

# FFMPEG

---

## Introduction and Examples

This presentation is about ffmpeg, a command line utility for converting and streaming video and audio. Besides a short introduction, I will run 10 live examples.

## Project / Tool Suite

- 3 main executables
  - ffmpeg (conversion)
  - ffprobe (interrogation)
  - ffserver (streaming)
- Libraries for inclusion in your projects



Ffmpeg is an open source project. Its main executables are ffmpeg (for converting video and audio), ffprobe (for getting video and audio properties), and ffserver (for streaming video and audio). You can also incorporate ffmpeg directly into your project through its native libraries. I will only be discussing ffmpeg, except for one ffprobe example.

Ffmpeg runs on Linux, Windows, and OSX.

Now we will start the examples.

## MP3 from Video

```
>ffmpeg -i imagine.mp4 imagine.mp3
```

- -i := next token is input
- last token is output
- audio transcoded from AAC to MP3

Perhaps you have a video and you like the sound track. Lets get the audio from an MP4 and encode it in MP3 format.

The command line is `ffmpeg -i <input video> <output audio>`

-i means the next token is the input – in this case a video file

There is no corresponding -o option. Anything on the command line that is not interpreted as an input file or option is assumed to be the output file.

In this case, the audio stream within `imagine.mp4` was in AAC format. So it had to be converted from AAC format to MP3 format. Technically, the `imagine.mp4` container was first demuxed into a video stream and an audio stream in AAC format. Then the AAC audio stream was decoded and encoded into MP3. Finally, it was muxed back to the output file in an MP3 container format.

## Media Container, Streams, CODECs

```
>ffprobe -i imagine.mp4
```

```
Input #0: mov,mp4,m4a,3gp,3g2,mj2, from 'imagine.mp4':
Metadata:
  major_brand      : isom
  minor_version    : 512
  compatible_brands: isomiso2avc1mp41
  encoder          : Lavf54.63.104
Duration: 00:03:57.25, start: 0.000000, bitrate: 1147 kb/s
Stream #0:0(und): Video: h264 (Main) (avc1 / 0x31637661), yuv420p, 1280x720 [SAR 1:1 DAR 16:9],
Metadata:
  handler_name     : VideoHandler
Stream #0:1(und): Audio: aac (mp4a / 0x6134706D), 44100 Hz, stereo, fltp, 125 kb/s (default)
Metadata:
  handler_name     : SoundHandler
```

So you may be wondering how I knew that the audio stream of the previous example input file `imagine.mp4` was in AAC format.

This is what `ffprobe` is for. `Ffprobe` tells you characteristics of containers and streams.

The syntax of the `ffprobe` command is very similar to the `ffmpeg` command – they both share many of the same options.

In the most basic case, you just specify the input file for which you would like to investigate the containers and streams.

In this case, Input #0 identifies the container to be of type `mp4`, `m4a`, etc.

Notice the duration and overall bitrate.

Stream #0:0 is the video stream and it is encoded in `h264` at a size of `1280x720`.

Stream #0:1 is the audio stream and it is encoded in AAC with a sampling rate of 44 kHz

#0:0 and #0:1 are stream specifiers. The first 0 in both means the first input file. The next number (0 or 1) is the stream number within the file. `Ffmpeg` commands can

have more than 1 input file, so sometimes the first number greater than 0.

## Copy Audio from Video

```
>ffmpeg -i imagine.mp4 -vn -c:a copy imagine.aac
```

- -vn := video. none.
- -c:a copy := copy audio in its input codec
- VERY FAST. bitwise copy

In the previous example, we transcoded (decoded then encoded) the audio stream from AAC to MP3.

But this re-encoding could introduce losses and also takes longer than simply copying a stream.

To extract the exact audio stream from the input file, we will use the “copy” codec.

Again, the input file is specified after -i.

Then the vn option tells ffmpeg to drop the video stream. The v means video and the n means no.

Then we specify the audio codec. -c means codec. :a qualifies c to audio and copy is the name of the codec. Technically, calling copy a “codec” is a bit of a misnomer, because nothing is being encoded. It is just a bit for bit copy, which makes it super fast.

Finally, the last token on the command line is taken to be the output file name.

## Reduce Audio Bitrate

```
>ffmpeg -i imagine.aac -ab 96k imagine96.aac
```

- -ab 96k := audio. bitrate. 96kbps
- requires re-encoding of audio
  - much slower than -c:a copy
- output file name extension determines codec
- bit depth (vertical)
  - possible amplitude levels
- sampling rate (horizontal)
  - time between samples
- bit rate (combination vertical + horizontal)
  - bits/s
- when changing bitrate, sampling rate does not change
  - effectively changing bit-depth

You may want to reduce a media file's size – perhaps for storage or streaming considerations.

To reduce the size of an audio file, you can specify its bitrate.

Again, -i identifies the input file

-a means audio. b means bitrate. And 96k means 96 kbps. This is a good time to talk about bits vs bytes and powers of 1000 vs 1024. lower case postfixes such as k are in bits. Upper case postfixes such as capital K are in bytes. Finally, you can insert a lower case i to mean 1024 instead of 1000.

And the last token is the output file.

Since we are changing the content of a stream, this example requires decoding and re-encoding which is much slower than the copy codec from the previous example.

Bit rate, bit depth, and sampling rate are often confused. Imagine you are looking at this audio in Audacity or some other audio editing tool. Bit depth is the number of possible amplitude levels. Sampling rate is the time duration between samples of the input. Bit rate is the combination of bit depth and sampling rate. Since ffmpeg doesn't change the sampling rate, changing the bit-rate effectively changes the bit-

depth.



## Copy Clip

```
>ffmpeg -i blink_lv.mp4 -ss 13:38 -t 02:00 -c copy i_miss_you_lv.mp4
```

- -ss 13:38 := skip 13 min 38 seconds into input
- -t 02:00 := duration. process input for 2 min
- -c copy := CODEC. copy.
  - shortcut for -c:v copy -c:a copy -c:s copy
  - -c is short for -codec
- notice video is messed up at start of output clip
  - because of key frames – GOP – to be discussed later

You may have a long clip from which you want to copy a time section.

Here is a Blink-182 concert. The song “I Miss You” starts at 13:38 into video and lasts 2 minutes. We want to extract it.

-ss specifies where the input frames should start being passed to the output

-t specifies for how long the input frames should be passed to the output

We use the copy codec again, but this time we use the shorthand of -c copy. Since there was no qualifier on the CODEC option, it applies for all input streams – video, audio, and subtitles.

Notice that the output file is messed up at the beginning. This is because of where the key frames were in the input file. Most video codecs have full-frames where all the pixels are specified and difference-frames where only the changes are specified. This concept is called GOP (or group of pictures) and we will talk about it again in a few slides

## Video Snapshots

```
>ffmpeg -i blink_lv.mp4 -r .01 blink_lv_%03d.png
```

- `-r .01` := process input at rate of .01 Hz
  - (1 frame every 100 seconds)
- `blink_lv_%03d.png`
  - `blink` := Blink-182 – a great band
  - `%03d` := string format for each image output

You may want to quickly review a long video. One way to do this is to create periodic image snapshots.

`-r` specifies the input rate in Hz.

As usual, the last token represents the output file, but in this case since there will be many output PNGs, the file name has a printf style format which resolves to the one-up output number.

## 1080 to 720

```
>ffmpeg -i michael_soccer1.MOV -vf scale=-1:720 michael_soccer1_720.MOV
```

- -vf scale=-1:720 :=
  - v := video.
  - f := filter
  - scale := scale filter. change dimensions
  - -1 := width. retain width/height ratio from source
  - 720 := height
- Re-encoding

### Stream mapping:

```
Stream #0:0 -> #0:0 <h264 <native> -> h264 <libx264>>  
Stream #0:1 -> #0:1 <pcm_s16le <native> -> aac <libvo_aacenc>>
```

We reduced an audio file size by reducing its bitrate. We can possibly reduce a video file size by reducing the frame size.

-v means video. F means filter. Scale is the name of the filter. -1 means retain the aspect ratio from the input file. 720 means the output height should be 720 pixels.

This involves re-encoding since the video stream has to be decoded.

And it shows the some defaults which are applied. Our output file has the extension .MOV which to ffmpeg implies a AAC audio stream. So in addition to the video stream being transcoded, the audio stream was also transcoded. This could be avoided by specifying -c:a copy.

## Animated GIF

```
>ffmpeg -i michael_soccer1_720.MOV -r 2 michael_soccer.gif
```

- -r 2 := take frames at 2 Hz (every 0.5 seconds)

Similar to taking video snapshots, we can create an animated GIF.

This time there is no printf style format string in the output file.

## Merge Audio and Video

```
>ffmpeg -i revolution.mp3 -i michael_soccer1_720.MOV -c:v copy -c:a copy -shortest soccer_w_music.mp4
```

- -i revolution.mp3 := audio we want
- -i michael\_soccer1\_720.MOV :=
  - contains video we want
- -shortest := end encoding after shortest clip
- default stream mappings
  - how was audio chosen? revolution.mp3, michael\_soccer, both?

```
Input #0, mp3, from 'revolution.mp3':
  Stream #0:0: Audio: mp3, 44100 Hz, 268 kb/s
Input #1, mov,mp4,m4a,3gp,3g2,mj2, from 'michael_soccer1_720.MOV':
  Stream #1:0(jpn): Video: h264 (High) (avc1)
  Stream #1:1(jpn): Audio: aac (mp4a) 48000 Hz,
Output #0, mp4, to 'soccer_w_music.mp4':
  Stream #0:0(jpn): Video: h264
  Stream #0:1: Audio: mp3 44100 Hz,
Stream mapping:
  Stream #1:0 -> #0:0 (copy)
  Stream #0:0 -> #0:1 (copy)
```

You may want to merge an audio and video file together, perhaps dropping the audio of the input video in favor of an external music track.

So we specify 2 input files.

We use the copy codec on both the video and audio stream.

So that the streams both end at the same time, we specify the shortest option.

You may be wondering why we didn't get both input audio streams, or why the audio stream from the video file didn't take preference over the external audio file. This again comes down to some default rules. Ffmpeg selects the input audio stream with the highest sampling rate by default. There are further tests for a tie. If you don't like the default rules, you can override them with the map option.

## Make a Ringtone

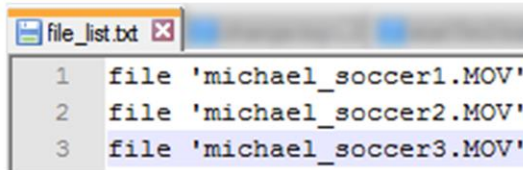
- Trim/Select audio to < 30 seconds
  - -ss hh:mm:ss := skip to beginning of output
  - -t 00:15 := 15 second clip
- copy to /sdcard/Music/Ringtones
- select in Settings-Sounds

You can make a ringtone for your phone by trimming an audio clip to less than 30 seconds then dropping it in /sdcard/Music/Ringtones

# Concatenate Videos

```
>ffmpeg -f concat -i file_list.txt -c copy michael_soccer_concatenated.MOV
```

- -f concat := force format. concatenate format.
- -i file\_list.txt := file names to concatenate in file\_list.txt
- -c copy := use same audio/video CODECs as source. bitwise copy



```
1 file 'michael_soccer1.MOV'
2 file 'michael_soccer2.MOV'
3 file 'michael_soccer3.MOV'
```



```
1 $srcDir = "C:\Users\Tim\Documents\ffmpeg_brownbag"
2 $fileList = "C:\Users\Tim\Documents\ffmpeg_brownbag\file_list2.txt"
3 Get-ChildItem $srcDir -File | ? {$_.Name -match "michael_soccer\d+\.MOV"} |
4 Sort Name | % { "file '" + $_.Name + "' " | Out-File -FilePath $fileList -Append }
```

You may have many small videos to concatenate into a larger video – perhaps after selecting the good clips out of a video.

For this, we use the concat format.

-f means format. Concat means concatenate the inputs

The inputs can be specified on the command line, but usually it is easier to put them in a file, so the input file becomes an index of the clips you want to concatenate.

As before, we use the copy codec.

There is a Powershell script to create the input file.

# Containers, Streams, CODECs

- Container is a file format
  - streams are within containers
  - examples: MP4, WebM, AVI
- Streams are media components
  - Video
    - H.264, MPEG-2, WMV, VP8
  - Audio
    - MP3, AAC, WMA
  - Subtitles
- Streams are formatted/stored/encoded with CODEC
  - generally, any CODEC/stream can be within a container, but numerous exceptions

```
>ffmpeg -formats
File formats:
D. = Demuxing supported
.E = Muxing supported
DE flv FLV (Flash Video)
E matroska Matroska
D matroska.webm Matroska / WebM
D mov.mp4.m4a.3gp.3g2.m42 QuickTime / MOV
```

```
>ffmpeg -codecs
Codecs:
D.... = Decoding supported
.E.... = Encoding supported
..U... = Video codec
..A... = Audio codec
..S... = Subtitle codec
...I.. = Intra frame-only codec
....L. = Lossy compression
.....S = Lossless compression
DEU.L. mpeg4 MPEG-4 part 2 (encoders: mpeg4 libxvid )
DEU.L. theora Theora (encoders: libtheora )
```

I have been referring to containers, streams, and codecs.

A container is almost synonymous with a file format. It is how more than 1 stream can be stored together – such as a video and audio stream in the same file.

Streams are the media components such as video, audio, and subtitles.

Common video streams are H264 and MPEG2.

Common audio streams are MP3 and AAC.

Containers are demuxed into streams. The streams are then decoded into frames. Then filtering is done on the frames. Then the frames are encoded back into streams and muxed back into the output container – usually in a file.



## GOP

- “Group of Pictures”
- full frame vs differential frame
- lots of video doesn’t change significantly from frame to frame
- I-frame: full-frame (intra-frame). independent of other frames
- {P, B, D}-frames: based on other frames
- larger GOP
  - smaller file size
  - seeking becomes difficult

We talked a bit ago about GOP – group of pictures.

Some frames are full-frames – all the pixel values are specified.

But since most of the picture content of videos doesn’t change from frame to frame, there are differential frames.

GOP specifies how often a full-frame is put into the output. A larger GOP is good for file size, but can make seeking difficult

## Tips

- order of options matter
- `-loglevel panic` := make ffmpeg silent
- `-y` := always overwrite output w/o asking
- `-n` := never overwrite output
- Save help
  - `ffmpeg -h > ffmpeg_h.txt`
  - `ffmpeg -h long ffmpeg_h_long.txt`
  - `ffmpeg -h full ffmpeg_h_full.txt`

Where you put an option on the ffmpeg command line matters. The same option could apply to the input file or output file depending on where you put it. So if your command is not working, make sure all the options are in the right positions.

You may want to programatically call ffmpeg and ignore logging output. Use `-loglevel panic` for this.

The `y` and `n` options specify if you want to automatically overwrite output files or never overwrite output files

Finally, read the documentation, which can be got with the `h` option.