und einfach umzusetzen ist - in nur einer Datei, mit Ausnahme einer Konfiguration, wäre ein Träumchen. Das habe ich mir gedacht und habe das telegram.sh ins Leben gerufen. Die Einstellungen zum Bot und zum Chat, sowie die Nachricht selbst, werden der Bash alle mit Argumenten/Schaltern übergeben, sodass es eine Konfigurationsdatei überflüssig macht.

Bash-typisch wird hier ein Linux vorausgesetzt. Ich persönlich verwende Debian. Zusätzlich wird curl benötigt.

Die zu übergebenden Parameter werden hier erklärt:

• *-b oder --bot* ist der BOT:TOKEN des Botfathers von Telegram,
• *-c oder --chat* ist der Chat, an dem die Nachricht gehen soll,
• *-m oder --msg* ist die Nachricht, die übergeben wird.

Beispielaufruf:

```
./telegram.sh -b 123456789:ABCdefGhIjKlMnOpQrStUvWxYz -c 987654321 -m "Hallo ich bins"
```

Hier das Script:

```
#!/bin/bash

# Telegram Send Script - CLI mit Parameterübergabe

# Usage info
usage() {
    echo "Nutzung: $0 -b BOT -c CHATID -m MESSAGE"
    exit 1
```

```
}

# Parameter parsen
while [[ $# -gt 0 ]]; do
    key="$1"
    case $key in
        -b|--bot)
        BOT="$2"
        shift; shift
        ;;
        -c|--chat)
        CHATID="$2"
        shift; shift
        ;;
        -m|--msg)
        MSG="$2"
        shift; shift
        ;;
        *)
        usage
        ;;
    esac
done

# Prüfung
if [ -z "$BOT" ] || [ -z "$CHATID" ] || [ -z "$MSG" ]; then
    usage
fi

# Telegram API URL
URL="https://api.telegram.org/bot${BOT}/sendMessage"

# Senden
curl -s -X POST "$URL" -d chat_id="$CHATID" -d text="$MSG"

# Ausgabe-Status
echo
```

Proxmox - Virtuelle Festplatte vergrößern und in der VM nutzbar machen (LVM + ext4)

Kurzablauf:
Virtuelle Platte in Proxmox vergrößern → Partition erweitern → LVM erweitern → Dateisystem erweitern.

Schritte

1. **Partition vergrößern (nach Proxmox-Resize):**

   `growpart /dev/sda 3`
2. **LVM-Physical Volume anpassen:**

   `pvresize /dev/sda3`
3. **LVM-Volume vergrößern (gesamten freien Platz nutzen):**

   `lvextend -l +100%FREE /dev/pbs/root`
4. **Dateisystem ext4 erweitern:**

   `resize2fs /dev/pbs/root`
5. **Erfolg prüfen:**

   `df -h`

Voraussetzungen

- Festplatte wurde vorher in Proxmox über „Disk Resize“ vergrößert.
- Tools installiert:

  `apt install cloud-guest-utils parted`

Ergebnis

Das Root-Dateisystem wächst ohne Neustart und der zusätzliche Speicher steht sofort zur Verfügung.

webcamloader-timezones

Die Webcamloader-Timezones sind hier zur Eingabe mit `-T`:

Script-Erwartung	IANA-Norm-Timezone zum Vergleich / Muster
adelaide	Australia/Adelaide
ae	Asia/Dubai
aegypten	Africa/Cairo
alabama	America/Chicago
alaska	America/Anchorage
alberta	America/Edmonton
aleutian	America/Adak
american-samoa	Pacific/Pago_Pago
amsamoa	Pacific/Pago_Pago
anchorage	America/Anchorage
ar	America/Argentina/Buenos_Aires
argentina	America/Argentina/Buenos_Aires
argentinien	America/Argentina/Buenos_Aires
arizona	America/Phoenix
arkansas	America/Chicago
at	Europe/Vienna

au-act	Australia/Sydney
au-eucla	Australia/Eucla
au-lord-howe	Australia/Lord_Howe
au-nsw	Australia/Sydney
au-nt	Australia/Darwin
au-qld	Australia/Brisbane
au-sa	Australia/Adelaide
au-tas	Australia/Hobart
au-vic	Australia/Melbourne
au-wa	Australia/Perth
australian-capital-territory	Australia/Sydney
austria	Europe/Vienna
be	Europe/Brussels
belgien	Europe/Brussels
belgium	Europe/Brussels
boise	America/Boise
brisbane	Australia/Brisbane
british-columbia	America/Vancouver
ca-ab	America/Edmonton
ca-atikokan	America/Atikokan
ca-bc	America/Vancouver
ca-creston	America/Creston
ca-dawson-creek	America/Dawson_Creek

ca-fort-nelson	America/Fort_Nelson
ca-mb	America/Winnipeg
ca-nb	America/Moncton
ca-nl	America/St_Johns
ca-ns	America/Halifax
ca-nt	America/Yellowknife
ca-nu	America/Iqaluit
ca-on	America/Toronto
ca-pe	America/Halifax
ca-qc	America/Toronto
ca-sk	America/Regina
ca-swift-current	America/Swift_Current
ca-yt	America/Whitehorse
california	America/Los_Angeles
canberra	Australia/Sydney
ch	Europe/Zurich
chicago	America/Chicago
chile	America/Santiago
chile	America/Santiago
china	Asia/Shanghai
cl	America/Santiago
cn	Asia/Shanghai
coeur-dalene	America/Los_Angeles

colorado	America/Denver
connecticut	America/New_York
cz	Europe/Prague
czech	Europe/Prague
daenemark	Europe/Copenhagen
darwin	Australia/Darwin
dc	America/New_York
de	Europe/Berlin
delaware	America/New_York
denmark	Europe/Copenhagen
denver	America/Denver
detroit	America/Detroit
deutschland	Europe/Berlin
district-of-columbia	America/New_York
dk	Europe/Copenhagen
edmonton	America/Edmonton
eg	Africa/Cairo
egypt	Africa/Cairo
el-paso	America/Denver
elpaso	America/Denver
emirates	Asia/Dubai
england	Europe/London
es	Europe/Madrid

eucla	Australia/Eucla
fi	Europe/Helsinki
finland	Europe/Helsinki
finnland	Europe/Helsinki
fl-panhandle	America/Chicago
florida	America/New_York
fr	Europe/Paris
france	Europe/Paris
frankreich	Europe/Paris
gb	Europe/London
georgia	America/New_York
germany	Europe/Berlin
gmt	UTC
gmt0	UTC
gr	Europe/Athens
greece	Europe/Athens
griechenland	Europe/Athens
grossbritannien	Europe/London
guam	Pacific/Guam
halifax	America/Halifax
hawaii	Pacific/Honolulu
hk	Asia/Hong_Kong
hobart	Australia/Hobart

holland	Europe/Amsterdam
hongkong	Asia/Hong_Kong
honolulu	Pacific/Honolulu
hu	Europe/Budapest
hungary	Europe/Budapest
idaho	America/Boise
idaho-pacific	America/Los_Angeles
ie	Europe/Dublin
il	Asia/Jerusalem
illinois	America/Chicago
in	Asia/Kolkata
india	Asia/Kolkata
indiana	America/Indiana/Indianapolis
indiana-central	America/Chicago
indianapolis	America/Indiana/Indianapolis
indien	Asia/Kolkata
iowa	America/Chicago
iqaluit	America/Iqaluit
ir	Asia/Tehran
iran	Asia/Tehran
iran	Asia/Tehran
ireland	Europe/Dublin
irkutsk	Asia/Irkutsk

irland	Europe/Dublin
israel	Asia/Jerusalem
israel	Asia/Jerusalem
it	Europe/Rome
italien	Europe/Rome
italy	Europe/Rome
japan	Asia/Tokyo
jekaterinburg	Asia/Yekaterinburg
jp	Asia/Tokyo
kamchatka	Asia/Kamchatka
kamtschatka	Asia/Kamchatka
kansas	America/Chicago
kansas-mountain	America/Denver
kentucky	America/New_York
kentucky-central	America/Chicago
korea	Asia/Seoul
kr	Asia/Seoul
krasnoyarsk	Asia/Krasnoyarsk
lord-howe	Australia/Lord_Howe
los-angeles	America/Los_Angeles
losangeles	America/Los_Angeles
louisiana	America/Chicago
ma	Africa/Casablanca

maine	America/New_York
malheur	America/Boise
manitoba	America/Winnipeg
marokko	Africa/Casablanca
maryland	America/New_York
massachusetts	America/New_York
melbourne	Australia/Melbourne
mexico	America/Mexico_City
mexiko	America/Mexico_City
michigan	America/Detroit
michigan-central	America/Chicago
minnesota	America/Chicago
mississippi	America/Chicago
missouri	America/Chicag
montana	America/Denver
montreal	America/Toronto
morocco	Africa/Casablanca
moscow	Europe/Moscow
moskau	Europe/Moscow
mx	America/Mexico_City
nebraska	America/Chicago
nebraska-mountain	America/Denver
netherlands	Europe/Amsterdam

neuseeland	Pacific/Auckland
nevada	America/Los_Angeles
nevada-mountain	America/Denver
new-brunswick	America/Moncton
new-hampshire	America/New_York
new-jersey	America/New_York
new-mexico	America/Denver
new-south-wales	Australia/Sydney
new-york	America/New_York
newfoundland	America/St_Johns
newmexico	America/Denver
newyork	America/New_York
newzealand	Pacific/Auckland
ng	Africa/Lagos
niederlande	Europe/Amsterdam
nigeria	Africa/Lagos
nl	Europe/Amsterdam
nmi	Pacific/Saipan
no	Europe/Oslo
north-carolina	America/New_York
north-dakota	America/Chicago
north-dakota-mountain	America/Denver
northcarolina	America/New_York

northdakota	America/Chicago
northern-mariana-islands	Pacific/Saipan
northern-territory	Australia/Darwin
northwest-territories	America/Yellowknife
norway	Europe/Oslo
norwegen	Europe/Oslo
nova-scotia	America/Halifax
nunavut	America/Iqaluit
nwt	America/Yellowknife
ny	America/New_York
nz	Pacific/Auckland
ohio	America/New_York
oklahoma	America/Chicago
ontario	America/Toronto
oregon	America/Los_Angeles
oregon-mountain	America/Boise
oesterreich	Europe/Vienna
pago-pago	Pacific/Pago_Pago
pagopago	Pacific/Pago_Pago
pei	America/Halifax
pennsylvania	America/New_York
pensacola	America/Chicago
perth	Australia/Perth

phoenix	America/Phoenix
pl	Europe/Warsaw
poland	Europe/Warsaw
polen	Europe/Warsaw
polska	Europe/Warsaw
portland	America/Los_Angeles
portugal	Europe/Lisbon
prince-edward-island	America/Halifax
pt	Europe/Lisbon
puerto-rico	America/Puerto_Rico
puertorico	America/Puerto_Rico
quebec	America/Toronto
québec	America/Toronto
queensland	Australia/Brisbane
regina	America/Regina
rhode-island	America/New_York
rhodeisland	America/New_York
ru-irk	Asia/Irkutsk
ru-kam	Asia/Kamchatka
ru-kras	Asia/Krasnoyarsk
ru-magadan	Asia/Magadan
ru-msk	Europe/Moscow
ru-novosibirsk	Asia/Novosibirsk

ru-omsk	Asia/Omsk
ru-samara	Europe/Samara
ru-vlad	Asia/Vladivostok
ru-yakutsk	Asia/Yakutsk
ru-ye	Asia/Yekaterinburg
sa	Asia/Riyadh
saipan	Pacific/Saipan
saskatchewan	America/Regina
saudi	Asia/Riyadh
saudi-arabien	Asia/Riyadh
schweden	Europe/Stockholm
schweiz	Europe/Zurich
se	Europe/Stockholm
seattle	America/Los_Angeles
sg	Asia/Singapore
singapore	Asia/Singapore
singapur	Asia/Singapore
sk	Europe/Bratislava
slovakia	Europe/Bratislava
slowakei	Europe/Bratislava
south-australia	Australia/Adelaide
south-carolina	America/New_York
south-dakota	America/Chicago

south-dakota-mountain	America/Denver
southafrica	Africa/Johannesburg
southcarolina	America/New_York
southdakota	America/Chicago
spain	Europe/Madrid
spanien	Europe/Madrid
st-johns	America/St_Johns
st-thomas	America/St_Thomas
stjohns	America/St_Johns
stthomas	America/St_Thomas
südafrika	Africa/Johannesburg
südkorea	Asia/Seoul
sweden	Europe/Stockholm
switzerland	Europe/Zurich
sydney	Australia/Sydney
taiwan	Asia/Taipei
tasmania	Australia/Hobart
tennessee	America/Chicago
tennessee-eastern	America/New_York
texas	America/Chicago
texas-mountain	America/Denver
th	Asia/Bangkok
thailand	Asia/Bangkok

thailand	Asia/Bangkok
toronto	America/Toronto
tr	Europe/Istanbul
tschechien	Europe/Prague
tuerkei	Europe/Istanbul
turkey	Europe/Istanbul
tw	Asia/Taipei
ua	Europe/Kyiv
uae	Asia/Dubai
uk	Europe/London
ukraine	Europe/Kyiv
ungarn	Europe/Budapest
up-ct	America/Chicago
upper-peninsula-ct	America/Chicago
us-virgin-islands	America/St_Thomas
usa-adak	America/Adak
usa-ak	America/Anchorage
usa-al	America/Chicago
usa-ar	America/Chicago
usa-as	Pacific/Pago_Pago
usa-az	America/Phoenix
usa-ca	America/Los_Angeles
usa-co	America/Denver

usa-ct	America/New_York
usa-dc	America/New_York
usa-de	America/New_York
usa-fl	America/New_York
usa-fl-ct	America/Chicago
usa-ga	America/New_York
usa-gu	Pacific/Guam
usa-hi	Pacific/Honolulu
usa-ia	America/Chicago
usa-id	America/Boise
usa-id-pt	America/Los_Angeles
usa-il	America/Chicago
usa-in	America/Indiana/Indianapolis
usa-in-ct	America/Chicago
usa-in-knox	America/Indiana/Knox
usa-in-tell-city	America/Indiana/Tell_City
usa-ks	America/Chicago
usa-ks-mt	America/Denver
usa-ky	America/New_York
usa-ky-ct	America/Chicago
usa-ky-louisville	America/Kentucky/Louisville
usa-ky-monticello	America/Kentucky/Monticello
usa-la	America/Chicago

usa-ma	America/New_York
usa-md	America/New_York
usa-me	America/New_York
usa-metlakatla	America/Metlakatla
usa-mi	America/Detroit
usa-mi-ct	America/Chicago
usa-mi-menominee	America/Menominee
usa-mn	America/Chicago
usa-mo	America/Chicag
usa-mp	Pacific/Saipan
usa-ms	America/Chicago
usa-mt	America/Denver
usa-nc	America/New_York
usa-nd	America/Chicago
usa-nd-beulah	America/North_Dakota/Beulah
usa-nd-center	America/North_Dakota/Center
usa-nd-mt	America/Denver
usa-nd-new-salem	America/North_Dakota/New_Salem
usa-ne	America/Chicago
usa-ne-mt	America/Denver
usa-nh	America/New_York
usa-nj	America/New_York
usa-nm	America/Denver

usa-nv	America/Los_Angeles
usa-nv-mt	America/Denver
usa-ny	America/New_York
usa-oh	America/New_York
usa-ok	America/Chicago
usa-or	America/Los_Angeles
usa-or-mt	America/Boise
usa-pa	America/New_York
usa-pr	America/Puerto_Rico
usa-ri	America/New_York
usa-sc	America/New_York
usa-sd	America/Chicago
usa-sd-mt	America/Denver
usa-tn	America/Chicago
usa-tn-et	America/New_York
usa-tx	America/Chicago
usa-tx-mt	America/Denver
usa-ut	America/Denver
usa-va	America/New_York
usa-vi	America/St_Thomas
usa-vt	America/New_York
usa-wa	America/Los_Angeles
usa-wi	America/Chicago

usa-wv	America/New_York
usa-wy	America/Denver
usvi	America/St_Thomas
utah	America/Denver
utc	UTC
utc-0	UTC
utc-1	Etc/GMT+1
utc-10	Etc/GMT+10
utc-11	Etc/GMT+11
utc-12	Etc/GMT+12
utc-2	Etc/GMT+2
utc-3	Etc/GMT+3
utc-4	Etc/GMT+4
utc-5	Etc/GMT+5
utc-6	Etc/GMT+6
utc-7	Etc/GMT+7
utc-8	Etc/GMT+8
utc-9	Etc/GMT+9
utc+0	UTC
utc+1	Etc/GMT-1
utc+10	Etc/GMT-10
utc+11	Etc/GMT-11
utc+12	Etc/GMT-12

utc+2	Etc/GMT-2
utc+3	Etc/GMT-3
utc+4	Etc/GMT-4
utc+5	Etc/GMT-5
utc+6	Etc/GMT-6
utc+7	Etc/GMT-7
utc+8	Etc/GMT-8
utc+9	Etc/GMT-9
vancouver	America/Vancouver
vereinigte-arabische-emirate	Asia/Dubai
vermont	America/New_York
victoria	Australia/Melbourne
vietnam	Asia/Ho_Chi_Minh
vietnam	Asia/Ho_Chi_Minh
virginia	America/New_York
vladivostok	Asia/Vladivostok
vn	Asia/Ho_Chi_Minh
washington	America/Los_Angeles
washington-dc	America/New_York
west-virginia	America/New_York
west-wendover	America/Denver
western-australia	Australia/Perth
westvirginia	America/New_York

westwendover	America/Denver
whitehorse	America/Whitehorse
winnipeg	America/Winnipeg
wisconsin	America/Chicago
wyoming	America/Denver
yekaterinburg	Asia/Yekaterinburg
yukon	America/Whitehorse
za	Africa/Johannesburg

Webcamloader (□□Debian / 08.25)

Das Projekt ist nun unter Webcamloader direkt zu finden.

Versioner □□ (Git Alternative Lite)

Versioner, Debian (+Derivate)
Wiki-Stand: 14.06.2026
Script-Stand: 14.06.2026

Vorwort

Der **Versioner** ist ein Bash-Script zur automatisierten Dateiversionierung auf Linux-Systemen. Er ist als einfache, lokale und robuste Alternative zu Git gedacht, wenn Dateien automatisch überwacht und versioniert werden sollen, ohne dass hierfür ein vollständiges Git-Repository gepflegt werden muss.

Das Script eignet sich besonders für produktive Serverumgebungen, in denen regelmäßig Scripte, Konfigurationsdateien, PHP-Dateien, SQL-Dateien, Webdateien oder ähnliche Arbeitsdateien verändert werden. Ziel ist es, Änderungen nachvollziehbar zu speichern und frühere Versionen bei Bedarf wiederherstellen zu können.

Der Versioner arbeitet ohne Datenbank, ohne Dienst im Hintergrund und ohne komplexe Projektstruktur. Er wird gezielt aufgerufen, beispielsweise per Cron, scannt angegebene Ordner und speichert geänderte Dateien zentral unter:

```
/srv/.versioner
```

Dort entstehen pro überwachte Datei eigene Ablageordner mit Vollversionen, Patches, Indexdateien und Metadaten.

Der Versioner ist ausdrücklich kein vollständiger Ersatz für Git im Entwickler-Sinn. Er ist vielmehr ein praktischer Schutzmechanismus für Serverdateien, Scripte und Konfigurationsbestände.

⚙ Funktionen im Wesentlichen

Der Versioner durchsucht beim Scan die angegebenen Ordner rekursiv nach passenden Dateien. Dabei werden nur definierte Dateitypen berücksichtigt. Temporäre Dateien, Logs, Lockfiles und andere unerwünschte Dateitypen werden ausgeschlossen.

Wird eine Datei zum ersten Mal gefunden, speichert der Versioner eine komprimierte Vollversion. Bei späteren Änderungen wird nach Möglichkeit nur ein Patch gespeichert. Dadurch bleibt die Ablage kleiner und trotzdem nachvollziehbar.

Die wichtigsten Funktionen:

- rekursives Scannen frei angegebener Ordner
- mehrere Include-Ordner pro Aufruf möglich
- mehrere Exclude-Pfade pro Aufruf möglich
- feste Include-Dateitypen im Script
- feste Exclude-Dateitypen im Script
- Speicherung von Vollversionen
- Speicherung von Patches
- automatische Patch-Verifikation
- automatische Umwandlung in Vollversion, falls ein Patch nicht sauber verifizierbar ist
- Erkennung gelöschter Dateien
- Wiederherstellung alter Versionen
- Anzeige gespeicherter Versionen
- Prüfung bestehender Restore-Ketten
- Lockfile-Schutz gegen parallele Ausführung
- Schutz gegen gerade bearbeitete Dateien durch Stabilitätsprüfung

---

☝ Technische Hinweise

Der Versioner ist ein Bash-Script und benötigt eine typische Linux-Umgebung. Entwickelt wurde die Logik für Debian und Debian-Derivate.

Benötigte Programme:

```
bash
find
awk
stat
sha256sum
gzip
diff
patch
```

```
flock
mktemp
cp
```

In normalen Debian-Installationen sind diese Programme in der Regel bereits vorhanden oder über Standardpakete verfügbar.

Das Script arbeitet standardmäßig mit folgendem Basisordner:

```
/srv/.versioner
```

Darin werden die internen Daten abgelegt:

```
/srv/.versioner/files
/srv/.versioner/current
/srv/.versioner/state
/srv/.versioner/versioner.lock
```

Die überwachten Originaldateien bleiben an ihrem eigentlichen Ort. Der Versioner kopiert und versioniert sie nur zusätzlich.

Wichtig: Der Versioner sollte mit einem Benutzer ausgeführt werden, der die zu überwachenden Dateien lesen darf. Für Wiederherstellungen muss der Benutzer zusätzlich Schreibrechte auf die Zieldatei besitzen.

Grundprinzip

Der Versioner arbeitet nicht dauerhaft als Dienst, sondern wird bewusst gestartet.

Typischer Aufruf:

```
./versioner.sh scan -i "/srv/scripte" -i "/var/www/public" -i "/home/mariobeh/script-data" -e "/exec-log/" -e "/logs/" -e "/log/" -e "/cache/" -e "/tmp/"
```

Dabei bedeutet:

```
scan
```

Startet den Scan-Modus.

```
-i "/pfad"
```

Definiert einen Ordner, der rekursiv überwacht werden soll.

```
-e "/name/"
```

Definiert einen Pfadbestandteil, der ausgeschlossen werden soll.

Mehrere `-i` und mehrere `-e` sind erlaubt.

Mindestens ein `-i` ist Pflicht.

Speicherort und interne Struktur

Alle Versionierungsdaten werden unterhalb von:

```
/srv/.versioner
```

gespeichert.

Die wichtigsten Unterordner:

files

```
/srv/.versioner/files
```

Hier liegen die eigentlichen Versionen. Für jede überwachte Originaldatei wird eine parallele Ordnerstruktur angelegt.

Beispiel Originaldatei:

```
/srv/scripte/webcamloader.sh
```

Ablage im Versioner:

```
/srv/.versioner/files/srv/scripte/webcamloader.sh/
```

Darin befinden sich beispielsweise:

```
00001-1780574106.full.gz
00002-1780574166.patch.gz
00003-1780574286.patch.gz
index.tsv
meta
```

current

```
/srv/.versioner/current
```

Hier speichert der Versioner den zuletzt bekannten aktuellen Stand einer Datei. Dieser Stand wird als Vergleichsbasis für neue Patches genutzt.

Beispiel:

```
/srv/.versioner/current/srv/scripte/webcamloader.sh
```

state

```
/srv/.versioner/state
```

Dieser Ordner ist für Statusdaten vorgesehen.

Lockfile

```
/srv/.versioner/versioner.lock
```

Dieses Lockfile verhindert, dass der Versioner mehrfach gleichzeitig läuft.

Unterstützte Dateitypen

Der Versioner versioniert nicht pauschal jede Datei. Das ist bewusst so, damit keine Binärdateien, Logs, Cache-Dateien oder große Zufallsdaten in der Versionierung landen.

Berücksichtigt werden unter anderem:

```
*.sh
*.bash
*.zsh
*.py
*.php
*.js
*.css
*.html
*.htm
*.conf
*.cfg
*.ini
*.service
*.timer
*.txt
*.sql
*.json
*.xml
*.yml
*.yaml
*.md
```

Diese Include-Liste ist fest im Script definiert.

Ausgeschlossene Dateitypen

Bestimmte Dateitypen werden unabhängig vom Ordner ausgeschlossen.

Dazu gehören:

```
*.md5
*.state
*.pending
*.pid
*.lock
*.tmp
*.log
*.csv
```

```
cronlog.txt
```

Diese Dateien werden nicht versioniert.

Der Hintergrund ist einfach: Solche Dateien ändern sich häufig automatisch, enthalten oft Laufzeitdaten oder sind für eine Wiederherstellung als Quellversion nicht sinnvoll.

Scan-Modus

Der Scan-Modus ist die Hauptfunktion des Versioners.

Aufruf:

```
./versioner.sh scan -i DIR [-i DIR ...] [-e /DIR/ ...]
```

Beispiel:

```
./versioner.sh scan \
  -i "/srv/scripte" \
  -i "/var/www/public" \
  -i "/home/mariobeh/script-data" \
  -e "/exec-log/" \
  -e "/logs/" \
  -e "/log/" \
  -e "/cache/" \
  -e "/tmp/"
```

Der Versioner durchsucht alle mit `-i` angegebenen Ordner rekursiv.

Dabei gilt:

- nicht vorhandene Include-Ordner werden übersprungen
- nur lesbare Dateien werden berücksichtigt
- ausgeschlossene Dateitypen werden übersprungen
- ausgeschlossene Pfadbestandteile werden übersprungen
- nur definierte Include-Dateitypen werden verarbeitet

Include-Ordner mit -i

Mit `-i` wird ein Startordner angegeben.

Beispiel:

```
-i "/srv/scripte"
```

Mehrere Include-Ordner sind möglich:

```
./versioner.sh scan \
  -i "/srv/scripte" \
  -i "/var/www/public" \
  -i "/home/USER/script-data"
```

Mindestens ein `-i` ist Pflicht. Ohne Include-Ordner bricht der Versioner ab.

Beispiel falscher Aufruf:

```
./versioner.sh scan
```

Ergebnis:

```
Fehler: scan benötigt mindestens ein -i DIR
```

Exclude-Pfade mit -e

Mit `-e` werden Pfadbestandteile ausgeschlossen.

Beispiel:

```
-e "/cache/"
```

Das bedeutet: Sobald im Dateipfad der Bestandteil `/cache/` vorkommt, wird die Datei nicht versioniert.

Beispiele, die ausgeschlossen werden:

```
/var/www/public/cache/index.php
/var/www/public/projekt/cache/data.json
/srv/scripte/test/tmp/datei.sh
```

Beispiele, die nicht ausgeschlossen werden:

```
/var/www/public/cachebuster/index.php
/var/www/public/my-cache/index.php
/srv/scripte/blog/index.php
```

Die Slashes vorne und hinten sind wichtig, damit nur echte Pfadsegmente erkannt werden.

Empfohlene Excludes:

```
-e "/exec-log/"
-e "/logs/"
-e "/log/"
-e "/cache/"
-e "/tmp/"
```

Warum keine Standardordner?

Der Versioner soll bewusst keine festen Standardordner mehr im Script haben.

Das hat mehrere Vorteile:

- der Cronjob ist selbsterklärend
- die überwachten Ordner sind direkt im Aufruf sichtbar
- verschiedene Server können unterschiedliche Ordner nutzen
- ein versehentlicher Scan falscher Bereiche wird verhindert
- das Script bleibt allgemein verwendbar

Der Aufruf definiert vollständig, was überwacht wird.

Ablauf beim Scan

Beim Scan passiert grob Folgendes:

1. Der Versioner setzt ein Lockfile.
2. Die angegebenen Include-Ordner werden durchsucht.
3. Pro Kandidat werden Include- und Exclude-Regeln geprüft.
4. Von jeder gefundenen Datei werden Größe und Änderungszeit gespeichert.
5. Der Versioner wartet kurz.
6. Danach werden Größe und Änderungszeit erneut geprüft.

7. Nur stabile Dateien werden verarbeitet.
8. Neue Dateien werden als Vollversion gespeichert.
9. Geänderte Dateien werden als Patch oder Vollversion gespeichert.
10. Gelöschte Dateien werden im Index als gelöscht markiert.

Der kurze Warteabgleich ist wichtig, damit Dateien nicht mitten während einer Bearbeitung versioniert werden.

Schutz vor unfertigen Dateien

Ein zentrales Ziel des Versioners ist es, keine Dateien zu sichern, die gerade geschrieben oder bearbeitet werden.

Deshalb wird beim Scan eine Stabilitätsprüfung gemacht:

- Vor dem kurzen Warten wird Größe und Änderungszeit erfasst.
- Nach dem Warten wird beides erneut geprüft.
- Wenn sich etwas geändert hat, wird die Datei übersprungen.

Beispielmeldung:

```
/srv/scripte/test.sh → übersprungen, Datei änderte sich gerade
```

Zusätzlich erstellt der Versioner beim Speichern einen Snapshot per `cp -a`. Danach wird nochmals geprüft, ob Original und Snapshot identisch sind. Wenn nicht, wird die Datei ebenfalls übersprungen.

Vollversionen

Wenn eine Datei zum ersten Mal versioniert wird, speichert der Versioner eine Vollversion.

Beispiel:

```
00001-1780574106.full.gz
```

Eine Vollversion enthält den kompletten Dateiinhalt in gzip-komprimierter Form.

Vollversionen entstehen unter anderem bei:

- initialer Speicherung

- leerer Datei
- zu langer Patch-Kette
- zu großem Patch
- fehlgeschlagener Patch-Verifikation

Patches

Bei späteren Änderungen versucht der Versioner, nur einen Patch zu speichern.

Beispiel:

```
00002-1780574166.patch.gz
```

Ein Patch enthält nur die Unterschiede zwischen der zuletzt bekannten aktuellen Version und dem neuen Stand.

Das spart Speicherplatz und reicht für typische Script- oder Konfigurationsänderungen vollständig aus.

Patch-Ketten

Der Versioner begrenzt Patch-Ketten.

Standardwert:

```
MAX_PATCH_CHAIN=100
```

Nach 100 Patches wird wieder eine Vollversion gespeichert.

Der Grund: Sehr lange Patch-Ketten machen Wiederherstellungen aufwendiger und fehleranfälliger. Regelmäßige Vollversionen halten das System robust.

Maximale Patch-Größe

Der Versioner prüft, ob ein Patch im Verhältnis zur Quelldatei zu groß ist.

Standardwert:

```
MAX_PATCH_PERCENT=50
```

Wenn ein Patch größer als 50 Prozent der Quelldatei ist, wird stattdessen eine neue Vollversion gespeichert.

Beispielgrund im Index:

```
patch-too-large-78pct
```

Das verhindert unsinnige Patch-Dateien, wenn sich eine Datei stark geändert hat.

Indexdatei

Jede versionierte Datei erhält eine eigene Indexdatei:

```
index.tsv
```

Diese Datei enthält alle Versionen tabellarisch getrennt mit Tabs.

Typische Informationen:

- Versionsnummer
- Unix-Zeitstempel
- Typ
- Hash
- Dateirechte
- UID
- GID
- Dateigröße
- Artefaktdatei
- Grund

Beispiel sinngemäß:

```
00001  1780574106  full   HASH  755  1000  1000  12345  00001-1780574106.full.gz   initial
00002  1780574166  patch  HASH  755  1000  1000  12410  00002-1780574166.patch.gz  patch
```

Die Indexdatei ist die zentrale Grundlage für Anzeige, Restore und Verify.

Metadatei

Neben dem Index gibt es eine Metadatei:

```
meta
```

Darin steht unter anderem der ursprüngliche Pfad der Datei.

Beispiel:

```
path=/srv/scripte/webcamloader.sh
created=2026-06-14 03:20:00
```

Diese Information wird genutzt, um versionierte Dateien später wieder eindeutig zuordnen zu können.

Versionen anzeigen

Eine Datei kann ohne Unterbefehl abgefragt werden.

Aufruf:

```
./versioner.sh DATEI
```

Beispiel:

```
./versioner.sh webcamloader.sh
```

Dann zeigt der Versioner die letzten 5 Versionen dieser Datei.

Für mehr Versionen:

```
./versioner.sh webcamloader.sh 20
```

Dann werden die letzten 20 Versionen angezeigt.

Die Ausgabe enthält unter anderem:

- Versionsnummer
- Datum und Uhrzeit

- Typ
- Hash
- Dateigröße
- Grund

Beispiel:

```
v1 | 2026-06-14 03:10:00 | full  | HASH | 12345 Bytes | initial
v2 | 2026-06-14 03:15:00 | patch | HASH | 12410 Bytes | patch
```

Datei suchen

Bei der Anzeige oder Wiederherstellung kann entweder ein vollständiger Pfad oder ein Dateiname verwendet werden.

Beispiel mit Dateiname:

```
./versioner.sh webcamloader.sh
```

Beispiel mit vollständigem Pfad:

```
./versioner.sh /srv/scripte/webcamloader.sh
```

Wenn ein Dateiname mehrfach vorkommt, meldet der Versioner eine Mehrdeutigkeit und fordert den vollständigen Pfad.

Das ist wichtig, damit nicht versehentlich die falsche Datei wiederhergestellt wird.

Restore: Version wiederherstellen

Mit `restore` kann eine alte Version wiederhergestellt werden.

Aufruf:

```
./versioner.sh restore DATEI VERSION
```

Beispiel:

```
./versioner.sh restore webcamloader.sh 173
```

Vor der Wiederherstellung legt der Versioner ein Backup der aktuellen Datei an.

Beispiel:

```
webcamloader.sh.versioner-backup-20260614-032000
```

Danach wird die gewünschte Version rekonstruiert und an den Originalpfad geschrieben.

Dabei versucht der Versioner, ursprüngliche Rechte und Eigentümer wiederherzustellen:

```
chmod
chown
```

Falls `chown` mangels Rechte nicht möglich ist, wird dies ignoriert. Das Script bricht deswegen nicht ab.

Wiederherstellung gelöschter Zustände

Wenn eine Datei gelöscht wurde, markiert der Versioner dies im Index als:

```
deleted
```

Wird eine solche gelöschte Version wiederhergestellt, entfernt der Versioner die aktuelle Datei, falls sie existiert.

Auch hier wird vorher ein Backup erstellt, wenn die Datei aktuell vorhanden ist.

Verify: Restore-Ketten prüfen

Mit `verify` können alle aktuellen Restore-Ketten geprüft werden.

Aufruf:

```
./versioner.sh verify
```

Dabei versucht der Versioner, die jeweils letzte bekannte Version einer Datei aus den gespeicherten Vollversionen und Patches zu rekonstruieren. Anschließend wird das Ergebnis mit der aktuellen Originaldatei verglichen.

Wenn alles passt, passiert nichts weiter.

Wenn ein Patch defekt oder nicht sauber rekonstruierbar ist, reagiert der Versioner automatisch:

- bei defektem Patch wird der letzte Patch durch eine Vollversion ersetzt
- bei anderen Problemen wird eine Reparatur-Vollversion gespeichert

Am Ende erscheint:

```
Verify abgeschlossen
```

Gelöschte Dateien

Der Versioner erkennt gelöschte Dateien anhand vorhandener Metadaten.

Wenn eine Datei früher versioniert wurde, aber beim Scan nicht mehr existiert, wird im Index eine neue Version mit Typ `deleted` eingetragen.

Beispiel:

```
00008 1780575000 deleted - - - - 0 - deleted
```

Dadurch bleibt nachvollziehbar, wann eine Datei verschwunden ist.

Cron-Nutzung

Der Versioner ist besonders für Cron geeignet.

Beispiel für stündliche Ausführung:

```
0 * * * * /srv/scripte/versioner.sh scan -i "/srv/scripte" -i "/var/www/public" -i "/home/mariobeh/script-data" -e "/exec-log/" -e "/logs/" -e "/log/" -e "/cache/" -e "/tmp/" >> /var/log/versioner.log 2>&1
```

Beispiel für Ausführung alle 5 Minuten:

```
*/5 * * * * /srv/scripte/versioner.sh scan -i "/srv/scripte" -e "/tmp/" -e "/cache/" >> /var/log/versioner.log 2>&1
```

Der Lockfile-Schutz verhindert parallele Läufe. Wenn ein Lauf noch aktiv ist, bricht der nächste Lauf ab.

Beispielaufrufe

Einen Ordner scannen

```
./versioner.sh scan -i "/srv/scripte"
```

Mehrere Ordner scannen

```
./versioner.sh scan \
  -i "/srv/scripte" \
  -i "/var/www/public" \
  -i "/home/USER/script-data"
```

Mehrere Ordner mit Ausschlüssen scannen

```
./versioner.sh scan \
  -i "/srv/scripte" \
  -i "/var/www/public" \
  -i "/home/USER/script-data" \
  -e "/exec-log/" \
  -e "/logs/" \
  -e "/log/" \
  -e "/cache/" \
  -e "/tmp/"
```

Versionen einer Datei anzeigen

```
./versioner.sh webcamloader.sh
```

Mehr Versionen anzeigen

```
./versioner.sh webcamloader.sh 20
```

Alte Version wiederherstellen

```
./versioner.sh restore webcamloader.sh 173
```

Alle Restore-Ketten prüfen

```
./versioner.sh verify
```

Typische Einsatzbereiche

Der Versioner eignet sich besonders für:

- Bash-Scripte
- PHP-Projekte
- Konfigurationsdateien
- systemd-Units
- SQL-Dateien
- JSON/YAML/XML-Konfigurationen
- Markdown-Dokumentation
- Webdateien
- kleine bis mittelgroße Projektverzeichnisse

Weniger geeignet ist der Versioner für:

- große Binärdateien
- Mediendateien
- Datenbank-Dateien
- Cache-Verzeichnisse
- Log-Verzeichnisse
- Build-Artefakte
- node_modules

- vendor-Verzeichnisse, falls sehr groß

Solche Verzeichnisse sollten über `-e` ausgeschlossen werden.

Versioner vs. Git

Der Versioner ist keine vollständige Git-Alternative im Entwickler-Sinn.

Git bietet:

- Branches
- Commits mit Nachrichten
- Remote-Repositories
- Merge-Logik
- Zusammenarbeit mehrerer Personen
- Historienanalyse
- Staging
- Tags
- Pull/Push

Der Versioner bietet stattdessen:

- automatische Sicherung
- einfache Wiederherstellung
- keine Projektinitialisierung
- keine Git-Kenntnisse nötig
- keine manuelle Commit-Pflege
- einfache Nutzung per Cron
- zentrale Ablage auf dem Server

Der Versioner ist also weniger ein Entwicklerwerkzeug, sondern eher eine automatische Datei-Zeitmaschine für Serverdateien.

Sicherheit und Grenzen

Der Versioner schützt nicht vor allem.

Er schützt gut gegen:

- versehentliches Überschreiben

- versehentliche Scriptfehler
- kaputte Änderungen
- gelöschte Dateien
- unbemerkte kleine Änderungen
- fehlende manuelle Backups vor Bearbeitungen

Er schützt nicht vollständig gegen:

- Festplattenausfall
- Komplettverlust des Servers
- Angreifer mit Root-Zugriff
- mutwilliges Löschen von `/srv/.versioner`
- Datenbankverlust
- große Binärdatenänderungen
- fehlerhafte Systemzustände außerhalb der versionierten Dateien

Der Versioner ersetzt daher kein echtes Backup.

Empfehlung: `/srv/.versioner` selbst sollte regelmäßig extern gesichert werden.

Rechte und Eigentümer

Der Versioner speichert beim Versionieren:

- Dateimodus
- UID
- GID
- Dateigröße
- Hash

Beim Restore versucht er, Modus und Eigentümer wiederherzustellen.

Das ist besonders nützlich bei Scripten und Konfigurationsdateien.

Beispiel:

```
chmod 755 script.sh
chown user:gruppe script.sh
```

Wenn der Restore nicht als Root läuft, kann `chown` fehlschlagen. Das wird vom Script toleriert.

Hash-Prüfung

Jede Version wird mit einem SHA256-Hash gespeichert.

Beim Wiederaufbau einer Version wird der Hash geprüft. Wenn der rekonstruierte Inhalt nicht zum erwarteten Hash passt, bricht die Wiederherstellung ab.

Dadurch wird verhindert, dass beschädigte Patch-Ketten unbemerkt falsche Dateien erzeugen.

Warum Patches geprüft werden

Nach dem Speichern eines Patches baut der Versioner die neue Version testweise wieder zusammen. Anschließend wird geprüft, ob der rekonstruierte Inhalt dem Snapshot entspricht.

Wenn die Prüfung erfolgreich ist, wird der Patch akzeptiert.

Wenn die Prüfung fehlschlägt, wird der Patch verworfen und durch eine Vollversion ersetzt.

Das macht den Versioner robuster, weil fehlerhafte Patches nicht stillschweigend in der Historie bleiben.

Empfohlener Produktivaufruf

Für eine typische Serverumgebung:

```
/srv/scripte/versioner.sh scan \
  -i "/srv/scripte" \
  -i "/var/www/public" \
  -i "/home/USER/script-data" \
  -e "/exec-log/" \
  -e "/logs/" \
  -e "/log/" \
  -e "/cache/" \
  -e "/tmp/"
```

Als Cronjob:

```
* * * * * /srv/scripte/versioner.sh scan -i "/srv/scripte" -i "/var/www/public" -i "/home/USER/script-data" -e "/exec-log/" -e "/logs/" -e "/log/" -e "/cache/" -e "/tmp/" >> /var/log/versioner.log 2>&1
```

Hinweise zur Ordnerauswahl

Nicht jeder Ordner sollte pauschal versioniert werden.

Sinnvoll:

```
/srv/scripte
/var/www/public
/home/USER/script-data
/etc
```

Mit Vorsicht:

```
/var/www
/home
/etc
```

Nicht sinnvoll:

```
/var/log
/tmp
/var/cache
/var/lib/mysql
/node_modules
/vendor
```

Je größer und dynamischer der Ordner, desto eher sollte er ausgeschlossen oder gar nicht erst als Include-Ordner gesetzt werden.

Fehlerbeispiele

Scan ohne -i

```
./versioner.sh scan
```

Fehler:

```
Fehler: scan benötigt mindestens ein -i DIR
```

Unbekannter Scan-Parameter

```
./versioner.sh scan -x test
```

Fehler:

```
Fehler: Unbekannter scan-Parameter: -x
```

Restore mit unklarem Dateinamen

Wenn mehrere Dateien denselben Namen haben:

```
./versioner.sh config.php
```

Dann meldet der Versioner eine Mehrdeutigkeit und verlangt den vollständigen Pfad.

Wartung

Der Versioner wächst mit der Anzahl der Änderungen.

Mögliche Wartungsmaßnahmen:

- gelegentlich Speicherplatz prüfen
- `/srv/.versioner` in Backups einbeziehen
- alte Versionen bei Bedarf manuell archivieren
- sehr große oder dynamische Ordner nicht scannen
- Exclude-Liste im Cron-Aufruf sauber pflegen
- regelmäßig `verify` ausführen

Beispiel:

```
./versioner.sh verify
```

Bekannte Grenzen

Der Versioner arbeitet dateibasiert. Er kennt keine semantischen Änderungen und keine Projektzusammenhänge wie Git.

Wenn zehn Dateien zusammen geändert wurden, weiß der Versioner nicht, dass diese Änderung logisch zusammengehört. Jede Datei hat ihre eigene Historie.

Auch gibt es keine Commit-Nachrichten. Der Grund einer Version ist technisch, beispielsweise:

```
initial
patch
patch-chain-limit
patch-too-large-80pct
deleted
```

Der Versioner ist bewusst einfach gehalten.

⚠ Haftungsausschluss

Der Versioner ist ein Hilfswerkzeug zur lokalen Dateiversionierung.

Er ersetzt kein vollständiges Backupkonzept.

Ich übernehme keine Haftung für:

- Datenverlust
- fehlerhafte Bedienung
- beschädigte Dateien
- falsche Restore-Versionen
- fehlende Backups
- falsch gesetzte Include- oder Exclude-Pfade
- zu große oder ungeeignete Verzeichnisbereiche
- Schäden durch produktiven Einsatz ohne vorherigen Test

Vor produktivem Einsatz sollte das Script in einer ungefährlichen Umgebung getestet werden.

Besonders Restore-Vorgänge sollten vorab verstanden und geprüft werden.

Script Download

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

DOWNLOAD (via mariobeh.de)

Das Script sollte zentral abgelegt werden, zum Beispiel:

```
/srv/scripte/versioner.sh
```

Ausführbar machen:

```
chmod +x /srv/scripte/versioner.sh
```

Testaufruf:

```
/srv/scripte/versioner.sh help
```

Produktivaufruf:

```
/srv/scripte/versioner.sh scan \
  -i "/srv/scripte" \
  -i "/var/www/public" \
  -i "/home/USER/script-data" \
  -e "/exec-log/" \
  -e "/logs/" \
  -e "/log/" \
  -e "/cache/" \
  -e "/tmp/"
```

Nur in Deutsch verfügbar.

Vielen Dank,
mariobeh.