Convert images/videos

Lossless conversion of webp to png

```
sudo apt-get install webp
dwebp file.webp -o file.png

#check:
convert file.webp ppm:- | sha1sum
convert file.png ppm:- | sha1sum
#or
if [ "$(convert file.webp ppm:- | sha1sum)" == "$(convert file.png ppm:- |
sha1sum)" ]; then echo "equal"; else echo "not equal"; fi

#recursively converting:
find . -name '*.webp' -type f -exec bash -c 'dwebp "$0" -o "${0%.webp}.png"'
{} \;
#find . -type f -name '*.webp' -delete
```

lossless conversion of png to webp

```
cwebp -z 9 "file.png" -o "file.webp"

#recursively converting:
find . -name '*.png' -type f -exec bash -c 'cwebp -z 9 "$0" -o
"${0%.png}.webp"' {} \;
#find . -type f -name '*.png' -delete
```

lossy conversion of png to jpg

Converts all png files in the directory recursively to jpg

```
find . -iname '*.png' | while read i; do mogrify -format jpg "$i" && rm
"$i"; echo "Converted $i to ${i%.*}.jpg"; done
```

lossless changing containers between webm/mov/mp4/mkv/m4v

webm/mov/mkv/mp4/m4v are container formats and can contain various encoded video or audio streams. To change the container between each other, the following command can be used, provided the used codecs inside the containers are compatible:

```
ffmpeg -i file.webm -c:a copy -c:v copy file.mkv

#recursively changing containers:
find . -iname '*.webm' -type f -exec bash -c 'ffmpeg -i "$0" -c:a copy -c:v
copy "${0%.*}.mkv"' {} \;
#find . -type f -iname '*.webm' -delete

#lossless change container mkv, webm, mov, m4v -> mp4
find . -type f \( -iname "*.mkv" -o -iname "*.webm" -o -iname "*.mov" -o -
iname "*.m4v" \) | while read f; do
    ffmpeg -i "$f" -codec copy "${f%.*}.mp4"
done
```

high quality conversion to mp4

```
#High quality non-mp4 to mp4 conversion
find . -type f \( -iname "*.mpg" -o -iname "*.asf" -o -iname "*.wmv" -o -
iname "*.mpg" -o -iname "*.mpeg" -o -iname "*.avi" -o -iname "*.divx" -o -
iname "*.rmvb" -o -iname "*.rm" -o -iname "*.m4v" -o -iname "*.flv" \) |
while read f; do
    ffmpeg -nostdin -i "$f" -vcodec libx264 -acodec aac "${f%.*}.mp4"
done
```

find all non-mp4 files

```
#Find non-mp4 files
find . -type f -not \( -iname "*.mp4" \)
```

downscale video to 720p mkv

The following command scales a video to 720p x264 codec with 30fps target framerate, copying the audio as is and any subtitles while reducing the overall quality with CRF of 28.

```
ffmpeg -i input.mp4 -vf scale=-1:720 -c:v libx264 -r 30 -crf 28 -c:a copy -
scodec copy output.720p.mkv
```

CRF option explained:

1. The range of the quantizer scale is 0-51: where 0 is lossless, 23 is default, and 51 is worst possible. A lower value is a higher quality and a subjectively sane range is 18-28. Consider 18 to be visually lossless or nearly so: it should look the same or nearly the same as the input but it isn't technically lossless.
2. The range is exponential, so increasing the CRF value +6 is roughly half the bitrate while -6 is roughly twice the bitrate. General usage is to choose the highest CRF value that still provides an acceptable quality. If the output looks good, then try a higher value and if it looks bad then

choose a lower value.

Scale option -1 means the output has to be divisible by 1 with same aspect ratio. Scale option -2 means the output has to be divisible by 2, etc.

To limit bitrate to 2Mbit, add

```
-b:v  2M -maxrate 2M -bufsize 1M
```

Useful bash script, put in ~/.local/bin and chmod +x it after:

video_downscale.sh

```
#!/bin/bash
maxwidth=1280
maxheight=720
bitrate="2M"
bufsize="1M"
# mp4 is better for streaming, mkv supports all sorts of mixed codecs
and subtitles
#container="mkv"
container="mp4"
#encoder="libx264" #libx264 = CPU, better quality and much smaller
filesize; h264_amf = AMD GPU; h265_nvenv = Nvidia GPU; h264_qsv = Intel
GPU

# ffpb is a wrapper for ffmpeg to show a progress bar and the remaining
time.
# install it using:
# pip install ffpb

# Check if ffpb is installed, then use it, otherwise use normal ffmpeg
builtin type -P "ffpb" &> /dev/null && binary="ffpb" || binary="ffmpeg"

# Check if a parameter is provided
if [ -z "$1" ]; then
    echo "Usage: "$(basename $0)" <file or files>"
    exit 1
fi

for input in "$@";
do
    if [ -f "$input" ]; then
        echo "Processing file: $input"
        output="${input%.*}.convertednew"
        # Check if mp4 container is desired and if subtitle is ASS
format, then convert to srt, otherwise just copy subs
        subtitles_present=$(ffprobe -v error -select_streams s -
show_entries stream=codec_name -of csv=p=0:s=x "$input")
        if [ "$subtitles_present" == "ass" ] && [ "$container" == "mp4"
];
```

```
        then
            subtitle_option="-c:s mov_text -metadata:s:s:0 language=en"
        else
            subtitle_option="-scodec copy"
        fi
        #check framerate to only ever reduce it and not increase it
        # ffprobe returns 25/1 or 24000/1001. result needs to be
calculated and needs to be an integer, thus the bash $(( )).
        fps=$(ffprobe -v 0 -of csv=p=0 -select_streams v -show_entries
stream=avg_frame_rate "$input" | sed 's#/# / #g')
        if [ "$fps" == "0 / 0" ]; then
            # some video files return 0 / 0 for avg_frame_rate, using
r_frame_rate instead
            fps=$(ffprobe -v 0 -of csv=p=0 -select_streams v -
show_entries stream=r_frame_rate "$input" | sed 's#/# / #g')
        fi
        fps=$(( $fps ))
        if [ $fps -gt 30 ]; then
            fps_option="-r 30"
        else
            fps_option=""
        fi
        #-c:a copy   #copies audio as is, but mp4 works best with aac
and wma cannot be in mp4 files
        #$binary -i "$input" -vf scale=-2:$resolution -c:v libx264 -r
30 -crf 28 -c:a aac -scodec copy -b:v $bitrate -maxrate $bitrate -
bufsize $bufsize "$output"
        # using complex filter to prevent upscaling and only ever
downscale
        cmd=$binary' -i "'"$input"'" -filter_complex
"scale=ceil(iw*min(1\,min('$maxwidth'/iw\,'$maxheight'/ih))/2)*2:-2" -
c:v libx264 '$fps_option' -crf 28 -c:a aac '$subtitle_option' -b:v
'$bitrate' -maxrate '$bitrate' -bufsize '$bufsize' -f '$container'
"'"$output"'"'
        $binary -i "$input" -filter_complex
"scale=ceil(iw*min(1\,min($maxwidth/iw\,$maxheight/ih))/2)*2:-2" -c:v
libx264 $fps_option -crf 28 -c:a aac $subtitle_option -b:v $bitrate -
maxrate $bitrate -bufsize $bufsize -f $container "$output"
        if [ $? -eq 0 ];
        then
            actualheight=$(ffprobe -v error -select_streams v -
show_entries stream=height -of csv=p=0:s=x "$output")
            #output2="${output%.*}."$actualheight"p.$container"
output2="${output//.convertednew/.$actualheight""p.}$container"
            mv "$output" "$output2"
            echo "Output file: $output2"
        else
            echo "An error occured. File not converted properly!"
            echo "Full ffmpeg command:"
            echo "${cmd//ffpb/ffmpeg}"
        fi
```

```
    fi
done

# When using in Nemo, it's helpful to force a key to be pressed before
closing the terminal to see the status or any errors. Uncomment the
following line:

#echo;read -rsn1 -p "Press any key to continue . . .";echo
```

To add this to Nemo filemanager as right-click option for video files, create a nemo_action file in ~/.local/share/nemo/actions

video_downscale.nemo_action

```
[Nemo Action]
Name=Video Downscale to max 720p
Comment=Video Downscale to max 720p
Exec=video_downscale.sh %F
Icon-Name=stock_down
Selection=notnone
Extensions=mp4;wmv;avi;mkv;mov;webm;mpg
Quote=double
EscapeSpaces=true
Terminal=true
```

When using in Nemo, it's helpful to add the following to the very end of the video_downscale.sh script to force a key to be pressed before closing the terminal to see the status or any errors:

```
echo;read -rsn1 -p "Press any key to continue . . .";echo
```

Reencode videos with high bitrate

processfiles.sh

```
#!/bin/bash
MYFILES=$(find /media/videofiles -type f -iname "*.mp4")
SAVEIFS=$IFS

IFS=$(echo -en "\n\b")
for FILE in ${MYFILES}
do
  bitrate=$(ffprobe -v quiet - select_streams v:0 -show_entries
stream=bit_rate -of default=noprint_wrappers=1:nokey=1 $FILE)
  if ! [[ $bitrate =~ ^[0-9]+$ ]];
  then
      continue
```

```
  fi
  if [ $bitrate -gt 8000000 ]
  then
      echo $bitrate" | "$FILE
      video_downscale.sh "$FILE"
  fi
done
IFS=$SAVEIFS
```

3D VR SBS to 2D

https://blog.interstellar.co.jp/en/2022/06/21/converting-vr180-videos-to-2d-videos-with-ffmpeg/

```
ffmpeg -i input.mp4 -vf
v360=input=equirect:output=flat:ih_fov=180:iv_fov=180:h_fov=93:v_fov=121:in_
stereo=sbs:w=960:h=640 -codec:v h264 output.mp4
```

https://video.stackexchange.com/questions/21084/how-to-convert-a-3d-movie-to-2d-using-ffmpeg

```
ffmpeg -i input.mkv -vf stereo3d=sbsl:ml -metadata:s:v:0 stereo_mode="mono"
output.mkv
```

Use sbsl2:ml or -aspect 16:9 if the aspect ratio is wrong

https://stackoverflow.com/questions/66960003/unwarping-180-vr-footage-with-ffmpeg-v360-filter

https://github.com/paulpaul999/vr-video-notes/blob/main/vr-to-flat/README.md

```
ffmpeg -i equirectangular.mp4 -filter:v
"v360=input=hequirect:output=flat:in_stereo=sbs:out_stereo=2d:d_fov=125:w=19
20:h=1080:pitch=+5" -map 0 -c copy -c:v h264 -pix_fmt yuv420p flat.mp4
```

Mono to Stereo MP3

Conversion using ffmpeg -ac 2 reduces volume by 3db, apparently to keep perceived loudness the same. To maintain the same volume use pan as below.

```
for i in {01..33}; do
  ffmpeg -i "Chapter "$i".mp3" -af "pan=stereo|c0=c0|c1=c0" "Chapter
"$i"s.mp3";
done
```