Photos aus der iCloud zu Immich

- iCloud to Immich
- Erhalt von HDR (HLG / BT.2020) bei Konvertierung mit FFmpeg

iCloud to Immich

Hier beschreibe ich, wie man seine Videos, die man vielleicht aus Apples Fotos.app exportiert hat, sorgenfrei mit korrekten Metadaten wie GPS Position und HDR mit Hile von FFmpeg von .mov in .mp4 umwandeln kann.

Das Umwandeln der Videos hat mich ein wenig Zeit gekostet. Erstens, weil ich erst etwas später erkannt habe, dass mein einfacher `ffmpeg` Befehl, nicht alle erforderlichen Exif- und Metadaten der `.mov`-Dateien mitkopiert hat.

Exportieren aus Photos.app

Das gewünschte Video oder das ganze Album auswählen und exportieren. Im Dialog die Haken bei Metadata und Ortsangaben (Location) setzen und einen Ort zum Speichern auswählen. Warten.

Umwandeln der Videos von .mov zu .mp4

Mit ein wenig Hilfe der Dokumentation, fish und der KI habe ich es dann hinbekommen, dass alle Videos aus `.mov` zu `.mp4` umgewandelt wurden, mit `h.246` kodiert und die Exif- sowie Metadaten übertragen werden. Dazu wird zusätzlich zu `ffmpeg` noch das `exiftool` benötigt. Dieses hat nicht nur bei der Überprüfung sondern auch nach der Umwandlung geholfen, die korrekten Exifdaten einzutragen, bzw. zu übertragen.

Das shell-Skript für macOS und Linux (`convert-videos.sh`)

```
#!/bin/bash

# 1. Backup-Ordner erstellen
mkdir -p original_movs

# 2. Schleife durch alle .mov Dateien (Groß-/Kleinschreibung beachten)
for f in *.[mM][oO][vV]; do
    # Prüfen, ob Dateien existieren
    [ -e "$f" ] || continue

    filename="${f%.*}"
    echo -e "\n\033[0;36m>>> Verarbeite: $f\033[0m"
```

```
    # 3. FFmpeg Konvertierung
    # -map 0:v -map 0:a extrahiert nur Video und Audio (verhindert
'mebx' Fehler)
    # -map_metadata 0 übernimmt die globalen Tags (GPS, Modell)
    ffmpeg -i "$f" -map 0:v -map 0:a -map_metadata 0 -c:v libx264 -crf
23 -preset medium -c:a copy -movflags use_metadata_tags
"${filename}.mp4"

    # 4. ExifTool Metadaten-Synchronisation
    # Schreibt das 2023er Datum in die MP4-Header
    echo -e "\033[0;33mSynchronisiere Datum für Immich...\033[0m"
    exiftool -overwrite_original -P "-CreateDate<CreationDate" "-
ModifyDate<CreationDate" "-Track*Date<CreationDate" "-
Media*Date<CreationDate" "${filename}.mp4"

    # 5. Original verschieben
    mv "$f" original_movs/
done

echo -e "\n\033[0;35mAlle Dateien fertig! Originale liegen in
'original_movs'.\033[0m"
```

1. **Datei erstellen**: Erstelle eine Datei namens `convert-videos.sh` im Ordner deiner Videos.
2. **Berechtigungen setzen**: Du musst das Skript ausführbar machen. Öffne das Terminal im Ordner und gib ein: `chmod +x convert-videos.sh`
3. **Skript starten**: `./convert-videos.sh`

Das PowerShell-Skript (`konvertiere-videos.ps1`)

```
# 1. Backup-Ordner erstellen, falls er noch nicht existiert
if (!(Test-Path "original_movs")) { New-Item -ItemType Directory -Name
"original_movs" }

# 2. Alle .mov Dateien im aktuellen Verzeichnis suchen
$dateien = Get-ChildItem -Filter *.mov

Write-Host "Gefunden: $($dateien.Count) Dateien zum Verarbeiten." -
ForegroundColor White

foreach ($datei in $dateien) {
    $ausgabeDatei = $datei.BaseName + ".mp4"
    Write-Host "`n>>> Verarbeite: $($datei.Name)" -ForegroundColor Cyan
```

```
    # 3. FFmpeg mit sicherem Mapping
    # -map 0:v (alle Videos), -map 0:a (alle Audios)
    # Wir lassen die inkompatiblen Datenströme (mebx) weg, nehmen aber
die Tags mit
    ffmpeg -i $datei.Name -map 0:v -map 0:a -map_metadata 0 -c:v
libx264 -crf 23 -preset medium -c:a copy -movflags use_metadata_tags
$ausgabeDatei

    # 4. Metadaten-Korrektur (wie gehabt)
    Write-Host "Synchronisiere Datum..." -ForegroundColor Yellow
    exiftool -overwrite_original -P "-CreateDate<CreationDate" "-
ModifyDate<CreationDate" "-Track*Date<CreationDate" "-
Media*Date<CreationDate" $ausgabeDatei

    # 5. Verschieben
    Move-Item $datei.Name "original_movs\"

    Write-Host "Erfolg: $ausgabeDatei ist bereit für Immich." -
ForegroundColor Green
}

Write-Host "`nAlle Dateien fertig! Originale liegen in
'original_movs'." -ForegroundColor Magenta
```

Ausführung unter Windows (mit PowerShell ISE)

Die PowerShell ISE ist auf jedem Windows-System vorinstalliert und ideal, um das Skript schnell und sicher auszuführen.

1. **PowerShell ISE öffnen**:

- Drücke die `Windows-Taste`, tippe **PowerShell ISE** ein und öffne sie.

2. **Skript einfügen**:

- Klicke oben links auf **Datei > Neu** (oder `Strg + N`).
- Kopiere das oben stehende PowerShell-Skript und füge es in das weiße Textfeld ein.

3. **Ordnerpfad festlegen**:

- Stelle sicher, dass du das Skript im gleichen Ordner speicherst, in dem auch deine `.mov` Dateien liegen (**Datei > Speichern unter...**).

4. **Skript starten**:

- Klicke in der Symbolleiste auf den **grünen "Ausführen"-Pfeil** (oder drücke `F5`).
- **Hinweis:** Falls eine Fehlermeldung bezüglich der „Ausführungsrichtlinie“ erscheint, gib einmalig folgenden Befehl in das blaue Fenster (die Konsole) unten ein: `Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope Process`

Erklärungen der einzelnen Befehle

1. Der FFmpeg-Befehl

`ffmpeg -i input.mov -map 0 -map_metadata 0 -c:v libx264 -crf 23 -preset medium -c:a copy -movflags use_metadata_tags output.mp4`

oder

`ffmpeg -i input.mov -map 0:v -map 0:a -map_metadata 0 -c:v libx264 -crf 23 -preset medium -c:a copy -movflags use_metadata_tags output.mp4`

- `-map 0`: Weist FFmpeg an, **alle** Datenströme der Quelldatei (Video, Audio und vor allem die DJI-Metadaten-Streams) in die Zieldatei zu übernehmen.
- oder `-map 0:v`: Weist FFmpeg an, den Video Datenstrom der Quelldatei in die Zieldatei zu übernehmen.
- oder `-map 0:a`: weist FFmpeg an, den Audio Datenstrom der Quelldatei in die Zieldatei zu übernehmen.
- `-map_metadata 0`: Stellt sicher, dass die globalen Metadaten (wie das Erstellungsdatum) eins zu eins kopiert werden.
- `-c:v libx264`: Nutzt den H.264 CPU-Encoder. Dies ist effizienter als GPU-Encoding und spart ca. 50% Speicherplatz bei gleichbleibender Qualität.
- `-crf 23`: "Constant Rate Factor". 23 ist der Standardwert für eine sehr gute Balance zwischen Dateigröße und Bildqualität.
- `-c:a copy`: Kopiert den Audiostream ohne Neukodierung, um Qualität zu erhalten und Zeit zu sparen.
- `-movflags use_metadata_tags`: Erlaubt es FFmpeg, zusätzliche Metadaten-Tags im MP4-Container zu schreiben, die normalerweise nur in MOV-Dateien vorkommen.

2. Der ExifTool-Befehl

`exiftool "-CreateDate<CreationDate" ... output.mp4`

- **Das Problem**: FFmpeg setzt beim Enkodieren oft das "Create Date" des MP4-Containers auf das aktuelle Datum (z.B. 2026).
- **Die Lösung**: Dieser Befehl liest das ursprüngliche `CreationDate` (das echte Flugdatum 2023 aus den QuickTime-Tags) aus und überschreibt damit alle anderen Zeitstempel (`CreateDate`, `ModifyDate`, `TrackDate`).
- **Wichtigkeit für Immich**: Dadurch erkennt Immich das Video korrekt in der Timeline von 2023 und nicht als "heute aufgenommen".
- `-P`: Bewahrt das ursprüngliche Dateimodifikationsdatum im Windows-Dateisystem.

3. Fehlerbehebung bei Apple/iPhone Dateien

Wenn du iPhone-Videos konvertierst, stößt du auf den Fehler `Could not find tag for codec none in stream #2`. Das liegt an Apple-Metadaten-Strömen (`mebx`), die nicht MP4-kompatibel sind.

Lösung im Vergleich:

- **DJI-Videos**: Nutzen oft `-map 0`, weil die GPS-Daten in einem kompatiblen Stream liegen.
- **iPhone-Videos**: Benötigen `-map 0:v -map 0:a`. Dies kopiert nur Video und Audio. Die wichtigen Metadaten (Standort, Erstellungsdatum) werden trotzdem über den Befehl `-map_metadata 0` und `-movflags use_metadata_tags` in den Header des MP4-Containers geschrieben.

Warum `-map 0:v -map 0:a`?

Dieser Befehl filtert die "toten" Datenströme heraus, die den MP4-Container zum Absturz bringen würden, behält aber die volle Videoqualität und alle Tags (GPS, Kamera-Modell), die Immich für die Karte und die Timeline benötigt.

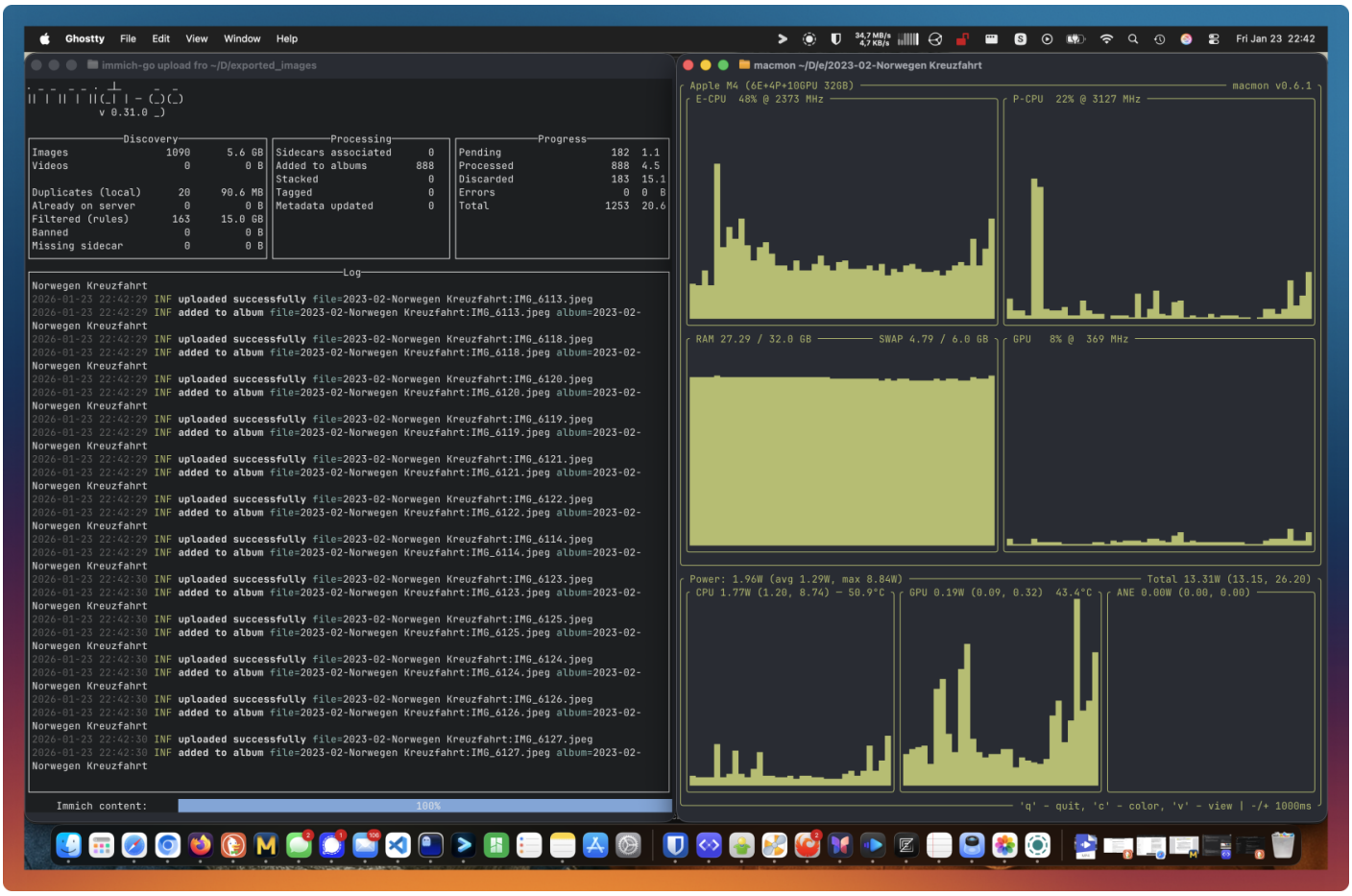
Warum kein GPU-Encoding (NVENC)?

Das Hardware-Encoding via Nvidia (NVENC) ist zwar extrem schnell, aber deutlich weniger effizient. In Tests wurde eine Datei von 63 MB mit dem CPU-Skript auf **27 MB** geschrumpft, während NVENC nur auf **55 MB** kam. Für die Langzeitarchivierung in Immich ist die CPU-Variante daher die bessere Wahl.

Zudem bietet der CPU-Encoder libx264 eine stabilere Beibehaltung von HDR-Flags (HLG), was bei GPU-Encodern je nach Treiberversion manchmal zu Problemen führen kann.

Hochladen der Videos zu Immich mit dem `immich-go` Kommandozeilenprogramm

`immich-go`



Erhalt von HDR (HLG / BT.2020) bei Konvertierung mit FFmpeg

Dein Smartphone, Drohne oder Kamera nimmt Videos im **HLG-Format** (Hybrid Log-Gamma) auf. Das ist ein HDR-Standard, der einen deutlich größeren Farbraum (**BT.2020**) und einen höheren Dynamikumfang nutzt als herkömmliche Videos (**BT.709**).

Im Gegensatz zu vielen einfachen Online-Konvertern oder Standard-Tools erkennt unser FFmpeg-Befehl die HDR-Flags automatisch:

1. **Farbraum-Erhalt**: Durch den Verzicht auf Filter (wie `format=yuv420p`) erkennt FFmpeg, dass die Quelle 10-Bit-Farben (HLG) hat, und versucht, diese Metadaten in den neuen H.264-Stream zu übertragen.
2. **Vermeidung von "Washed Out"-Effekten**: Ein häufiger Fehler ist das sog. "Tone Mapping" auf SDR. Da wir die Metadaten-Tags (`-movflags use_metadata_tags`) kopieren, weiß dein Abspielgerät (oder Immich), dass es die Helligkeit für HDR-Displays hochfahren muss.

Nach der Konvertierung zeigt `exiftool` diese entscheidenden Zeilen:

- `Color Primaries: BT.2020, BT.2100`
- `Transfer Characteristics: BT.2100 HLG`

> "Sollten die Farben in Immich im Webbrowser blass aussehen, liegt das meist am Browser oder dem Monitor, der kein HDR unterstützt. Immich behält die Originaldatei jedoch im vollen HDR-Glanz bei, sodass sie auf HDR-TVs oder Smartphones mit hoher Helligkeit korrekt angezeigt wird."

Zusammenfassung (TL;DR)

Feature	Methode im Skript	Vorteil
Größe	`libx264` (CPU)	~50% Ersparnis gegenüber Original / NVENC.

Feature	Methode im Skript	Vorteil
Farben	HLG / BT.2020 Erhalt	Voller HDR-Kontrast bleibt erhalten.
Ort	`-map 0`	GPS-Koordinaten wandern mit ins MP4.
Datum	`ExifTool` Sync	Korrekte Sortierung in der Immich-Timeline (2023 statt 2026).