Automatiser la conversion d’un dossier video

Par Rémi Albertucci

Table des matières

I Intro

Contexte

Les fichiers vidéo sont souvent encodés dans des formats peu optimisés pour le stockage : h264 en haute résolution, gros bitrate, fichiers de plusieurs gigaoctets…

Un simple rip DVD en vu de l'ajouter à votre serveur Jellyfin, ou une video tournée à la GoPro suffisent à prendre plusieurs Go de votre disque dur.

L'idée ici est de les recompresser automatiquement en HEVC (h265), qui offre une compression nettement meilleure à qualité équivalente : on divise souvent le poids par deux sans différence visible à l'écran.

Pour ça, on va utiliser ffmpeg avec accélération matérielle via un GPU Nvidia, ce qui permet de traiter les fichiers rapidement sans surcharger le CPU.

Le tout sera automatisé pour tourner en arrière-plan sur une ou deux machines, avec les vidéos stockées sur un NAS partagé.

Prérequis

On travaille sur une machine sous Debian 13 (Trixie), avec une GPU Nvidia disposant du support NVENC (ici une Quadro P1000).

Les drivers Nvidia sont à installer sur l'hôte, leur mise en place ne sera pas détaillée ici, mais dans un autre guide présent sur https://remifuret.fr.

Côté logiciels, il faut installer ffmpeg avec le support NVENC, ainsi que mediainfo et jq :

```
apt install ffmpeg mediainfo jq
```

Le dossier vidéo doit être accessible sur la machine. Si vos vidéos sont sur un NAS, vous pouvez monter le partage SMB de façon permanente via `/etc/fstab` :

```
//<IP-distante>/videos ~/videos cifs
credentials=/etc/samba/creds,uid=1000,gid
=1000,iocharset=utf8 0 0
```

Le fichier de credentials contiendra simplement :

```
username=xxx
password=xxx
```

Préparation

Ce script peut tourner sur une machine Debian ou similaire en bare metal, dans une machine virtuelle, ou un conteneur LXC avec un pass GPU.

Dans un soucis de simplicité, je recommande un conteneur LXC Debian 13 : tres peu gourmand en ressrouces car le GPU fait le gros du travail, le LXC a été construit autour d’un coeur processeur et 256Mo de RAM.

De plus, dans une optique de clustering, une VM se « réserve » le GPU et empêche le partage avec une autre VM, sauf GPU récents. Ce n’est pas le cas de la P1000.

II Script

Explications

Le script parcourt un dossier de fichiers vidéo et décide pour chacun s'il doit être converti ou non, avant de lancer la conversion via ffmpeg.

Logique générale

Pour chaque fichier trouvé, le script commence par vérifier s'il a déjà été traité ou s'il a déjà échoué lors d'une exécution précédente — auquel cas il passe au suivant.

Sinon, il analyse le fichier (codec, résolution, poids) et applique les règles suivantes :

- Si le fichier est déjà en HEVC et fait moins de 3 Go, il est considéré comme déjà optimisé
- Si le fichier fait moins de 2,2 Go quelle que soit sa source, il est également ignoré
- Si la résolution ou les dimensions sont illisibles, le fichier est marqué en erreur

Si le fichier passe ces filtres, la conversion est lancée.

Le script encode en HEVC via `hevc_nvenc` avec un bitrate variable. Si la vidéo source dépasse le 1080p, elle est redimensionnée ; en dessous, la résolution d'origine est conservée.

L'audio est converti en AAC 192k stéréo, et les sous-titres sont copiés sans modification.

Cas particulier : les fichiers déjà en AV1 ne peuvent pas être décodés matériellement par la P1000, le script bascule alors automatiquement sur un décodage CPU.

Vérification du résultat

Une fois la conversion terminée, le script compare le poids du fichier converti avec l'original. Si le gain est inférieur à 7% ou si le fichier converti est plus lourd, la conversion est abandonnée et l'original est conservé.

Si le gain est suffisant, l'original est remplacé.

Journal

Trois fichiers de log sont tenus à jour dans `.conversion/films/` : les conversions réussies, les échecs, et les erreurs de format. Cela permet de reprendre le script à tout moment sans retraiter les fichiers déjà gérés.

Variables

Variables générales

En haut du script, quelques valeurs sont à adapter selon votre configuration.

Les chemins des fichiers de log et de verrou sont construits automatiquement à partir de l'emplacement du script lui-même (`SCRIPT_DIR`), vous n'avez donc pas à les modifier.

En revanche, le chemin du dossier vidéo est en dur dans les commandes `find` en bas du script :

```
find ~/videos/films …
```

C'est la première chose à adapter à votre configuration.

Les seuils de poids qui déterminent si un fichier doit être converti sont définis directement dans les conditions du script :

```
3006477107  # 2,8 Go — seuil pour un fichier déjà en HEVC
2362232012  # 2,2 Go — seuil universel en dessous duquel on ignore le fichier
```

Enfin, la limite horaire est fixée à 17h dans cette ligne :

```
if (( heure_actuelle >= 17 )); then
```

Les options ffmpeg

La commande de conversion est la suivante :

```
ffmpeg -nostdin -hide_banner -loglevel error
-stats \
$hw_opts \
-i "${video}" \
${vf_filter:+-vf "$vf_filter"} \
-c:v hevc_nvenc -preset p6 -rc vbr -cq 26
-maxrate 14M -bufsize 28M \
-c:a aac -b:a 192k -ac 2 \
-c:s copy \
"${nom}.tmp.mkv"
```

- Encodeur : `hevc_nvenc` est l'encodeur HEVC matériel Nvidia. Si vous n'avez pas de GPU Nvidia, remplacez-le par `libx265` pour un encodage CPU (nettement plus lent).
- Preset : `-preset p6` correspond à un bon compromis vitesse/qualité sur NVENC (p1 étant le plus rapide, p7 le plus lent et qualitatif).

- Qualité : `-rc vbr -cq 26` active le mode bitrate variable avec un facteur de qualité de 26. Plus la valeur est basse, meilleure est la qualité et plus le fichier sera lourd. Une valeur entre 24 et 32 est raisonnable selon vos besoins.
- Bitrate maximum : `-maxrate 14M -bufsize 28M` plafonne le bitrate à 14 Mb/s pour éviter les pics. À ajuster si vous traitez beaucoup de contenus en 4K.
- Audio : `-c:a aac -b:a 192k -ac 2` convertit tous les flux audio en AAC stéréo 192k. Le `-ac 2` force le downmix en stéréo — à supprimer si vous voulez conserver le son surround.
- Accélération matérielle : `$hw_opts` vaut `-hwaccel cuda -hwaccel_output_format cuda` pour les fichiers non-AV1, et est vide pour les fichiers AV1 que la P1000 ne sait pas décoder matériellement.

Pour permettre à deux machines de travailler en parallèle sur le même dossier sans se marcher dessus, le script utilise un système de verrou basé sur la création de dossiers.

Avant de traiter un fichier, le script tente de créer un dossier `.lock` associé à ce fichier dans `.conversion/.locks/` :

```
function lock_file() {
    local lockfile="$LOCKS_DIR/$(basename
"$1").lock"
    if ! mkdir "$lockfile" 2>/dev/null; then
        return 1
    fi
    return 0
}
```

La création d'un dossier avec `mkdir` est une opération atomique sur la plupart des systèmes de fichiers, y compris SMB : deux machines ne peuvent pas créer le même dossier simultanément.

Si le dossier existe déjà, `mkdir` échoue et le script passe au fichier suivant. Une fois la conversion terminée, le verrou est supprimé :

```
function unlock_file() {
    local lockfile="$LOCKS_DIR/$(basename
"$1").lock"
    rm -rf "$lockfile"
}
```

Ce dossier `.locks/` étant situé sur le partage NAS, il est visible par les deux machines, ce qui rend le système efficace en parallèle.

Un point d'attention : si le script est interrompu brutalement (coupure réseau, redémarrage), les verrous ne sont pas supprimés automatiquement. Il faudra alors vider manuellement le dossier `.locks/` avant de relancer.

III Programmation

Pour que le script tourne automatiquement, on utilise une tâche cron.

L'idée est de le lancer en début d'après-midi et de le laisser tourner quelques heures, le script s'arrêtera de lui-même à 17h.

Ajoutez la ligne suivante avec `crontab -e` :

```
0 13 * * 1 bash ~/scripts/convert-films.sh >> ~/scripts/.conversion/films/cron.log 2>&1
```

Cela lance le script tous les lundis à 13h, et redirige toute la sortie (normale et erreurs) vers un fichier de log dédié.

IV Clustering

Le script est conçu pour tourner en parallèle sur plusieurs machines accédant au même dossier partagé.

Pour éviter que deux machines ne convertissent le même fichier simultanément, il utilise un système de verrou : avant de traiter un fichier, il crée un dossier `.lock` associé à ce fichier dans `.conversion/.locks/`. Si le dossier existe déjà, c'est qu'une autre machine est en train de traiter ce fichier, et le script passe au suivant.

La mise en place sur une deuxième machine est identique à la première : même installation des paquets, même montage du partage SMB, même script déposé au même chemin, même crontab.

Les deux machines partageant le même dossier `.conversion/films/`, elles partagent aussi automatiquement les journaux — une conversion réussie sur l'une ne sera pas retentée par l'autre.

V Conclusion

Ce setup permet de traiter automatiquement une bibliothèque vidéo sans intervention manuelle, avec un gain d'espace significatif.

La logique de reprise et les journaux permettent de l'arrêter et de le relancer sans risque de doublons ou de fichiers corrompus.

C’est un bon point de départ pour qui voudrait l'adapter : changer les seuils pour l’ajuster à d’autres types de contenus, ajouter d'autres formats de sortie, ou étendre le clustering à plus de deux machines.

Annexe : script complet

```
#!/bin/bash

# Script pour convertir les videos avec ffmpeg, en vue de stockage/archivage:
#       _script pour GPU Nvidia avec NVENC (ici p1000 donc decodage si pas av1 et encodage h265)
#       _convertit le flux video en hevc en taille d'origine si =<1080p ; convertit en hevc 1080p si >1080p
#       _framerate comme la source ; bitrate variable
#       _convertit tous les flux audios en aac 192bs s'il y en a
#       _garde tous les sous titres s'il y'en a
#       _s'arrete passe 16h

#############################################################################
# Definition des variables

if [ -t 1 ]; then
    RED=$'\033[0;31m'
    GREEN=$'\033[0;32m'
    ORANGE=$'\033[0;33m'
    BLUE=$'\033[0;34m'
    PURPLE=$'\033[0;35m'
    LIGHT_PURPLE=$'\033[1;35m'
    NC=$'\033[0m'
else
    RED="" GREEN="" ORANGE="" BLUE="" PURPLE="" LIGHT_PURPLE="" NC=""
fi

NB_Format_error=0
Nb_Conversion_successful=0
NB_Conversion_failed=0
nb_analyses=0
nb_total_fichiers=0
total_gain=0
total_gain_go=0
```

```
gain_mo=0
gain=0
gain_percent=0
heure_arret=0

#############################################################################
# Definition des chemins et dossiers*

SCRIPT_DIR="$(dirname "$(realpath "$0")")"

mkdir -p "$SCRIPT_DIR/.conversion" && mkdir -p
"$SCRIPT_DIR/.conversion/films"

CONVERSION_SUCCESSFUL="$SCRIPT_DIR/.conversion/films/converted.txt"
touch "$CONVERSION_SUCCESSFUL"

CONVERSION_FAILED="$SCRIPT_DIR/.conversion/films/failed_to_convert.txt"
touch "$CONVERSION_FAILED"

FORMAT_ERROR="$SCRIPT_DIR/.conversion/films/format_error.txt"
touch "$FORMAT_ERROR"

LOCKS_DIR="$SCRIPT_DIR/.conversion/.locks"
mkdir -p "$LOCKS_DIR"

#############################################################################
# Définition des fonctions

function is_already_treated() {
    grep -Fxq -- "$1" "$CONVERSION_SUCCESSFUL"
}
function mark_as_treated() {
        if ! is_already_treated "$1"; then
                echo "$1" >> "$CONVERSION_SUCCESSFUL"
        fi
}

function has_already_failed() {
```

```
    grep -Fxq -- "$1" "$CONVERSION_FAILED"
}
function mark_as_failed() {
        if ! has_already_failed "$1"; then
                echo "$1" >> "$CONVERSION_FAILED"
        fi
}

function has_error() {
    grep -Fxq -- "$1" "$FORMAT_ERROR"
}
function mark_as_error() {
        if ! has_error "$1"; then
                echo "$1" >> "$FORMAT_ERROR"
        fi
}

function is_number() {
    [[ "$1" =~ ^[0-9]+$ ]]
}

function extract_info {
        ffprobe_output=$(ffprobe -v error -select_streams v:0 -show_entries
stream=width,height,codec_name -show_entries format=size -of json "$video")

    height=$(echo "$ffprobe_output" | jq '.streams[0].height // empty')
    width=$(echo "$ffprobe_output" | jq '.streams[0].width // empty')
    codec=$(echo "$ffprobe_output" | jq -r '.streams[0].codec_name // empty')
    weight=$(echo "$ffprobe_output" | jq '.format.size // empty')

    [[ -z "$height" ]] && height=$(mediainfo --Inform="Video;%Height%" "$video")
    [[ -z "$width" ]] && width=$(mediainfo --Inform="Video;%Width%" "$video")
    [[ -z "$codec" ]] && codec=$(mediainfo --Inform="Video;%CodecID%"
"$video")
    [[ -z "$weight" ]] && weight=$(mediainfo --Inform="General;%FileSize%"
"$video")
}
```

```
function normalize() {
    weight=$(echo "$weight" | tr -d '"\r\n ')
    height=$(echo "$height" | tr -d '"\r\n ')
    width=$(echo "$width" | tr -d '"\r\n ')

    ((weight = weight+0))
    ((height = height+0))
    ((width = width+0))

    height=${height:-0}
    width=${width:-0}
    codec=${codec:-""}
    weight=${weight:-2362232012}
}

function lock_file() {
    local lockfile="$LOCKS_DIR/$(basename "$1").lock"
    if ! mkdir "$lockfile" 2>/dev/null; then
        return 1
    fi
    return 0
}

function unlock_file() {
    local lockfile="$LOCKS_DIR/$(basename "$1").lock"
    rm -rf "$lockfile"
}

function convert() {
    local video="$1"
    local nom="$2"

    echo "Conversion de : ${BLUE}$video${NC}"
        echo

        if [[ "$codec" == "av1" ]] || [[ "$codec" == "V_AV1" ]]; then
                hw_opts=""
```

```
        scale_filter="scale=w=1920:h=1080:force_original_aspect_ratio=decrease"
        else
                hw_opts="-hwaccel cuda -hwaccel_output_format cuda"
scale_filter="scale_cuda=w=1920:h=1080:force_original_aspect_ratio=decrease"
        fi

        if (( height > 1080 )); then
                vf_filter="$scale_filter"
        else
                vf_filter=""
        fi

        ffmpeg -nostdin -hide_banner -loglevel error -stats \
        $hw_opts \
        -i "${video}" \
        ${vf_filter:+-vf "$vf_filter"} \
        -c:v hevc_nvenc -preset p6 -rc vbr -cq 28 -maxrate 14M -bufsize 28M \
        -c:a aac -b:a 192k -ac 2 \
        -c:s copy \
        "${nom}.tmp.mkv"

        local result=$?
    if [ $result -ne 0 ]; then
       echo "${RED}Erreur ffmpeg sur${NC} $video"
    fi
    return $result
}

#############################################################################
# Script

echo "Début de l'analyse des fichiers"
echo

find ~/videos/films -name "*.tmp.mkv" -delete

while IFS= read -r -d '' video; do
```

```
        echo "Analyse de ${BLUE}${video}${NC}"
        ((nb_total_fichiers++))

        if is_already_treated "${video}" || has_already_failed "${video}"; then
                continue
        fi

        extract_info
        normalize
        ((nb_analyses++))

        # Si relativement <3go et hevc OU <2Go => ignorer
   if ( (( weight < 3006477107 )) && [[ "$codec" =~ ^(hevc|h265|x265|av1|
V_MPEGH/ISO/HEVC|hvc1|hev1)$ ]] ) || ((weight < 2362232012)); then
                mark_as_treated "${video}"
                echo "${video}${BLUE} déjà optimisée${NC}"
                echo
        continue
        fi

        # Si height ou width indefinis => erreur
        if ! is_number "$height" || (( height == 0 )) || ! is_number "$width" || (( width
== 0 )); then
                ((NB_Format_error++))
                echo "${RED}Problème de format sur le fichier${NC} ${video}"
                echo
                mark_as_error "${video}"
                continue
        fi

        if ! lock_file "${video}"; then
   continue
        fi

        nom="${video%.*}" #suppression extension pour garder le nom
        convert "${video}" "${nom}"
```

```
        result=$?
        unlock_file "${video}"


#si la dernière action a réussi alors ; sinon
        if [ $result -eq 0 ]; then
                weight_after_conversion=$(ffprobe -v error -show_entries format=size
-of default=noprint_wrappers=1:nokey=1 "${nom}.tmp.mkv")
                gain=$(( weight - weight_after_conversion ))
                gain_percent=$(( gain * 100 / weight ))
                if (( weight_after_conversion < weight ))  && (( gain_percent >= 7 ));
then
                        rm -f -- "${video}"
                        mv -f -- "${nom}.tmp.mkv" "${nom}.mkv"
                        mark_as_treated "${nom}.mkv"
                        gain_mo=$(( gain / 1024 / 1024 ))
                        (( total_gain += gain ))
                        echo "${GREEN}Conversion de${NC} ${nom}.mkv ${GREEN}
réussie - gain : ${gain_mo} Mo${NC}"
                        echo
                elif ((weight_after_conversion > weight)) || ! is_number
"$weight_after_conversion" || ((gain_percent < 7)); then
                        rm -f -- "${nom}.tmp.mkv"
                        mark_as_treated "${video}"
                        echo "${ORANGE}Fichier non retenu (trop lourd ou gain
insuffisant: ${gain_percent}%)${NC}, suppression de ${BLUE}${nom}.tmp.mkv$
{NC}"
                        echo
                fi
                ((Nb_Conversion_successful++))
        else
                ((NB_Conversion_failed++))
                rm -f -- "${nom}.tmp.mkv"
                mark_as_failed "${video}"
                echo "${RED}Problème de conversion avec le fichier${NC} ${video}"
                echo
        fi

        # Limite d'heure
```

```
        heure_actuelle=$(date +%H)
        if (( heure_actuelle >= 17 )); then
                echo "Il est $(date +%H:%M), arrêt du script."
                heure_arret=1
                break
        fi

done < <(find ~videos/films -type f \( -iname "*.mp4" -o -iname "*.mkv" -o -iname "*.avi" -o -iname "*.webm" -o -iname "*.mov" \) -print0)

################################################################################

total_gain_go=$(echo "scale=2; $total_gain / 1024 / 1024 / 1024" | bc)
if ((heure_arret == 0)); then
        echo "Toutes les videos du dossier ont ete traitees"
fi
################################################################################

echo
echo "Nombre de fichiers convertis:.....${GREEN}${Nb_Conversion_successful}${NC}"
echo "Espace récupéré:.....${GREEN}${total_gain_go} Go${NC}"
echo "Nombre d'échecs de conversion: .....${RED}${NB_Conversion_failed}${NC}"
echo "Nombre de fichier ayant un problème de format: ${ORANGE}${NB_Format_error}${NC}"
echo "Nombres de fichiers analysés:.....${PURPLE}${nb_analyses}${NC}"
echo "Nombre total de fichiers:.....${BLUE}${nb_total_fichiers}${NC}"
```