

Chapter 2 discusses various fundamental tasks you can accomplish with the audio stream in FFmpeg.

Chapter 2

Audio Processing

2.1.1 Introduction to Transcoding

One of the basic tasks you can perform on an audio track in FFmpeg is to convert it into another format. This process known as *Transcoding*, is the direct digital-to-digital conversion of one stream encoding to another, whether video or audio. Transcoding is usually done in cases where a target device – media player such as iPod, iPad, DVD players or a software application, does not support the format or has limited storage capacity that requires a condensed file size. Transcoding can also be used to convert an incompatible or obsolete format to a better-supported format.

Transcoding is generally a “lossy process” - a data encoding method which compresses data by discarding (losing) some of it to minimize the amount of data that need to be stored in a file; however, transcoding can also be “lossless” if the input is losslessly compressed and the output is either losslessly compressed or stored in a uncompressed state. Although compression can reduce file size considerably, repeatedly performing transcoding on a single file using lossy compression can create a ‘generation loss’ – a reduction in the quality of the audio when copying, which would cause further reduction in quality on making a copy of the copy. So you need to keep this in mind while repeatedly transcoding between various formats.

Although I could not show you here the difference between an original and lossy audio compression (due to the limitation of the media of course), the following shows an example of a lossy compression in an image. The original JPG image is on the left and a lossy image of the same after repeated compression is shown on the right. As we lose precious information forever during compression, we cannot get back the original image using the compressed image.



2.1.2 Audio compression

Audio compression is a form of data compression designed to reduce the transmission bandwidth and storage requirement of a digital audio stream. Audio compression algorithms are implemented in software as audio codec's, - which is a software program or library capable of encoding/decoding a digital audio stream.

Audio compression is either lossy or lossless as discussed earlier. Lossless audio compression produces a version of digital audio that can be decoded to an exact digital duplicate of the original audio stream. This is in contrast to the irreversible changes upon playback from lossy compression techniques such as Vorbis and MP3.



Bitrates are explained in Chapter 1.

The whole idea behind audio compression in FFmpeg is to lower the audio bitrate (96kbps, 128kbps, 192 kbps etc.), this effectively also reduces the fidelity or quality of the audio. So you want to keep in mind that, a high bitrate audio file confirms a better sound quality, so by lowering its bitrate you are actually degrading the quality.

For normal computer use, the 128kbs rate produces a quality equal to that of an audio CD. But in the case of an MP3 use, it is necessary to use a 256kbs bitrate to reach an identical result to that of the CD quality sound.

2.1.3 Getting your audio file ready

Now that we have gone through a short introduction to compression, we will now work on the process of transcoding audio files.

To run the example commands in this section, you will need an audio file in a **.wav** or an **.mp3** format. You can get hold of a wav file by ripping an audio track from a music CD or downloading an mp3 file from the Internet. Call the resulting file 'myaudio.mp3'. For this section I used the 'Solo Piano 7' Opening file from <http://www.archive.org/details/solo-piano-7>.

Next, we will get ffmpeg to identify the file. This will tell us the various details of the audio file. The simple way to get this information is to just tell ffmpeg to use it for input. For this we need to use the *-i* option. Enter the following command at your prompt.



```
ffmpeg -i myaudio.mp3
```

The exact output on my PC is shown below; which may differ from yours depending on the version of ffmpeg you are using.

```
D:\ffmpeg>ffmpeg -i myaudio.mp3

ffmpeg version N-31100-g9251942, Copyright (c) 2000-2011 the FFmpeg
developers
Input #0, mp3, from 'myaudio.mp3':
  Metadata:
    album       : solo piano 7
    artist      : Torley
    album_artist : Torley
    composer    : Torley
    genre       : Piano
    track       : 001/176
    title       : 001 – Openings
    date        : 2008
  Duration: 00:01:39.50, start: 0.000000, bitrate: 193 kb/s
  Stream #0.0: Audio: mp3, 44100 Hz, stereo, s16, 192 kb/s
At least one output file must be specified
```

There is a lot of information we can gather from the output - the track is 1 minute 39.50 seconds long, the bitrate is 193kb/s, the audio is encoded in mp3 format at 44100Hz (44.1KHz) and has two channels (stereo). All this information will come in handy during a transcoding process.

2.1.4 Transcoding to a different format

Let us now convert the downloaded file to a simple wav format. Notice that we have not specified any format option or flag, just the complete output filename. FFmpeg automatically guesses which encoders to use by noticing the format of the input and output files, this can be a big help if you keep forgetting the option name or are just being lazy. If you are not going to specify the encoder format, make sure you mention the full filename, along with the appropriate format extension.



```
ffmpeg -i myaudio.mp3 myaudio.wav
```

The output of the command is shown below.

```
ffmpeg version N-31100-g9251942, Copyright (c) 2000-2011 the FFmpeg
developers
Input #0, mp3, from 'myaudio.mp3':
  Metadata:
    album       : solo piano 7
    artist      : Torley
    album_artist : Torley
    composer    : Torley
    genre       : Piano
    track       : 001/176
    title       : 001 - Openings
    date        : 2008
  Duration: 00:01:39.50, start: 0.000000, bitrate: 193 kb/s
  Stream #0.0: Audio: mp3, 44100 Hz, stereo, s16, 192 kb/s
File 'myaudio.wav' already exists. Overwrite ? [y/N] y
Output #0, wav, to 'myaudio.wav':
  Metadata:
    album       : solo piano 7
    artist      : Torley
    album_artist : Torley
    composer    : Torley
    genre       : Piano
    track       : 001/176
    title       : 001 - Openings
    date        : 2008
    encoder     : Lavf53.4.0
  Stream #0.0: Audio: pcm_s16le, 44100 Hz, stereo, s16, 1411 kb/s
Stream mapping:
  Stream #0.0 -> #0.0
Press [q] to stop, [?] for help
size= 17141kB time=00:01:39.50 bitrate=1411.2kbits/s
video:0kB audio:17141kB global headers:0kB muxing overhead 0.000251%
```

Notice how large the resulting wav file is (17 Mb) as compared to the original mp3 format (2.1 Mb). This being for the reason that the wav file is not compressed like its mp3 counterpart. Incidentally, the audio format of the wav is Pulse-code modulation (PCM), technically *PCM signed 16 bit little-endian* format.

As you can see from the screenshot above the output of an ffmpeg command is quite large, so from here on I'll just specify the command and do away with the output screen unless it is required for explanation.

2.1.5 Changing the bitrate of the audio

As we learned in Chapter 1, bitrates control the file size and the quality of an audio or video stream. Lowering the bitrate will result not only in a reduced file size but also diminish the quality of the final output. This can be required if you have a high quality audio recording and need to lower the quality for a reduced file size to stream over the Web. For example the following command will set the bitrate of the mp3 file to 64kb/s. This uses the `-ab` option to the job.

`-ab <value>`



```
ffmpeg -i myaudio.mp3 -ab 64k out.mp3
```

The higher the value the better is the audio quality. This is one of the important factors responsible for the audio quality. But that doesn't mean you can make a poor audio file sound better by increasing its bitrate. The resultant file will just be of bigger size.

Another example - to transcode an mp3 file to an AAC format, with a bitrate of 128K, we can use the following.



```
ffmpeg -i myaudio.mp3 -ab 128k myaudio.aac
```

As we saw earlier the original audio track has 2 channels (stereo). Many times it is not necessary to have 2 channels, like in a speech recording, where its really doesn't matter.. In such cases you can further reduce the file size by setting the audio channels to mono or '1'. For output streams it is set by default to the number of input audio channels.

`-ac <value>`



```
ffmpeg -i myaudio.mp3 -ac 1 out.mp3
```

Note that once you convert a stereo channel to a mono, you cannot convert it back to a stereo channel audio. That information is lost forever. The same thing happens with bitrates. Once you reduce a bitrate of an audio file, you cannot just increase the bitrate back again to get the original quality. That information is already gone. So as a precaution, *never work* with your original media files. Make a copy of the original and work with the copy.

The other important audio option is `-acodec`. This option lets you choose the type of audio codec you want to use. e.g. if you are using `ffmpeg` on a `mp3` file, then it will need the audio codec `libmp3lame`. You can specify it using `-acodec libmp3lame`. Although, by default, `ffmpeg` takes care of the codecs you need (by guessing it from the output file format) but if you need anything different, then go for this option. `FFmpeg` uses a default encoder for each audio stream, using the output file extension to guess the encoder to use. This option lets you force `FFmpeg` to use a specific audio encoder rather than the default. The following for example will extract the audio stream from a `.flv` video and save it as an `.mp3` file using the `libmp3lame` encoder.

`-acodec <encoder/decoder>`



```
ffmpeg -i myvideo.flv -acodec libmp3lame myaudio.mp3
```

Sometime you `FFmpeg` may be unable to correctly decode the input file, giving the error something like the following.

Error while decoding stream #0.0

In such cases you can force `FFmpeg` to use a particular decoder to decode the input file. The following example will force `FFmpeg` to use the `mp3` codec to decode the input file audio.



```
ffmpeg -acodec libmp3lame -i myvideo.flv myaudio.mp3
```

Note that the `-acodec` option comes before the `-i` option when we want the codec to apply to the input stream and comes after the `-i` option when we want the codec to apply to the output stream. To see what codecs are available on your system, issue the following command.



```
ffmpeg -codecs
```

Sometimes you may want to completely disable the audio recording for which we can use the `-an` option. This can be used to strip out an audio stream from a video file. When you use this option, all the other audio related attributes are cancelled out, which is fine, as they would not matter without the audio. So for example you are want to disable the audio from a video file and only copy the video stream, you can use the following.



```
ffmpeg -i myvideo.flv -an out.flv
```

Another important option is `-ar`, the audio sampling frequency. This lets you set the maximum sampling frequency of the audio stream. Audio sampling was discussed in Chapter 1. You can use the option to reduce the sampling frequency to a lower value to reduce file storage or Internet bandwidth capacity. The default value is set at 44100Hz. The value is given in Hz. So the following will resample the input audio to 11025Hz with a single channel (mono).

`-ar <value in Hz>`



```
ffmpeg -i myaudio.mp3 -ar 11025 -ac 1 myaudio.mp3
```

Note that once you have reduced the sampling frequency some of the audio data is lost. You cannot again resample it to a higher value and expect increase in the audio quality.

2.1.6 Audio grabbing

Until now we have looked into how to transform existing audio stream into other formats. FFmpeg can also grab audio from external devices such as a microphone. This can be useful if you need to record from your desktop microphone or create a screencast. Note that the following command will not work on a Windows machine. You need to have a Linux machine to correctly grab the mic audio. Enter the following command at your Linux prompt.



```
ffmpeg -f oss -i /dev/dsp ./audio.wav
```

This will start recording the input audio from the mic to the ‘audio.wav’ file. Once started you will need to press ‘q’ to stop the recording. We will now look into the various options given above.

The option `-f` denotes the format to be used for the input stream. There are various formats FFmpeg supports; you can find the complete list by issuing the following command.



```
ffmpeg -formats
```

Here we are using the 'oss' format, which stands for Open Sound System input device. The Open Sound System (OSS) is an interface for making and capturing sound in Unix or Unix-like operating systems. In the Linux kernel, there have historically been two uniform sound APIs. One is OSS; the other is ALSA (*Advanced Linux Sound Architecture*). ALSA is available for Linux only.

The device '/dev/dsp' is the default audio input device in the Linux system. It's connected to the main speakers and the primary recording source such as a microphone. The system administrator can set /dev/dsp to be a symbolic link to the desired default device.

The 'audio.wav' file is where the recorded audio will be saved.

Another example - the following will record the mic audio to the file 'rec.flac' in the current directory, this is a flac format file.



```
ffmpeg -f alsa -ar 48000 -i front ./rec.flac
```

.AAC Advanced Audio Coding File - declared the new audio-file standard in 1997, designed to replace its predecessor, MP3. It provides better quality at lower bit rates, and its Apple's standard iTunes and iPod audio format.

.AIF(F) Audio Interchange File Format - developed by Electronic Arts and Apple back in the '80s. AIFF files contain uncompressed audio, resulting in large file sizes.

.m4a Apple Lossless - This file format uses lossless compressions for digital music.

.MP3 MPEG Layer 3 - the most popular digital-audio music format, designed by a team of European engineers in 1991 to conserve the quality of a song while storing it in a small, compact file.

.OGG Ogg Vorbis - one of the most popular license-free, open-source audio-compression formats. It's efficient for streaming and compression because it creates smaller files than MP3 while maintaining audio quality.

.RA(M) Real Audio Media - developed by RealNetworks in 1995. It has a wide variety of uses, from videos to music, but is mainly used for streaming audio such as that from Internet radio stations.

.WAV Windows WAVE - IBM and Microsoft-developed format popular audio format among PC computer users; it can hold both compressed and uncompressed audio.

.WMA Windows Media Audio - designed by Microsoft to be an MP3 competitor, but with the introduction of iTunes and iPods, it's fallen far behind MP3 in popularity.

2.3

Audio processing recipes

MP3 to AAC High Quality Stereo



```
ffmpeg -i in.mp3 -acodec aac -ac 2 -ar 48000 -ab 192k out.aac
```

MP3 to AAC High Quality 5.1



```
ffmpeg -i in.mp3 -acodec aac -ac 6 -ar 48000 -ab 448k out.aac
```

Convert to low quality mp3 to preserve storage



```
ffmpeg -i in.mp3 -ab 64K out.mp3
```

MP3 to Vorbis OGG (can be played in HTML 5)



```
ffmpeg -i in.mp3 -acodec vorbis -aq 50 out.ogg
```