# Tips para FFmpeg

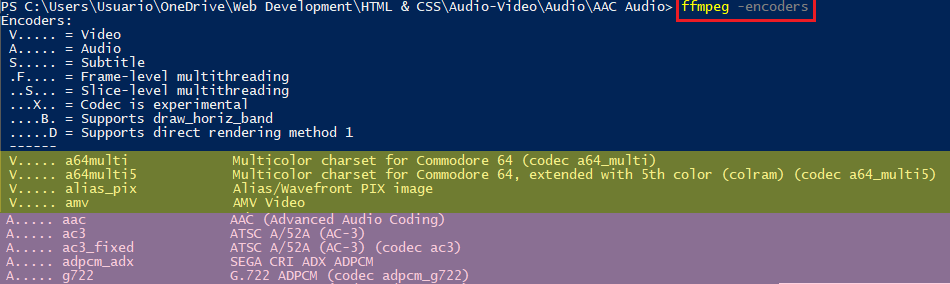
Most of the equipment available on the market (cameras, cellphones and even audio recorders) usually deliver files already compressed. These devices should ideally record or store the media in uncompressed files like **WAV or AIFF** to make up for inevitable future loss when re-encoding, but I guess it also makes sense that media is stored with compressed formats. Specially, on mobile devices.

## Encoders Vs. Codecs (?)

****Throughout this guide you´ll see a lot the word *codec* which implicitly implies that codecs like **VP9, H.264 or AAC** do the actual encoding when they are actually the resulting file of *encoders* like: libx264 for **H.264,** libvpx-vp9 for **VP9** and libav for **AAC,** justto mention few codecs and their respective encoders. There is actually more than one encoder for each codec ***(for the most part... I think, it seems a few codecs like Opus or Vorbis only have one encoder.. not sure),*** *I didn’t want to go deeper on the multiple encoders that there is for each codec and the features you may see on one encoder but not in the other which btw does a better job compressing files while doing it faster and keeping a good quality*. All that bullshit is way beyond the scope I should cover as web designer for now, I think… However, I ran into an issue that I didn’t fix when encoding to AAC using the native encoder compiled into FFmpeg back then. As I dropped the bitrate of the main AAC audio copy using the **CBR (Constant Bit Rate)** parameter I noticed a mismatch between the CBR I chose to encode with and the output of FFprobe when confirming the bitrate of the outcome AAC files. According to **[this article](https://video.stackexchange.com/questions/27760/why-is-ffprobe-showing-another-bitrate-than-the-one-i-encoded-with)** ithappened because the native AAC encoderI had compiled in FFmpeg back then didn’t have the feature to encode with CBR, only VBR or so I think. I was missing the AAC encoder called “fdk-aac” apparently, which does offer CBR encoding. Like I said, I wanted to avoid the hustle of dealing with the various encoders and their features or even worse, compiling other encoders like “fdk-aac” into my FFmpeg build.[Here](https://stackoverflow.com/questions/18746359/compile-ffmpeg-with-libfdk-aac) is a guide on how to compile that missing encoder. Later, I found out that the mismatching was not only happening on AAC but on every other [audio codec.](#exp) So, it is like every default encoder for every codec I needed was either **CBR** disabled or partially enabled because I actually got them compressed **(smaller size)** while using **CBR** parameters on the commands. I prefer **CBR** over **VBR** because it allows you to have a more granular measure and comparison over the size of the files encoded. Here is a command example on how to use **VBR (Variable Bit Rate),** the quality rate range goes from **1** to **5** and this example has a quality rate of **5,** which is the highest. This command was done using the native and *only* **AAC** encoder in my FFmpeg build back then, notice the **-c:a aac** parameter inside the yellow box **(c** for **codec** and **a** for **audio)**. Its function is to choose the encoder but since we only had the native **AAC** encoder there was no need to use it:

If you compile another **AAC** encoder into FFmpeg, like “fdk-aac” the command would be the one below instead. You would have to use the parameter inside the yellow box, which tells FFmpeg to use the encoder “fdk-aac” instead of the native one above.

You can use the command below to display all the encoders compiled in your current **FFmpeg** build. The output from this command is huge so I decided to cut off a few snapshots from it and put them together, which shows a few encoders and the stream they encode for. Yellow for video encoders and pink for audio ones.



## Retaining Quality

There were some comments in forums suggesting that a good way to mitigate data loss when re-encoding ***a lossy file*** is to command the output file to have the highest **Bit Rate** or **CRF** possible. Such is the example with my audio named ***Ridiculous*** which was an already compressed [**Opus**](#opus)audio file I made myself from my mobile that later was re-encoded to other 3 *(less efficient)* codecs: **mp3, Vorbis & AAC,** on my PC**.** This initial re-encoding to other codecs is where I figured it makes sense to set a high **Bit Rate** to mitigate data loss as much as possible.

This section of [**retaining quality**](#retaining) is one of the latest I composed in this guide, meaning that I came to know about this ***“Tip”*** after re-encoding all my videos and audios so, I didn’t worry much about the quality of output audio or video files when re-encoding, I just wanted to have diversity of formats and codecs for web support and low file size for quick loading times. I started re-encoding my cats video named ***Lechita*** that I recorded using my mobile too, it was a **WebM** video file container that had inside a **V9** video codec & **Opus** audio codec. Both codecs are among the most powerful and widely supported compression tools nowadays. With this format it had a size of **1.54 MB** which then became **9.89 MB** when re-encoded to an **MP4** video file container that had inside a **H.264** video codec & **AAC** audio codec, both widely supported codecs that are less efficient in compressing a videos and audio. It had taken these codecs a lot more bytes to encode a video and audio with the same quality.

Going back to the previous example about my audio ***Ridiculous*** that was originally compressed with **Opus** and from which 3 other formats came out around %60 - %65 bigger. I couldn’t tell a difference in quality between the original files: ***Ridiculous & Lechita*** and the re-encoded versions of them but I did notice the rise in file size which is obviously due to the compression capabilities between the newer codecs like **V9** or **Opus** and the older ones like **H.264** or **AAC, etc.**

Back then, as a rookie all I did to re-encode was this, because the goal was just to get another format, later on I would worry about file size & quality:

******

I didn’t know about retaining quality by mitigating data loss by re-encoding the output file with the highest **Bit Rate** or **CRF** possible or the **-QSCALE** parameter and yet just by using the command above I got the same quality or so I think. Of course, I got bigger files with the less efficient video & audio codecs but upon re-encoding and testing many times I figured later a good balance between size and quality.

## Selecting your codecs

So what do you do when you want to use a container like Matroska (which can handle almost any stream) but still influence which codecs are in the output? FFmpeg to the rescue! You can select the codecs needed by using the -c flag.

This flag lets you set the different codec to use for each stream. For example, to set the audio stream to be Vorbis, you would use the following command:

ffmpeg -i input.mp3 -c:a libvorbis output.ogg

The same can be done to change the video as well as the audio stream:

ffmpeg -i input.mp4 -c:v vp9 -c:a libvorbis output.mkv

This will make a Matroska container with a [VP9](https://en.wikipedia.org/wiki/VP9) video stream and a Vorbis audio stream, essentially the same as the WebM we made earlier.

The command **ffmpeg -codecs** will print every codec FFmpeg knows about. The output of this command will change depending on the version of FFmpeg you have installed.

## Changing a single stream

More often than you'd like, the file you have is partially correct with only a single stream in the wrong format. It can be very time consuming to re-encode the correct stream. FFmpeg can help with this situation:

ffmpeg -i input.webm -c:v copy -c:a flac output.mkv

This command copies the video stream from input.webm into output.mkv and encodes the Vorbis audio stream into a FLAC. The -c flag is really powerful.

## Changing a container

The prior example can be applied to both the audio and video streams, allowing you to convert from one container format to another without having to do any additional stream encoding:

ffmpeg -i input.webm -c:av copy output.mkv

## Profile and level

The profile defines which tools the decoder must use in order to decompress/play the video. Levels specify the size of the video a decoder must be able to handle. They specify a maximum bit-rate for the video and a maximum number of macroblocks per second. I didn’t dig into these parameters because they are advance for my purpose which back then was determine the level and profile so that I could complete the MIME type with the codec parameters.

### H.264

For this example, I used a **Webm** file to re-encode into an **MP4** with a given profile and level. You can also use the **-c:v copy** parameter if all you want to do is changing the profile. This [**link**](https://developer.mozilla.org/en-US/docs/Web/Media/Formats/codecs_parameter#ISO-BMFF), [**this** **one**](https://cconcolato.github.io/media-mime-support/#audio/mp4;%20codecs=%22mp4a.40.2%22), [**this other too**](https://superuser.com/questions/563997/how-can-i-set-a-h-264-profile-level-with-ffmpeg) and [**this one**](https://stackoverflow.com/questions/16363167/html5-video-tag-codecs-attribute) were very handful to figure how to encode with different profiles, levels and their respective codec parameter once input into HTML. These links also helped me to clear the confusion I had with how FFprobe showed the level of H.264 videos because, instead of having a “.” Dot in between as shown in the commands below, it shows like this .





There are sub levels as we see with the main profile.









You can also specify the encoder you want to use in case you had multiple encoders for the same codec. I think I only had one encoder for H.264 **“x264”.** So, I didn’t have to enter the parameter inside the yellow box below.





### AAC

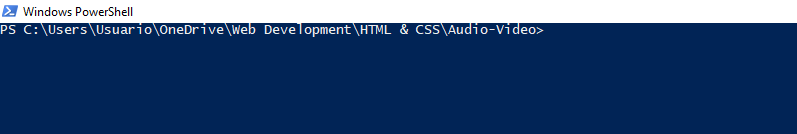
The example for encoding with other profiles like AAC High Efficiency v1 “**HE-AAC”** or AAC High Efficiency v2 **“HE-AAC v2”** wasn’t possible as I didn’t “fdk-aac”, which is the best encoder for AAC apparently.Besides, **LC (Low Profile)** which is default when encoding with the native AAC encoder is better for low bit rates, which is exactly the goal I had when reencoding the original audio as I used the audios on my webpage. Another reason why is worth to stick to LC is because compiling “fdk-aac” into the FFmpeg build on Windows is very complex.

## Video

* A screenshot of a cell phone

  Description automatically generatedTo easily work with FFmpeg from any folder you need to set up a PATH VARIABLE in windows. [Here](https://www.youtube.com/watch?v=MPV7JXTWPWI) is how.
* With the PATH VARIABLE set up you can **SHIFT+RIGHT-CLICK** and you´ll see the option inside the red box below:

This is how the **POWER\_SHELL WINDOW** looks like:



### How I reduced file size and yet kept a decent quality.

#### Resolution Downgrade

##### Simple Rescaling:

###### **MP4**



Before scaling the resolution down the original video **“lechita.mp4”** had a resolution of 720x1280 ***“where the 1st value is width and the 2nd is height”*** so due to the dimensions you can tell the video was and remains taller than wider **“fit for mobile in a portrait orientation”.** You may also notice that you are free to choose the resolution dimensions that you want, meaning that you are not bond to the common resolutions such as: **360, 460, 720 (all referring to the width´s resolution of the video).**

**This** is another way of rescaling which is a bit less verbose.

****

The command utility FFmpeg **DOESN´T** take figures not divisible by 2 when scaling **MP4** files, while **WebM** [**DOES**](#does) take not divisible figures.

I chose **416** for **width** because that is the viewport size of the mobile I was working with.

###### **WebM**



Before scaling the resolution down the original video **“lechita.webm”** had a resolution of 720x1280 ***“where the 1st value is width and the 2nd is height”*** so due to the dimensions you can tell the video was and remains taller than wider **“fit for mobile in a portrait orientation”.** You may also notice that you are free to choose the resolution dimensions that you want, meaning that you are not bond to the common resolutions such as: **360, 460, 720 (all referring to the width´s resolution of the video).**

**This** is another way of rescaling which is a bit less verbose.



The command utility FFmpeg **DOES** take figures not divisible by 2 when scaling **WebM** files.

The file size optimization of **WebM** files when *rescaling* is almost useless as I had to rescale the 720x1280original video to an ***awful*** resolutionof 100x250 to barely see a difference in file size, it went from **1.54MB** to **1.43MB.** There was **NO DIFFERENCE** in file size between the original resolution 720x1280 and 416x700, which is insane.

##### Keeping the Aspect Ratio

If we'd like to keep the aspect ratio, we need to specify only one component, either width or height, and set the other component to -1. For example, this command line:

 This one is for **MP4.** The *parameter* for *scaling* has to be **-vf,** it can´t be **-s.**

 This one is for **Webm.** The *parameter* for *scaling* has to be **-vf,** it can´t be **-s.**

This will set the width of the output video frame to 416 pixels and then calculate the height of the output video frame according to the aspect ratio of the input video. The resulting video will have a dimension of 416⨉740 pixels.

##### Aspect Ratio



#### Quality Downgrade

##### Constant Quality

###### **MP4**

There is a parameter called ***quantizer*** which affects the quality of the encoded file. For **MP4** is simply ***“-crf+Number”*** andI didn’t use the whole command line example as I did with the **WebM** example below because every command example shown in this word file was made with **MP4**. The quantizer values go from 1 to 50 for **MP4** video, ***the higher the number the worse the quality and smaller size.*** Values below **25** don´t really justify the file size for the quality added but values above **32** startlose too much quality. The right spot being between ***-crf 30 Or 32.* E.g:**



###### **WebM**

For **WebM** is ***“-crf+Number -b:v 0”*** where the **-b:v 0** I think stands for **bitrate:video*.*** The *quantizer* values for **WebM** files goes from 1 to 63, ***the higher the number the worse the quality and smaller size.*** Considering how small was the **WebM** file after encoding it from the original **MP4** there wasn´t much room for file size optimization. However, I started with *quantizer* at **-crf 63** whichreduced the file size by almost a third along its quality which worsened badly. Around **-crf 54** the file size started to decrease while keeping good quality and finally tried **-crf 59** which reduced the file size around 20% *(by going from* ***1.54MB*** *“on lechita.webm”**at full size to* ***1.29MB*** *upon using* ***-crf 59)*** while keeping a decent quality. **E.g:**



**WebM** is a bit less straightforward than **MP4** on *quality* adjustments because if you don’t use ***-b:v 0*** on the command the compression will be *significantly* less efficient.

#### Single command for Rescaling and Quality Decrease!

###### **MP4**

How to scale resolution down while adjusting the quantizer with a ***single*** command?



For **MP4** there was pretty much no difference in size nor quality between encoding using this command above ***“which is way quicker and more practical”*** and the two-step manner I explain below in [**Notations**](#nota)**.**

###### **WebM**

Beyond crap quality – smallest size possible **660KB,** check [**this**](#QA) section which shows the file size received with ***-crf 63*** when dropping quality exclusively, **NO** rescaling at all.

 Still crappy quality – 2nd smallest size **829KB,** check [**this**](#QA) section which shows the file size received with ***-crf 59*** when dropping quality exclusively, **NO** rescaling at all.

 Very bad quality but kind of acceptable **980KB,** check [**this**](#QA) section which shows the file size received with ***-crf 54*** when dropping quality exclusively, **NO** rescaling at all.

** Acceptable quality,** this is the sweet spot **1.27MB,** the value ***-crf 49*** just above didn’t show on [**this**](#QA) section as the others because the file size wasn’t optimal. The file size we got combining rescaling at 416x700 and ***-crf 49*** even beats the file size obtained with ***-crf 59*** from [**this**](#QA) section without even rescaling. <3

As you can tell using both parameters ***(rescaling and quantizer)*** simultaneously to optimize file size is ***way*** less straightforward for **WebM** file than what it is for **MP4** files. There is more background as to why that is in the paragraph below.

Among the values given to the *quantizer* in [**this**](#QA) previous sectionI used ***-crf 59*** individually *(meaning there was* ***NO*** *rescaling that time)* which reduced the file size around 20% as we saw [**here**](#perce)**,** getting in return a file size of ***1.29MB*** which is what I kind of expected for encoding using quantizer and rescaling parameters simultaneously in a single command. The reason why I kind of expected around ***1.29MB*** is because rescaling **WebM** video files has pretty much no effect on file size as I got no improvement when rescaled from 720x1280 to 416x700 as we saw [**here.**](#useless)What is more I had to resort to an awful resolution of 100x250 to barely see any difference in file size, as we saw [**here**](#useless).

For some unknown reason ***to me,*** using both parameters simultaneously decreased both file size and quality drastically when encoding.

###### **Notations**

**BE MINDFUL THAT ALL THE HUSTLE OF THIS SECTION CALLED *“NOTATIONS”* WAS DONE ONLY TO *MP4* FILES AND BEFORE I FOUND OUT ABOUT A SINGLE COMMAND THAT COULD RESCALE AND DROP QUALITY ON VIDEO FILES.** ☹

* *So far,* I have *only* learnt to reduce **file size** by either using the **quantizer parameter *“which affects the quality”*** or the **resolution parameter *“which affects how many pixels are laid upon the x axis & y axis”***. Using a **constant** **bit rate** parameter was kind of complex to me because a video track may have a group of frames with a higher motion on them and the **constant bit rate** set up for plain and simple scenes would not be sufficient for high motion scenes, resulting in bad quality for those demanding scenes. Of course, there is **variable bit rate** which is better to tailor the *higher* **bit rate** at which high motion scenes are displayed but also *slow down* the **bit rate** at which simple scenes are displayed.
* From the original file “**lechita.mp4”** I transcoded/encoded the video files within the other subfolders.

A screenshot of a social media post

Description automatically generated

Of course, the video files size within those subfolders *highlighted in green and light blue* is lesser that the original file **“lechita.mp4”,** *highlighted in red*, because they were either transcoded/encoded into files with either lower resolution or quality or both.

It seemed to me that reducing the resolution reduces the file size more dramatically than dropping quality. So, upon[**dropping resolutions**](#resolution) a few times I chose ***“lechilla 416x700.mp4”*** the file *highlighted in pink* below for further optimization ***(file compression)*** within the folder above *highlighted in light blue****.*** At this point the file *highlighted in pink* below ***“lechilla 416x700.mp4”***weights 4.70MB which is still kind of heavy.

A screenshot of a cell phone

Description automatically generated

The *further optimization* only means that I would further reduce the file size of ***“lechilla 416x700.mp4”*** by [**dropping quality**](#quality)**.** All video files with maximum file size optimization are shown below, they are within a folder called ***“Maximum File Size Optimization (RES)-(QA)”.***

***A screenshot of a social media post

Description automatically generated***

* **For some reason I kept getting an error when used numbers on the input file name*.***

A picture containing computer

Description automatically generated

## How to find out the format and codecs of one file

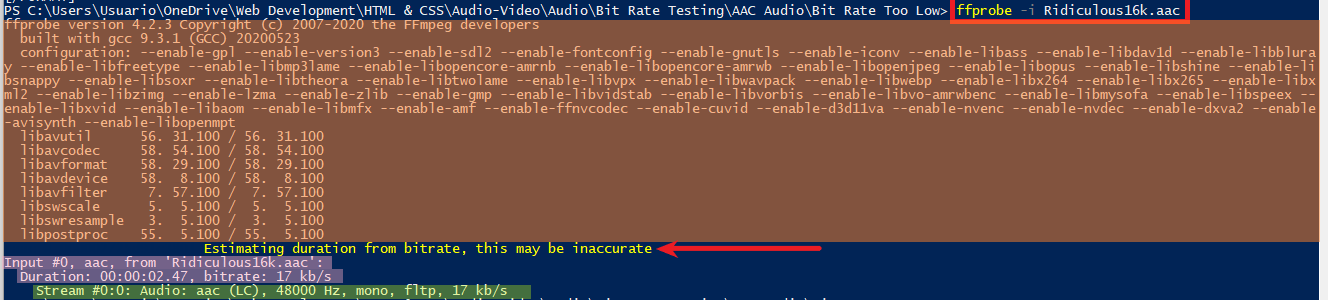
There are 2 ways I know of getting this information: Through PowerShell and VLC Player.

### FFprobe (Simple)



This command above works for *both* **Video** & **Audio** and will only show the following, although the way this information is presented differs among codecs :

* Highlighted in orange, there is the FFmpeg version, encoders compiled and working and their versions.
* Highlighted in pink, On the 1st line there is the **container** type [**(AAC)**](#its) and next to it the file´s name. On the 2nd line there is the duration of the file, **2** seconds and, the Bit Rate which is not reliable information, there is even a warning in yellow telling you that *(See the red arrow).*
* Highlighted in yellow at the bottom there is the stream/codec info which 1st shows the codec the file was encoded with **(AAC)**, the profile **(LC),** sample rate **(48000Hz)**, channel **(Mono)** and the Bit Rate at the end, [in this case **17k.**](#parra)



This example above shows what **FFprobe** outputs when the file is AAC but look at what it shows below, when the file is Opus.

* Highlighted in orange, there is the FFmpeg version, encoders compiled and working and their versions.
* Highlighted in pink, On the 1st line there is the **container** type **(OGG)** and next to it the file´s name, on the 2nd line there is the duration of the file, **2** seconds and, the Bit Rate which is the only source of that parameter when checking Opus files with FFprobe (whether Simple or Verboise).
* Highlighted in yellow, at the bottom there is the stream/codec info which 1st shows the codec the file was encoded with **(Opus)**,sample rate **(48000Hz)**, channel **(Mono)** and the Bit Rate which is missing here and showing above instead.
* Highlighted in light blue, there is only the encoder used, which only seems to be missing for AAC.

A picture containing chart

Description automatically generated

It´s worth mentioning that **Opus** was the only audio codec that didn’t show **(not even with FFmpeg verboise)** the Bit Rate parameter on the stream/codec description highlighted in yellow, but I guess that is something implicitly characteristic on the **Opus** codec or the encoder libopus.

I didn’t include examples when using **FFprobe** for the codecs **MP3** or **Vorbis** like I did above for **AAC** and **Opus**, however, neither of these 4 codecspresented the information identically when using **FFprobe.**

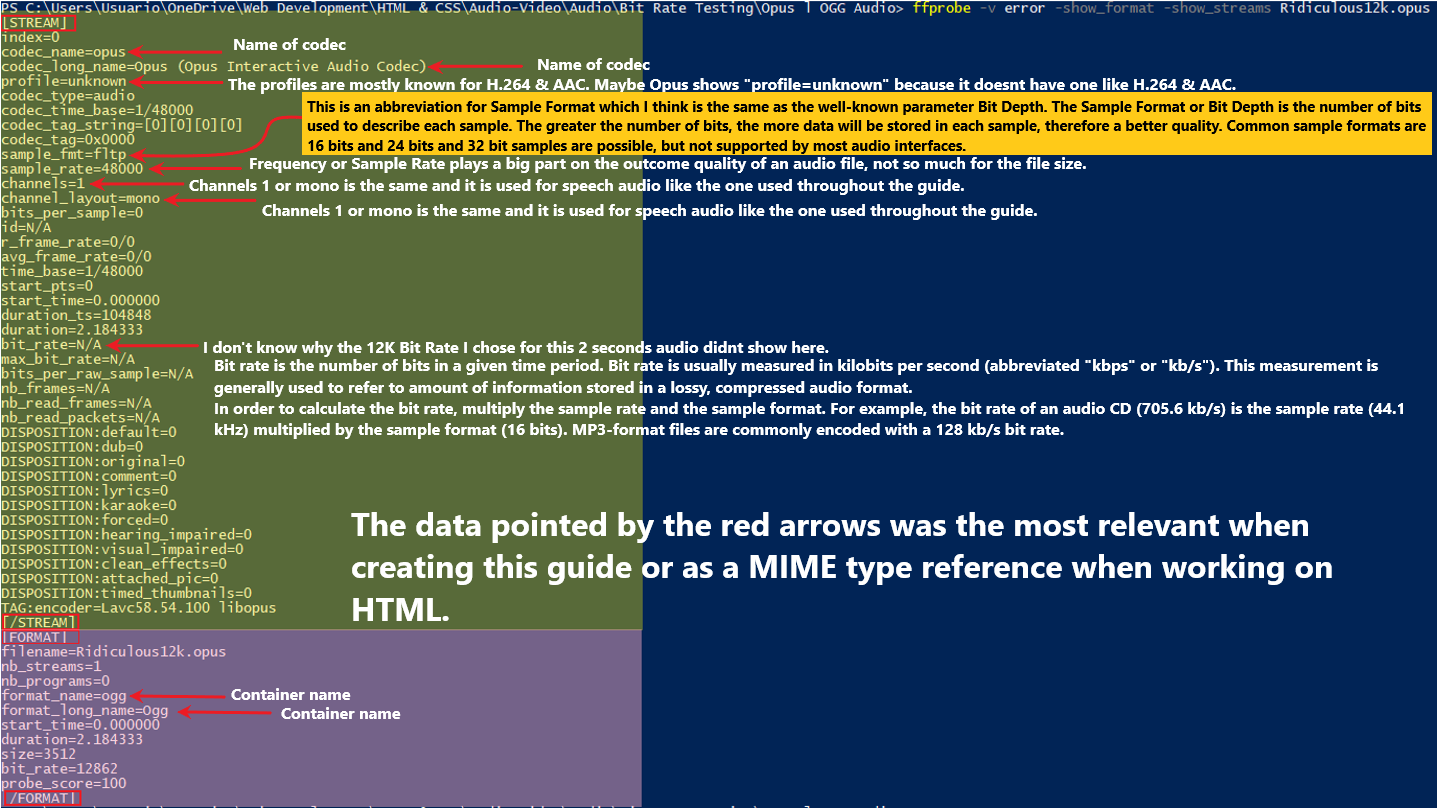
### FFprobe (Verboise)

This command below works for *both* **Video** & **Audio** files and allows to find out more about files than what the simple version of FFprobe can, besides that, it also displays the file´s info as a list which avoids confusion.



Highlighted in yellow, there is the stream/codec information which is also marked off by the **STREAM** labels and below highlighted in pink, the container information which is also marked off by the **FORMAT** labels.

#### Audio

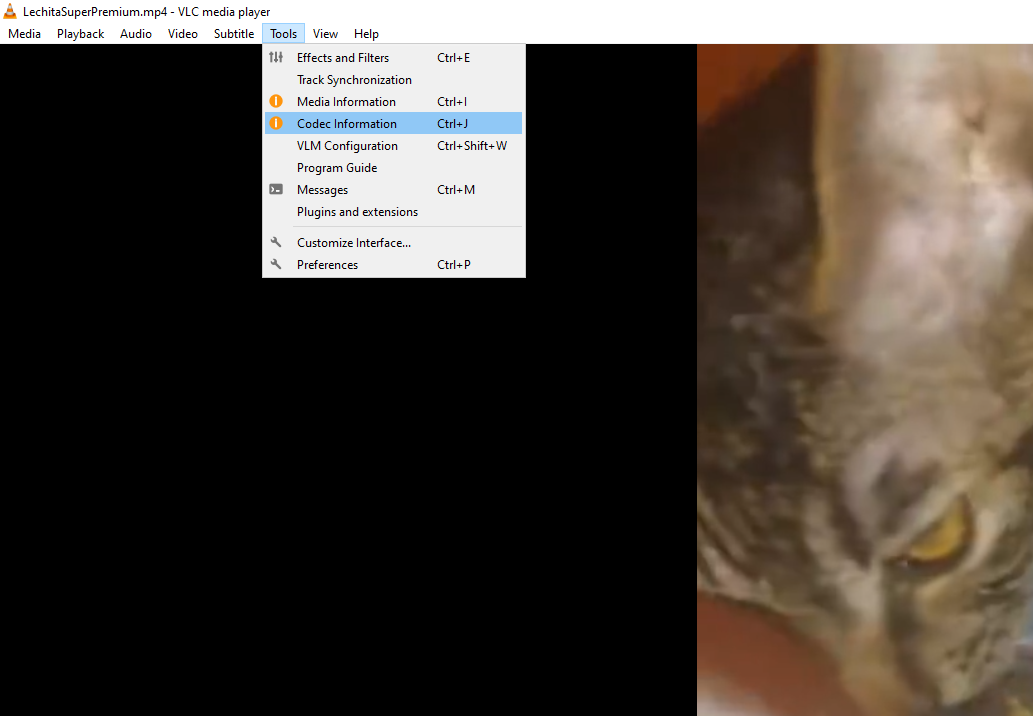


#### Video



### VLC Player

You may use this if you are clueless on how FFmpeg works.



## Audio

[PCM or Pulse-Code Modulation](https://en.wikipedia.org/wiki/Pulse-code_modulation" \t "_blank) is the general format for uncompressed audio. All recordings begin their life as soundwaves in an analog setting or dedicated recorder device. PCM converts this information into digital format by sampling that recording. You may say this is a *codec* and its containers would be **WAV & AIFF.**

Sampling rate and bit depth are used to sample the recording. The sampling rate defines how many samples are taken per second. Bit depth refers to the number of bits that are allocated to each sample.

There are two major kind of formats in audio:

### Uncompressed Formats (Codecs)

These are barely mentioned here as they can’t be used in **WEB DESIGN:**

#### WAV

#### AIFF

### Compressed Formats

Which splits into:

#### Lossless Audio Formats (Codecs)

Not really used for **WEB DESIGN** because they excel on quality that competes with *Uncompressed Formats*

#### Lossy Audio Formats (Codecs)

These are all used in **WEB** **DESIGN** for they high compression capabilities:

##### Vorbis

Normally used within a **WebM** or **OGG** video container/format. It is older and less efficient than **Opus**

##### Opus

Normally used within a **WebM** or **OGG** video container/format. It is Newer and more efficient than **Vorbis,** specially on low *Bit Rates.*

##### AAC

Normally used within a **MP4** video container/format. It is superior to **MP3,** probably as *good* as **Vorbis** but less efficient than **Opus.** Apparently, Chrome doesn’t play *main profile* **AAC** audio within a **MP4** container. I didn’t have the chance to test this as I couldn’t find out how to change the profile in which you encode with. Back on the 20s, choosing the right *profile* was extremely important due to the hardware and network bandwidth limitations back then. Specially the network bandwidth because back on the 20s, only big companies could afford high internet speeds. Processing power was also very low back then on probably most of portable media player devices and maybe a good chunk of desktop PC processors. Nowadays almost everyone has a connection that ranges from decent to excellent and even maybe the lowest tier of portable media player devices crushes the performance of the most powerful PC back then. It´s been 15 years. The only question to be made before choosing a *Profile* nowadays could be the software/hardware support for a given profile. Nowadays, seems like the most supported *profile* is **LC (Low profile).**

The only thing I cared about is the **MIME TYPES** along **its codec parameter** to add them in my **HTML** file. Luckily, the support of **LC** is good, so I don’t have to worry about support and shit.

##### MP3

Normally used within **MPEG** video container/format but it can also go within a **MP4** video container/format. It is less efficient than the 3 codecs above but remains running everywhere due to hardware and software support. The same goes for browser support because it runs on everyone.

### Audio Encoding

To start off this command is all I had to use to get an **MP3** from an **Opus** audio file ***(OGG).***

******

Remember, this guide is focused on the smallest size possible while keeping a decent quality **AND** also remember that all encoded files named ***Ridiculous*** come from an *already compressed* file that I recorded from my mobile. See [**this.**](#already)

#### **Bit Rate**

When related to audio, this refers to how much of the detail of that audio, or how much of its complexity, we can store. A higher bitrate means storing more of the complexity, and a lower bitrate means less. As you’d imagine, that naturally translates to higher quality for higher bitrates, since it stores more of the detail of the original.

##### Opus *(Best codec for speech at low Bit Rates)*

###### Speech *(For my 2 seconds audio)*

* **Opus** only supports bitrates **down to 4 Kb/s (Throughout this guide you´ll see just a “K” instead of Kb/s)**.
* With **Opus,** the lowest Bit Rate possible for my 2 seconds ***speech*** named “***Ridiculous”,*** that ensures not perceptible [***Compression artifacts***](#compre)is **12K.**
* The encoding with **Opus** is very straightforward, unlike **Vorbis,** as there is no need to worry about matching the **Bit Rate** to the **Sampling Rate**.



This command above outputs an Opus audio file of **4K** with a **Sampling Rate** of 48000 Hz *(48 kHz).*

* *Forums and web pages regarding encoding of* ***Opus*** *recommend a minimum of* ***32K*** *to keep a decent/good quality on all sorts of speech only audio files, either long or short ones. My 2 seconds audio could take* ***12K*** *without any noticeable* [***compression articles***](#compre)***.*** *However, it is very likely that happened due to the 2 seconds duration of my audio, longer speech only audio files may take higher Bit Rates.*

###### Music

I didn’t experiment with **Opus** music files but according to internet it seems that **Bit Rate**for music can be set up to about **96K** and still get a decent quality.

##### Vorbis (Not great on Web Support)

###### Speech *(For my 2 seconds audio)*

* If the [**command above**](#above) is tried, we´d get this error below.



* Simply put, Vorbis low **Bit Rates** need low [**Sampling Rates**](#KHZ)andhigh **Bit Rates** need high [**Sampling Rates**](#KHZ)**,** the following are some guidelines if you were to use **32K.**
* **44100 Hz:** Typical Audio CD sample rate. Rejected by FFmpeg for a 32k file.
* **32000 Hz:** Adequate for speech and also adequate for other audio files where a smaller file size is required with an expected small loss of quality. Rejected by FFmpeg for a 32k file.
* **22050 Hz:** Adequate for speech and usable for other audio with the expectation that there will be audio quality loss. Accepted by FFmpeg for a 32k file.
* **11025 Hz:** Very poor sound quality. Accepted by FFmpeg for a 32k file.
* **8000 Hz:** Slightly lower sampling rate than a modern telephony system, not recommended for most recording tasks. Accepted by FFmpeg for a 32k file.
* The command below includes both: **Bit Rates &** [**Sampling Rates**](#KHZ)**.** With a **Bit Rate** of **16K** *(which is the minimum for* ***Vorbis)*** I could´ve even tried a [**Sampling Rate**](#KHZ)of 8000Hz which is the minimum, but the quality dropped noticeably so I stayed with 22050 Hz.



* *Forums and web pages regarding encoding of* ***Vorbis*** *recommend a minimum of* ***32K*** *to keep a decent/good quality on all sorts of speech only audio files, either long or short ones. My 2 seconds audio could take* ***16K*** *without any noticeable* [***compression articles***](#compre)***.*** *However, it is very likely that happened due to the 2 seconds duration of my audio, longer speech only audio files may take higher Bit Rates.*

###### Music

I didn’t experiment with **Vorbis** music files but according to internet it seems that **Bit Rate**for music can be set up to about **96K** and still get a decent quality.

##### AAC

###### Speech *(For my 2 seconds audio)*

* At least for my 2 seconds audio file it doesn’t make sense to go under 24K because there is too much quality loss.
* The minimum Bit Rate for **AAC**  appears to be 1K but it doesn’t make sense as the whole audio is pretty much gone under 6K.
* As **Opus** this codec is also very straightforward to encode.
* The sweet spot for both quality and file compression was **32K**.
* *Forums and web pages regarding encoding of* ***AAC*** *recommend a minimum of* ***32K*** *to keep a decent/good quality on all sorts of speech only audio files, either long or short ones. May be out of coincidence my 2 second audio required* ***32K*** *too,**in order to sound good.*

###### Music

I didn’t experiment with **AAC** music files but according to internet it seems that **Bit Rate**for music can be set up to about **96K** and still get a decent quality.

##### MP3

###### Speech *(For my 2 seconds audio)*

* As **Opus** and **ACC** this codec is also very straightforward to encode.
* The minimum Bit Rate for **MP3**  appears to be 1K, but it is not worth it to go under 35k because of quality loss, at least for my 2 seconds audio file.
* The sweet spot for both quality and file compression was **45K**.
* At least when encoding with **CBR** *(as I did with every single audio file).*Therewas a file size cap of **8.85KB** between the Bit Rates **1K** & **36k** meaning that between those Bit Rates the file size stayed at **8.85KB.** However, the quality did change as I closed to **1KB**.The same for Bit Rates from **37K** to **44K** which output audio files with size cap of **11KB,** of course the quality dropped as I closed to **37KB** and, finally a file size cap of **13.1KB** starting at **45K.**
* *Forums and web pages regarding encoding of* ***MP3*** *recommend a minimum of* ***64K*** *to keep a decent/good quality on all sorts of speech only audio files, either long or short ones. My 2 seconds audio could take* ***45K*** *without any noticeable* [***compression articles***](#compre)***.*** *However, it is very likely that happened to the 2 seconds duration of my audio, longer speech only audio files may take higher Bit Rates.*

###### Music

I didn’t experiment with **MP3** music files but according to internet it seems that in order to get decent quality you need at least **128K**.

#### **Channels**

##### Mono

Mono should be for speech only. The original Opus audio recorded with my cellphone came with only one channel so there was no need to re-encode the channels. So, It seems that mobile recorder devices like my cellphone are set to record *speech audio* as mono by default. I guess that can be changed though.



##### Stereo

Stereo should only be for music and complex soundtracks. *Stereo files require double the bit rate (data rate) of mono files.*



#### **Sampling Rate (Frequency)**

These numbers are relative to **the number of times per second that the analog sound is “registered”, in order to be rebuilt digitally** (44.1 kHz equals 44,100 Hz samples per second).

Dropping the Sampling Rates lowers the quality noticeably but didn’t reduce the file size on every codec. In fact, codecs like:

* **MP3** and **Opus** which actually saw a file size increase of a few kilobytes when dropping Sampling Rate from 48 kHz to 22.05 kHz

Dropping Sampling Rates wasn’t that effective to optimize file size because the codecs in which it worked, like **AAC** or **Vorbis** only saw a difference of a few kilobytes as shown below.

Avoid re-encoding twice to drop Bit Rate first and then Sampling Rate, or vice versa. If you have to drop them both use the following command.



The command example above was done to re-encode **AAC** because it was the only codec along with **Vorbis** with actual size optimization when dropping Sampling Rates. The resulting file of the command above dropped the Sampling Rate from 48 kHz to 22.05 kHz and the Bit Rate to 12k which made it around 30 kilobytes smaller than the other **AAC** file which only got the Bit Rate dropped to 12k. A difference of 30 kilobytes is ridiculously small but it is still worth noticing, given that other codecs actually got a few kilobytes bigger when dropping Sampling Rates.

##### For Music

44.1 Hz stereo, unless your source material is 48 Hz stereo (in which case, use 48).

##### For Speech

22.05 kHz mono.

##### For a Mix of Music/Speech

44.1 kHz stereo, unless your source material is 48 kHz stereo (in which case, use 48).

Here is a list of legit **Sample Rates** when encoding.

* 8 kHz
* 11.025 kHz
* 12 kHz
* 16 kHz
* 22.05 kHz
* 24 kHz
* 32 kHz
* 44.1 kHz
* 48 kHz

#### **Bit Depth**

The higher the value the better and heavier the audio file will be. Values tend to be *(****16, 24, 32).*** This value kicks in during sampling through [**PCM**](#pcm).

#### **Latency**

VER LINK BELOW PARA MAS INFO

https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Audio\_codecs#Vorbis