# ffmpeg Documentation

## Table of Contents



# 1 Synopsis# TOC

ffmpeg [*global_options*] {[*input_file_options*] -i `input_url`} ... {[*output_file_options*] `output_url`} ...

# 2 Description# TOC

`ffmpeg` is a very fast video and audio converter that can also grab from a live audio/video source. It can also convert between arbitrary sample rates and resize video on the fly with a high quality polyphase filter.

`ffmpeg` reads from an arbitrary number of input "files" (which can be regular files, pipes, network streams, grabbing devices, etc.), specified by the `-i` option, and writes to an arbitrary number of output "files", which are specified by a plain output url. Anything found on the command line which cannot be interpreted as an option is considered to be an output url.

Each input or output url can, in principle, contain any number of streams of different types (video/audio/subtitle/attachment/data). The allowed number and/or types of streams may be limited by the container format. Selecting which streams from which inputs will go into which output is either done automatically or with the `-map` option (see the Stream selection chapter).

To refer to input files in options, you must use their indices (0-based). E.g. the first input file is `0`, the second is `1`, etc. Similarly, streams within a file are referred to by their indices. E.g. `2:3` refers to the fourth stream in the third input file. Also see the Stream specifiers chapter.

As a general rule, options are applied to the next specified file. Therefore, order is important, and you can have the same option on the command line multiple times. Each occurrence is then applied to the next input or output file. Exceptions from this rule are the global options (e.g. verbosity level), which should be specified first.

Do not mix input and output files – first specify all input files, then all output files. Also do not mix options which belong to different files. All options apply ONLY to the next input or output file and are reset between files.

- To set the video bitrate of the output file to 64 kbit/s:

  ```
  ffmpeg -i input.avi -b:v 64k -bufsize 64k output.avi
  ```

- To force the frame rate of the output file to 24 fps:

  ```
  ffmpeg -i input.avi -r 24 output.avi
  ```

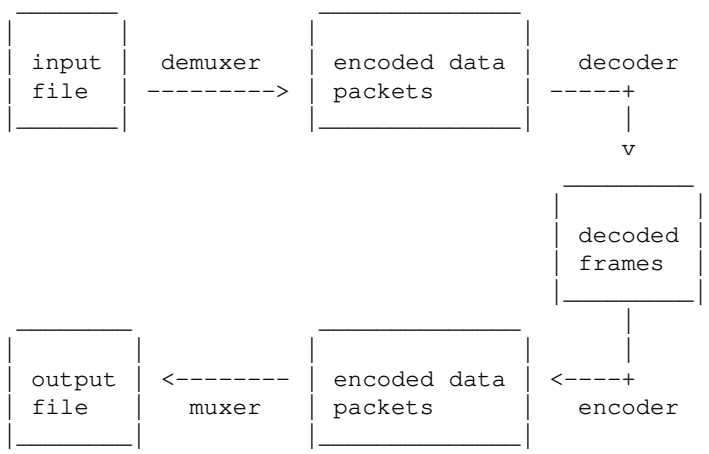
- To force the frame rate of the input file (valid for raw formats only) to 1 fps and the frame rate of the output file to 24 fps:

  ```
  ffmpeg -r 1 -i input.m2v -r 24 output.avi
  ```

The format option may be needed for raw input files.

# 3 Detailed description# TOC

The transcoding process in `ffmpeg` for each output can be described by the following diagram:




`ffmpeg` calls the libavformat library (containing demuxers) to read input files and get packets containing encoded data from them. When there are multiple input files, `ffmpeg` tries to keep them synchronized by tracking lowest timestamp on any active input stream.

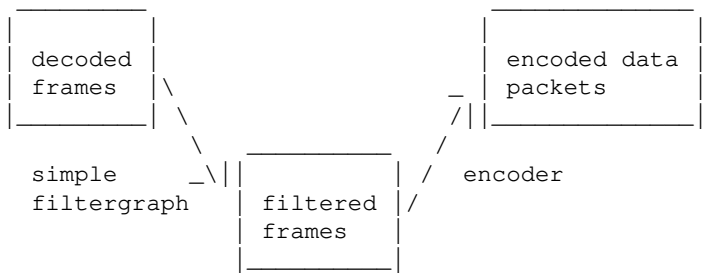
Encoded packets are then passed to the decoder (unless streamcopy is selected for the stream, see further for a description). The decoder produces uncompressed frames (raw video/PCM audio/...) which can be processed further by filtering (see next section). After filtering, the frames are passed to the encoder, which encodes them and outputs encoded packets. Finally those are passed to the muxer, which writes the encoded packets to the output file.

# 3.1 Filtering# TOC

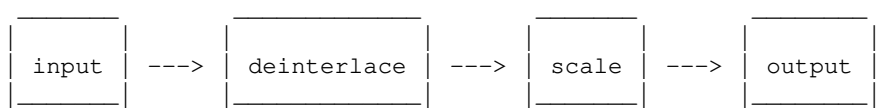
Before encoding, `ffmpeg` can process raw audio and video frames using filters from the libavfilter library. Several chained filters form a filter graph. `ffmpeg` distinguishes between two types of filtergraphs: simple and complex.

## 3.1.1 Simple filtergraphs# TOC

Simple filtergraphs are those that have exactly one input and output, both of the same type. In the above diagram they can be represented by simply inserting an additional step between decoding and encoding:



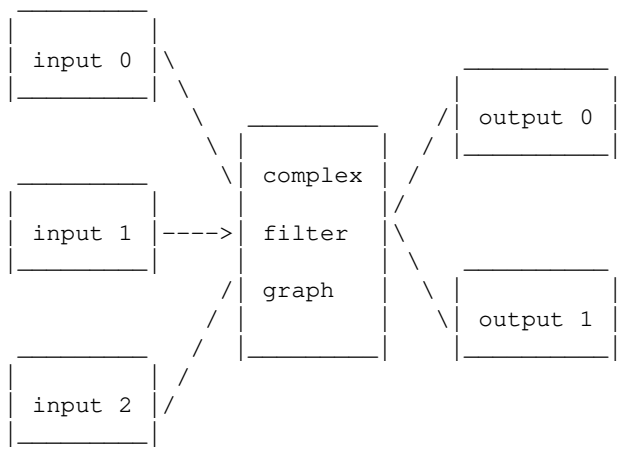
Simple filtergraphs are configured with the per-stream `-filter` option (with `-vf` and `-af` aliases for video and audio respectively). A simple filtergraph for video can look for example like this:




Note that some filters change frame properties but not frame contents. E.g. the `fps` filter in the example above changes number of frames, but does not touch the frame contents. Another example is the `setpts` filter, which only sets timestamps and otherwise passes the frames unchanged.

### 3.1.2 Complex filtergraphs# TOC

Complex filtergraphs are those which cannot be described as simply a linear processing chain applied to one stream. This is the case, for example, when the graph has more than one input and/or output, or when output stream type is different from input. They can be represented with the following diagram:



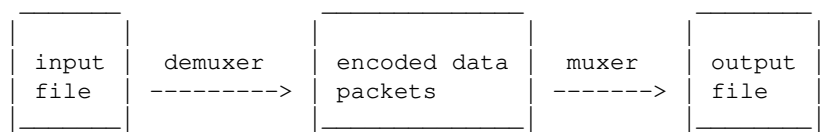

Complex filtergraphs are configured with the `-filter_complex` option. Note that this option is global, since a complex filtergraph, by its nature, cannot be unambiguously associated with a single stream or file.

The `-lavfi` option is equivalent to `-filter_complex`.

A trivial example of a complex filtergraph is the `overlay` filter, which has two video inputs and one video output, containing one video overlaid on top of the other. Its audio counterpart is the `amix` filter.

## 3.2 Stream copy# TOC

Stream copy is a mode selected by supplying the `copy` parameter to the `-codec` option. It makes `ffmpeg` omit the decoding and encoding step for the specified stream, so it does only demuxing and muxing. It is useful for changing the container format or modifying container-level metadata. The diagram above will, in this case, simplify to this:




Since there is no decoding or encoding, it is very fast and there is no quality loss. However, it might not work in some cases because of many factors. Applying filters is obviously also impossible, since filters work on uncompressed data.

# 4 Stream selection# TOC

By default, `ffmpeg` includes only one stream of each type (video, audio, subtitle) present in the input files and adds them to each output file. It picks the "best" of each based upon the following criteria: for video, it is the stream with the highest resolution, for audio, it is the stream with the most channels, for subtitles, it is the first subtitle stream. In the case where several streams of the same type rate equally, the stream with the lowest index is chosen.

You can disable some of those defaults by using the `-vn/-an/-sn/-dn` options. For full manual control, use the `-map` option, which disables the defaults just described.

# 5 Options# TOC

All the numerical options, if not specified otherwise, accept a string representing a number as input, which may be followed by one of the SI unit prefixes, for example: 'K', 'M', or 'G'.

If 'i' is appended to the SI unit prefix, the complete prefix will be interpreted as a unit prefix for binary multiples, which are based on powers of 1024 instead of powers of 1000. Appending 'B' to the SI unit prefix multiplies the value by 8. This allows using, for example: 'KB', 'MiB', 'G' and 'B' as number suffixes.

Options which do not take arguments are boolean options, and set the corresponding value to true. They can be set to false by prefixing the option name with "no". For example using "-nofoo" will set the boolean option with name "foo" to false.

## 5.1 Stream specifiers# TOC

Some options are applied per-stream, e.g. bitrate or codec. Stream specifiers are used to precisely specify which stream(s) a given option belongs to.

A stream specifier is a string generally appended to the option name and separated from it by a colon. E.g. `-codec:a:1 ac3` contains the `a:1` stream specifier, which matches the second audio stream. Therefore, it would select the ac3 codec for the second audio stream.

A stream specifier can match several streams, so that the option is applied to all of them. E.g. the stream specifier in `-b:a 128k` matches all audio streams.

An empty stream specifier matches all streams. For example, `-codec copy` or `-codec: copy` would copy all the streams without reencoding.

Possible forms of stream specifiers are:

*stream_index*

> Matches the stream with this index. E.g. `-threads:1 4` would set the thread count for the second stream to 4.

*stream_type*[:*stream_index*]

> *stream_type* is one of following: 'v' or 'V' for video, 'a' for audio, 's' for subtitle, 'd' for data, and 't' for attachments. 'v' matches all video streams, 'V' only matches video streams which are not attached pictures, video thumbnails or cover arts. If *stream_index* is given, then it matches stream number *stream_index* of this type. Otherwise, it matches all streams of this type.

p:*program_id*[:*stream_index*]

> If *stream_index* is given, then it matches the stream with number *stream_index* in the program with the id *program_id*. Otherwise, it matches all streams in the program.

#*stream_id* or i:*stream_id*

> Match the stream by stream id (e.g. PID in MPEG-TS container).

m:*key*[:*value*]

> Matches streams with the metadata tag *key* having the specified value. If *value* is not given, matches streams that contain the given tag with any value.

u

> Matches streams with usable configuration, the codec must be defined and the essential information such as video dimension or audio sample rate must be present.
>
> Note that in `ffmpeg`, matching by metadata will only work properly for input files.

## 5.2 Generic options# TOC

These options are shared amongst the ff* tools.

-L

> Show license.

-h, -?, -help, --help [*arg*]

Show help. An optional parameter may be specified to print help about a specific item. If no argument is specified, only basic (non advanced) tool options are shown.

Possible values of *arg* are:

`long`

Print advanced tool options in addition to the basic tool options.

`full`

Print complete list of options, including shared and private options for encoders, decoders, demuxers, muxers, filters, etc.

`decoder=`*`decoder_name`*

Print detailed information about the decoder named *decoder_name*. Use the `-decoders` option to get a list of all decoders.

`encoder=`*`encoder_name`*

Print detailed information about the encoder named *encoder_name*. Use the `-encoders` option to get a list of all encoders.

`demuxer=`*`demuxer_name`*

Print detailed information about the demuxer named *demuxer_name*. Use the `-formats` option to get a list of all demuxers and muxers.

`muxer=`*`muxer_name`*

Print detailed information about the muxer named *muxer_name*. Use the `-formats` option to get a list of all muxers and demuxers.

`filter=`*`filter_name`*

Print detailed information about the filter name *filter_name*. Use the `-filters` option to get a list of all filters.

`-version`

Show version.

`-formats`

Show available formats (including devices).

`-demuxers`

Show available demuxers.

`-muxers`

Show available muxers.

`-devices`

Show available devices.

`-codecs`

Show all codecs known to libavcodec.

Note that the term 'codec' is used throughout this documentation as a shortcut for what is more correctly called a media bitstream format.

`-decoders`

Show available decoders.

`-encoders`

Show all available encoders.

`-bsfs`

Show available bitstream filters.

`-protocols`

Show available protocols.

`-filters`

Show available libavfilter filters.

`-pix_fmts`

Show available pixel formats.

`-sample_fmts`

Show available sample formats.

`-layouts`

Show channel names and standard channel layouts.

`-colors`

Show recognized color names.

`-sources` *`device`*`[,`*`opt1`*`=`*`val1`*`[,`*`opt2`*`=`*`val2`*`]...]`

Show autodetected sources of the input device. Some devices may provide system-dependent source names that cannot be autodetected. The returned list cannot be assumed to be always complete.

```
ffmpeg -sources pulse,server=192.168.0.4
```

`-sinks` *`device`*`[,`*`opt1`*`=`*`val1`*`[,`*`opt2`*`=`*`val2`*`]...]`

Show autodetected sinks of the output device. Some devices may provide system-dependent sink names that cannot be autodetected. The returned list cannot be assumed to be always complete.

```
ffmpeg -sinks pulse,server=192.168.0.4
```

`-loglevel [repeat+]`*`loglevel`* | `-v [repeat+]`*`loglevel`*

Set the logging level used by the library. Adding "repeat+" indicates that repeated log output should not be compressed to the first line and the "Last message repeated n times" line will be omitted. "repeat" can also be used alone. If "repeat" is used alone, and with no prior loglevel set, the default loglevel will be used. If multiple loglevel parameters are given, using 'repeat' will not change the loglevel. *loglevel* is a string or a number containing one of the following values:

`'quiet, -8'`

Show nothing at all; be silent.

`'panic, 0'`

Only show fatal errors which could lead the process to crash, such as an assertion failure. This is not currently used for anything.

`'fatal, 8'`

Only show fatal errors. These are errors after which the process absolutely cannot continue.

`'error, 16'`

Show all errors, including ones which can be recovered from.

`'warning, 24'`

Show all warnings and errors. Any message related to possibly incorrect or unexpected events will be shown.

`'info, 32'`

Show informative messages during processing. This is in addition to warnings and errors. This is the default value.

‘verbose, 40’

Same as info, except more verbose.

‘debug, 48’

Show everything, including debugging information.

‘trace, 56’

By default the program logs to stderr. If coloring is supported by the terminal, colors are used to mark errors and warnings. Log coloring can be disabled setting the environment variable AV_LOG_FORCE_NOCOLOR or NO_COLOR, or can be forced setting the environment variable AV_LOG_FORCE_COLOR. The use of the environment variable NO_COLOR is deprecated and will be dropped in a future FFmpeg version.

-report

Dump full command line and console output to a file named *program-YYYYMMDD-HHMMSS*.log in the current directory. This file can be useful for bug reports. It also implies -loglevel verbose.

Setting the environment variable FFREPORT to any value has the same effect. If the value is a ’:’-separated key=value sequence, these options will affect the report; option values must be escaped if they contain special characters or the options delimiter ’:’ (see the “Quoting and escaping” section in the ffmpeg-utils manual).

The following options are recognized:

file

set the file name to use for the report; %p is expanded to the name of the program, %t is expanded to a timestamp, %% is expanded to a plain %

level

set the log verbosity level using a numerical value (see -loglevel).

For example, to output a report to a file named ffreport.log using a log level of 32 (alias for log level info):

```
FFREPORT=file=ffreport.log:level=32 ffmpeg -i input output
```

Errors in parsing the environment variable are not fatal, and will not appear in the report.

`-hide_banner`

Suppress printing banner.

All FFmpeg tools will normally show a copyright notice, build options and library versions. This option can be used to suppress printing this information.

`-cpuflags flags (global)`

Allows setting and clearing cpu flags. This option is intended for testing. Do not use it unless you know what you're doing.

```
ffmpeg -cpuflags -sse+mmx ...
ffmpeg -cpuflags mmx ...
ffmpeg -cpuflags 0 ...
```

Possible flags for this option are:

- 'x86'
  - 'mmx'
  - 'mmxext'
  - 'sse'
  - 'sse2'
  - 'sse2slow'
  - 'sse3'
  - 'sse3slow'
  - 'ssse3'
  - 'atom'
  - 'sse4.1'
  - 'sse4.2'
  - 'avx'
  - 'avx2'
  - 'xop'
  - 'fma3'
  - 'fma4'
  - '3dnow'
  - '3dnowext'
  - 'bmi1'
  - 'bmi2'
  - 'cmov'
- 'ARM'
  - 'armv5te'
  - 'armv6'
  - 'armv6t2'
  - 'vfp'
  - 'vfpv3'

‘neon’
‘setend’
‘AArch64’
‘armv8’
‘vfp’
‘neon’
‘PowerPC’
‘altivec’
‘Specific Processors’
‘pentium2’
‘pentium3’
‘pentium4’
‘k6’
‘k62’
‘athlon’
‘athlonxp’
‘k8’

`-opencl_bench`

This option is used to benchmark all available OpenCL devices and print the results. This option is only available when FFmpeg has been compiled with `--enable-opencl`.

When FFmpeg is configured with `--enable-opencl`, the options for the global OpenCL context are set via `-opencl_options`. See the "OpenCL Options" section in the ffmpeg-utils manual for the complete list of supported options. Amongst others, these options include the ability to select a specific platform and device to run the OpenCL code on. By default, FFmpeg will run on the first device of the first platform. While the options for the global OpenCL context provide flexibility to the user in selecting the OpenCL device of their choice, most users would probably want to select the fastest OpenCL device for their system.

This option assists the selection of the most efficient configuration by identifying the appropriate device for the user’s system. The built-in benchmark is run on all the OpenCL devices and the performance is measured for each device. The devices in the results list are sorted based on their performance with the fastest device listed first. The user can subsequently invoke `ffmpeg` using the device deemed most appropriate via `-opencl_options` to obtain the best performance for the OpenCL accelerated code.

Typical usage to use the fastest OpenCL device involve the following steps.

Run the command:

```
ffmpeg -opencl_bench
```

Note down the platform ID (*pidx*) and device ID (*didx*) of the first i.e. fastest device in the list. Select the platform and device using the command:

```
ffmpeg -opencl_options platform_idx=pidx:device_idx=didx ...
```

`-opencl_options options (global)`

Set OpenCL environment options. This option is only available when FFmpeg has been compiled with `--enable-opencl`.

*options* must be a list of *key*=*value* option pairs separated by ':'. See the "OpenCL Options" section in the ffmpeg-utils manual for the list of supported options.

## 5.3 AVOptions# TOC

These options are provided directly by the libavformat, libavdevice and libavcodec libraries. To see the list of available AVOptions, use the `-help` option. They are separated into two categories:

`generic`

These options can be set for any container, codec or device. Generic options are listed under AVFormatContext options for containers/devices and under AVCodecContext options for codecs.

`private`

These options are specific to the given container, device or codec. Private options are listed under their corresponding containers/devices/codecs.

For example to write an ID3v2.3 header instead of a default ID3v2.4 to an MP3 file, use the `id3v2_version` private option of the MP3 muxer:

```
ffmpeg -i input.flac -id3v2_version 3 out.mp3
```

All codec AVOptions are per-stream, and thus a stream specifier should be attached to them.

Note: the `-nooption` syntax cannot be used for boolean AVOptions, use `-option 0`/`-option 1`.

Note: the old undocumented way of specifying per-stream AVOptions by prepending v/a/s to the options name is now obsolete and will be removed soon.

## 5.4 Main options# TOC

`-f fmt (input/output)`

Force input or output file format. The format is normally auto detected for input files and guessed from the file extension for output files, so this option is not needed in most cases.

`-i url (input)`

input file url

`-y` (*global*)

Overwrite output files without asking.

`-n` (*global*)

Do not overwrite output files, and exit immediately if a specified output file already exists.

`-stream_loop` *number* (*input*)

Set number of times input stream shall be looped. Loop 0 means no loop, loop -1 means infinite loop.

`-c[:`*stream_specifier*`]` *codec* (*input/output,per-stream*)
`-codec[:`*stream_specifier*`]` *codec* (*input/output,per-stream*)

Select an encoder (when used before an output file) or a decoder (when used before an input file) for one or more streams. *codec* is the name of a decoder/encoder or a special value `copy` (output only) to indicate that the stream is not to be re-encoded.

For example

```
ffmpeg -i INPUT -map 0 -c:v libx264 -c:a copy OUTPUT
```

encodes all video streams with libx264 and copies all audio streams.

For each stream, the last matching `c` option is applied, so

```
ffmpeg -i INPUT -map 0 -c copy -c:v:1 libx264 -c:a:137 libvorbis OUTPUT
```

will copy all the streams except the second video, which will be encoded with libx264, and the 138th audio, which will be encoded with libvorbis.

`-t` *duration* (*input/output*)

When used as an input option (before `-i`), limit the *duration* of data read from the input file.

When used as an output option (before an output url), stop writing the output after its duration reaches *duration*.

*duration* must be a time duration specification, see (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual.

-to and -t are mutually exclusive and -t has priority.

`-to` *position* (*output*)

Stop writing the output at *position*. *position* must be a time duration specification, see (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual.

-to and -t are mutually exclusive and -t has priority.

`-fs` *`limit_size`* (*`output`*)

Set the file size limit, expressed in bytes. No further chunk of bytes is written after the limit is exceeded. The size of the output file is slightly more than the requested file size.

`-ss` *`position`* (*`input/output`*)

When used as an input option (before `-i`), seeks in this input file to *position*. Note that in most formats it is not possible to seek exactly, so `ffmpeg` will seek to the closest seek point before *position*. When transcoding and `-accurate_seek` is enabled (the default), this extra segment between the seek point and *position* will be decoded and discarded. When doing stream copy or when `-noaccurate_seek` is used, it will be preserved.

When used as an output option (before an output url), decodes but discards input until the timestamps reach *position*.

*position* must be a time duration specification, see (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual.

`-sseof` *`position`* (*`input/output`*)

Like the `-ss` option but relative to the "end of file". That is negative values are earlier in the file, 0 is at EOF.

`-itsoffset` *`offset`* (*`input`*)

Set the input time offset.

*offset* must be a time duration specification, see (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual.

The offset is added to the timestamps of the input files. Specifying a positive offset means that the corresponding streams are delayed by the time duration specified in *offset*.

`-timestamp` *`date`* (*`output`*)

Set the recording timestamp in the container.

*date* must be a date specification, see (ffmpeg-utils)the Date section in the ffmpeg-utils(1) manual.

`-metadata[:metadata_specifier]` *`key=value`* (*`output,per-metadata`*)

Set a metadata key/value pair.

An optional *metadata_specifier* may be given to set metadata on streams, chapters or programs. See `-map_metadata` documentation for details.

This option overrides metadata set with `-map_metadata`. It is also possible to delete metadata by using an empty value.

For example, for setting the title in the output file:

```
ffmpeg -i in.avi -metadata title="my title" out.flv
```

To set the language of the first audio stream:

```
ffmpeg -i INPUT -metadata:s:a:0 language=eng OUTPUT
```

`-disposition[:stream_specifier]` *`value`* (*`output`*,*`per-stream`*)

Sets the disposition for a stream.

This option overrides the disposition copied from the input stream. It is also possible to delete the disposition by setting it to 0.

The following dispositions are recognized:

```
default
dub
original
comment
lyrics
karaoke
forced
hearing_impaired
visual_impaired
clean_effects
captions
descriptions
metadata
```

For example, to make the second audio stream the default stream:

```
ffmpeg -i in.mkv -disposition:a:1 default out.mkv
```

To make the second subtitle stream the default stream and remove the default disposition from the first subtitle stream:

```
ffmpeg -i INPUT -disposition:s:0 0 -disposition:s:1 default OUTPUT
```

`-program`
`[title=`*`title`*`:][program_num=`*`program_num`*`:]st=`*`stream`*`[:st=`*`stream`*`...]`
(*`output`*)

Creates a program with the specified *title*, *program_num* and adds the specified *stream*(s) to it.

`-target` *type* (*output*)

Specify target file type (`vcd`, `svcd`, `dvd`, `dv`, `dv50`). *type* may be prefixed with `pal-`, `ntsc-` or `film-` to use the corresponding standard. All the format options (bitrate, codecs, buffer sizes) are then set automatically. You can just type:

```
ffmpeg -i myfile.avi -target vcd /tmp/vcd.mpg
```

Nevertheless you can specify additional options as long as you know they do not conflict with the standard, as in:

```
ffmpeg -i myfile.avi -target vcd -bf 2 /tmp/vcd.mpg
```

`-dframes` *number* (*output*)

Set the number of data frames to output. This is an obsolete alias for `-frames:d`, which you should use instead.

`-frames[:`*stream_specifier*`]` *framecount* (*output,per-stream*)

Stop writing to the stream after *framecount* frames.

`-q[:`*stream_specifier*`]` *q* (*output,per-stream*)
`-qscale[:`*stream_specifier*`]` *q* (*output,per-stream*)

Use fixed quality scale (VBR). The meaning of *q*/*qscale* is codec-dependent. If *qscale* is used without a *stream_specifier* then it applies only to the video stream, this is to maintain compatibility with previous behavior and as specifying the same codec specific value to 2 different codecs that is audio and video generally is not what is intended when no stream_specifier is used.

`-filter[:`*stream_specifier*`]` *filtergraph* (*output,per-stream*)

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

*filtergraph* is a description of the filtergraph to apply to the stream, and must have a single input and a single output of the same type of the stream. In the filtergraph, the input is associated to the label `in`, and the output to the label `out`. See the ffmpeg-filters manual for more information about the filtergraph syntax.

See the -filter_complex option if you want to create filtergraphs with multiple inputs and/or outputs.

`-filter_script[:`*stream_specifier*`]` *filename* (*output,per-stream*)

This option is similar to `-filter`, the only difference is that its argument is the name of the file from which a filtergraph description is to be read.

`-filter_threads` *nb_threads* (*global*)

Defines how many threads are used to process a filter pipeline. Each pipeline will produce a thread pool with this many threads available for parallel processing. The default is the number of available CPUs.

`-pre[:`*`stream_specifier`*`] `*`preset_name`* (*`output,per-stream`*)

Specify the preset for matching stream(s).

`-stats` (*`global`*)

Print encoding progress/statistics. It is on by default, to explicitly disable it you need to specify `-nostats`.

`-progress` *`url`* (*`global`*)

Send program-friendly progress information to *url*.

Progress information is written approximately every second and at the end of the encoding process. It is made of "*key*=*value*" lines. *key* consists of only alphanumeric characters. The last key of a sequence of progress information is always "progress".

`-stdin`

Enable interaction on standard input. On by default unless standard input is used as an input. To explicitly disable interaction you need to specify `-nostdin`.

Disabling interaction on standard input is useful, for example, if ffmpeg is in the background process group. Roughly the same result can be achieved with `ffmpeg ... < /dev/null` but it requires a shell.

`-debug_ts` (*`global`*)

Print timestamp information. It is off by default. This option is mostly useful for testing and debugging purposes, and the output format may change from one version to another, so it should not be employed by portable scripts.

See also the option `-fdebug ts`.

`-attach` *`filename`* (*`output`*)

Add an attachment to the output file. This is supported by a few formats like Matroska for e.g. fonts used in rendering subtitles. Attachments are implemented as a specific type of stream, so this option will add a new stream to the file. It is then possible to use per-stream options on this stream in the usual way. Attachment streams created with this option will be created after all the other streams (i.e. those created with `-map` or automatic mappings).

Note that for Matroska you also have to set the mimetype metadata tag:

```
ffmpeg -i INPUT -attach DejaVuSans.ttf -metadata:s:2 mimetype=application/x-truetype-font out.mkv
```

(assuming that the attachment stream will be third in the output file).

`-dump_attachment[:`*`stream_specifier`*`] `*`filename`* ` (`*`input,per-stream`*`)`

Extract the matching attachment stream into a file named *filename*. If *filename* is empty, then the value of the `filename` metadata tag will be used.

E.g. to extract the first attachment to a file named 'out.ttf':

```
ffmpeg -dump_attachment:t:0 out.ttf -i INPUT
```

To extract all attachments to files determined by the `filename` tag:

```
ffmpeg -dump_attachment:t "" -i INPUT
```

Technical note – attachments are implemented as codec extradata, so this option can actually be used to extract extradata from any stream, not just attachments.

`-noautorotate`

Disable automatically rotating video based on file metadata.

## 5.5 Video Options# TOC

`-vframes `*`number`* ` (`*`output`*`)`

Set the number of video frames to output. This is an obsolete alias for `-frames:v`, which you should use instead.

`-r[:`*`stream_specifier`*`] `*`fps`* ` (`*`input/output,per-stream`*`)`

Set frame rate (Hz value, fraction or abbreviation).

As an input option, ignore any timestamps stored in the file and instead generate timestamps assuming constant frame rate *fps*. This is not the same as the `-framerate` option used for some input formats like image2 or v4l2 (it used to be the same in older versions of FFmpeg). If in doubt use `-framerate` instead of the input option `-r`.

As an output option, duplicate or drop input frames to achieve constant output frame rate *fps*.

`-s[:`*`stream_specifier`*`] `*`size`* ` (`*`input/output,per-stream`*`)`

Set frame size.

As an input option, this is a shortcut for the `video_size` private option, recognized by some demuxers for which the frame size is either not stored in the file or is configurable – e.g. raw video or video grabbers.

As an output option, this inserts the `scale` video filter to the *end* of the corresponding filtergraph. Please use the `scale` filter directly to insert it at the beginning or some other place.

The format is '`wxh`' (default - same as source).

`-aspect[:`*stream_specifier*`] `*aspect*` (`*output*`,`*per-stream*`)`

Set the video display aspect ratio specified by *aspect*.

*aspect* can be a floating point number string, or a string of the form *num*:*den*, where *num* and *den* are the numerator and denominator of the aspect ratio. For example "4:3", "16:9", "1.3333", and "1.7777" are valid argument values.

If used together with `-vcodec copy`, it will affect the aspect ratio stored at container level, but not the aspect ratio stored in encoded frames, if it exists.

`-vn (`*output*`)`

Disable video recording.

`-vcodec `*codec*` (`*output*`)`

Set the video codec. This is an alias for `-codec:v`.

`-pass[:`*stream_specifier*`] `*n*` (`*output*`,`*per-stream*`)`

Select the pass number (1 or 2). It is used to do two-pass video encoding. The statistics of the video are recorded in the first pass into a log file (see also the option -passlogfile), and in the second pass that log file is used to generate the video at the exact requested bitrate. On pass 1, you may just deactivate audio and set output to null, examples for Windows and Unix:

```
ffmpeg -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y NUL
ffmpeg -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y /dev/null
```

`-passlogfile[:`*stream_specifier*`] `*prefix*` (`*output*`,`*per-stream*`)`

Set two-pass log file name prefix to *prefix*, the default file name prefix is "ffmpeg2pass". The complete file name will be `PREFIX-N.log`, where N is a number specific to the output stream

`-vf `*filtergraph*` (`*output*`)`

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

This is an alias for `-filter:v`, see the -filter option.

## 5.6 Advanced Video options# TOC

`-pix_fmt[:`*stream_specifier*`] `*format*` (`*input/output*`,`*per-stream*`)`

Set pixel format. Use `-pix_fmts` to show all the supported pixel formats. If the selected pixel format can not be selected, ffmpeg will print a warning and select the best pixel format supported by the encoder. If *pix_fmt* is prefixed by a +, ffmpeg will exit with an error if the requested pixel format can not be selected, and automatic conversions inside filtergraphs are disabled. If *pix_fmt* is a single +, ffmpeg selects the same pixel format as the input (or graph output) and automatic conversions are disabled.

`-sws_flags `*flags*` (`*input/output*`)`

Set SwScaler flags.

`-vdt `*n*

Discard threshold.

`-rc_override[:`*stream_specifier*`] `*override*` (`*output*`,`*per-stream*`)`

Rate control override for specific intervals, formatted as "int,int,int" list separated with slashes. Two first values are the beginning and end frame numbers, last one is quantizer to use if positive, or quality factor if negative.

`-ilme`

Force interlacing support in encoder (MPEG-2 and MPEG-4 only). Use this option if your input file is interlaced and you want to keep the interlaced format for minimum losses. The alternative is to deinterlace the input stream with `-deinterlace`, but deinterlacing introduces losses.

`-psnr`

Calculate PSNR of compressed frames.

`-vstats`

Dump video coding statistics to `vstats_HHMMSS.log`.

`-vstats_file `*file*

Dump video coding statistics to *file*.

`-vstats_version `*file*

Specifies which version of the vstats format to use. Default is 2.

version = 1 :

```
frame= %5d q= %2.1f PSNR= %6.2f f_size= %6d s_size= %8.0fkB time=
%0.3f br= %7.1fkbits/s avg_br= %7.1fkbits/s
```

version > 1:

```
out= %2d st= %2d frame= %5d q= %2.1f PSNR= %6.2f f_size= %6d s_size=
%8.0fkB time= %0.3f br= %7.1fkbits/s avg_br= %7.1fkbits/s
```

`-top[:`*stream_specifier*`] `*n*` (`*output,per-stream*`)`

top=1/bottom=0/auto=-1 field first

`-dc `*precision*

Intra_dc_precision.

`-vtag `*fourcc/tag*` (`*output*`)`

Force video tag/fourcc. This is an alias for `-tag:v`.

`-qphist (`*global*`)`

Show QP histogram

`-vbsf `*bitstream_filter*

Deprecated see -bsf

`-force_key_frames[:`*stream_specifier*`] `*time*`[,`*time...*`] (`*output,per-stream*`)`
`-force_key_frames[:`*stream_specifier*`] expr:`*expr*` (`*output,per-stream*`)`

Force key frames at the specified timestamps, more precisely at the first frames after each specified time.

If the argument is prefixed with `expr:`, the string *expr* is interpreted like an expression and is evaluated for each frame. A key frame is forced in case the evaluation is non-zero.

If one of the times is "`chapters`[*delta*]", it is expanded into the time of the beginning of all chapters in the file, shifted by *delta*, expressed as a time in seconds. This option can be useful to ensure that a seek point is present at a chapter mark or any other designated place in the output file.

For example, to insert a key frame at 5 minutes, plus key frames 0.1 second before the beginning of every chapter:

```
-force_key_frames 0:05:00,chapters-0.1
```

The expression in *expr* can contain the following constants:

`n`

the number of current processed frame, starting from 0

`n_forced`

the number of forced frames

`prev_forced_n`

the number of the previous forced frame, it is `NAN` when no keyframe was forced yet

`prev_forced_t`

the time of the previous forced frame, it is `NAN` when no keyframe was forced yet

`t`

the time of the current processed frame

For example to force a key frame every 5 seconds, you can specify:

```
-force_key_frames expr:gte(t,n_forced*5)
```

To force a key frame 5 seconds after the time of the last forced one, starting from second 13:

```
-force_key_frames expr:if(isnan(prev_forced_t),gte(t,13),gte(t,prev_forced_t+5))
```

Note that forcing too many keyframes is very harmful for the lookahead algorithms of certain encoders: using fixed-GOP options or similar would be more efficient.

`-copyinkf[:`*`stream_specifier`*`]` (*output,per-stream*)

When doing stream copy, copy also non-key frames found at the beginning.

`-init_hw_device` *type*`[=`*name*`][:`*device*`[,`*key=value*`...]]`

Initialise a new hardware device of type *type* called *name*, using the given device parameters. If no name is specified it will receive a default name of the form "*type*%d".

The meaning of *device* and the following arguments depends on the device type:

`cuda`

*device* is the number of the CUDA device.

`dxva2`

*device* is the number of the Direct3D 9 display adapter.

`vaapi`

*device* is either an X11 display name or a DRM render node. If not specified, it will attempt to open the default X11 display (*$DISPLAY*) and then the first DRM render node (*/dev/dri/renderD128*).

`vdpau`

*device* is an X11 display name. If not specified, it will attempt to open the default X11 display (*$DISPLAY*).

`qsv`

*device* selects a value in '`MFX_IMPL_*`'. Allowed values are:

```
auto
sw
hw
auto_any
hw_any
hw2
hw3
hw4
```

If not specified, '`auto_any`' is used. (Note that it may be easier to achieve the desired result for QSV by creating the platform-appropriate subdevice ('`dxva2`' or '`vaapi`') and then deriving a QSV device from that.)

`-init_hw_device` *type*`[=`*name*`]@`*source*

Initialise a new hardware device of type *type* called *name*, deriving it from the existing device with the name *source*.

`-init_hw_device list`

List all hardware device types supported in this build of ffmpeg.

`-filter_hw_device` *name*

Pass the hardware device called *name* to all filters in any filter graph. This can be used to set the device to upload to with the `hwupload` filter, or the device to map to with the `hwmap` filter. Other filters may also make use of this parameter when they require a hardware device. Note that this is typically only required when the input is not already in hardware frames - when it is, filters will derive the device they require from the context of the frames they receive as input.

This is a global setting, so all filters will receive the same device.

`-hwaccel[:`*`stream_specifier`*`] `*`hwaccel`*` (`*`input,per-stream`*`)`

Use hardware acceleration to decode the matching stream(s). The allowed values of *hwaccel* are:

`none`

Do not use any hardware acceleration (the default).

`auto`

Automatically select the hardware acceleration method.

`vda`

Use Apple VDA hardware acceleration.

`vdpau`

Use VDPAU (Video Decode and Presentation API for Unix) hardware acceleration.

`dxva2`

Use DXVA2 (DirectX Video Acceleration) hardware acceleration.

`vaapi`

Use VAAPI (Video Acceleration API) hardware acceleration.

`qsv`

Use the Intel QuickSync Video acceleration for video transcoding.

Unlike most other values, this option does not enable accelerated decoding (that is used automatically whenever a qsv decoder is selected), but accelerated transcoding, without copying the frames into the system memory.

For it to work, both the decoder and the encoder must support QSV acceleration and no filters must be used.

This option has no effect if the selected hwaccel is not available or not supported by the chosen decoder.

Note that most acceleration methods are intended for playback and will not be faster than software decoding on modern CPUs. Additionally, `ffmpeg` will usually need to copy the decoded frames from the GPU memory into the system memory, resulting in further performance loss. This option is thus mainly useful for testing.

`-hwaccel_device[:stream_specifier] hwaccel_device (input,per-stream)`

Select a device to use for hardware acceleration.

This option only makes sense when the `-hwaccel` option is also specified. It can either refer to an existing device created with `-init_hw_device` by name, or it can create a new device as if '`-init_hw_device`' *type*:*hwaccel_device* were called immediately before.

`-hwaccels`

List all hardware acceleration methods supported in this build of ffmpeg.

## 5.7 Audio Options# TOC

`-aframes number (output)`

Set the number of audio frames to output. This is an obsolete alias for `-frames:a`, which you should use instead.

`-ar[:stream_specifier] freq (input/output,per-stream)`

Set the audio sampling frequency. For output streams it is set by default to the frequency of the corresponding input stream. For input streams this option only makes sense for audio grabbing devices and raw demuxers and is mapped to the corresponding demuxer options.

`-aq q (output)`

Set the audio quality (codec-specific, VBR). This is an alias for -q:a.

`-ac[:stream_specifier] channels (input/output,per-stream)`

Set the number of audio channels. For output streams it is set by default to the number of input audio channels. For input streams this option only makes sense for audio grabbing devices and raw demuxers and is mapped to the corresponding demuxer options.

`-an (output)`

Disable audio recording.

`-acodec codec (input/output)`

Set the audio codec. This is an alias for `-codec:a`.

`-sample_fmt[:stream_specifier] sample_fmt (output,per-stream)`

Set the audio sample format. Use `-sample_fmts` to get a list of supported sample formats.

`-af filtergraph (output)`

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

This is an alias for `-filter:a`, see the -filter option.

## 5.8 Advanced Audio options# TOC

`-atag` *fourcc/tag* (*output*)

Force audio tag/fourcc. This is an alias for `-tag:a`.

`-absf` *bitstream_filter*

Deprecated, see -bsf

`-guess_layout_max` *channels* (*input,per-stream*)

If some input channel layout is not known, try to guess only if it corresponds to at most the specified number of channels. For example, 2 tells to `ffmpeg` to recognize 1 channel as mono and 2 channels as stereo but not 6 channels as 5.1. The default is to always try to guess. Use 0 to disable all guessing.

## 5.9 Subtitle options# TOC

`-scodec` *codec* (*input/output*)

Set the subtitle codec. This is an alias for `-codec:s`.

`-sn` (*output*)

Disable subtitle recording.

`-sbsf` *bitstream_filter*

Deprecated, see -bsf

## 5.10 Advanced Subtitle options# TOC

`-fix_sub_duration`

Fix subtitles durations. For each subtitle, wait for the next packet in the same stream and adjust the duration of the first to avoid overlap. This is necessary with some subtitles codecs, especially DVB subtitles, because the duration in the original packet is only a rough estimate and the end is actually marked by an empty subtitle frame. Failing to use this option when necessary can result in exaggerated durations or muxing failures due to non-monotonic timestamps.

Note that this option will delay the output of all data until the next subtitle packet is decoded: it may increase memory consumption and latency a lot.

`-canvas_size` *size*

Set the size of the canvas used to render subtitles.

## 5.11 Advanced options# TOC

`-map` [`-`]*input_file_id*[`:`*stream_specifier*][`?`][`,`*sync_file_id*[`:`*stream_specifier*]] | *[linklabel]* (*output*)

Designate one or more input streams as a source for the output file. Each input stream is identified by the input file index *input_file_id* and the input stream index *input_stream_id* within the input file. Both indices start at 0. If specified, *sync_file_id*:*stream_specifier* sets which input stream is used as a presentation sync reference.

The first `-map` option on the command line specifies the source for output stream 0, the second `-map` option specifies the source for output stream 1, etc.

A - character before the stream identifier creates a "negative" mapping. It disables matching streams from already created mappings.

A trailing `?` after the stream index will allow the map to be optional: if the map matches no streams the map will be ignored instead of failing. Note the map will still fail if an invalid input file index is used; such as if the map refers to a non-existent input.

An alternative *[linklabel]* form will map outputs from complex filter graphs (see the `-filter_complex` option) to the output file. *linklabel* must correspond to a defined output link label in the graph.

For example, to map ALL streams from the first input file to output

```
ffmpeg -i INPUT -map 0 output
```

For example, if you have two audio streams in the first input file, these streams are identified by "0:0" and "0:1". You can use `-map` to select which streams to place in an output file. For example:

```
ffmpeg -i INPUT -map 0:1 out.wav
```

will map the input stream in `INPUT` identified by "0:1" to the (single) output stream in `out.wav`.

For example, to select the stream with index 2 from input file `a.mov` (specified by the identifier "0:2"), and stream with index 6 from input `b.mov` (specified by the identifier "1:6"), and copy them to the output file `out.mov`:

```
ffmpeg -i a.mov -i b.mov -c copy -map 0:2 -map 1:6 out.mov
```

To select all video and the third audio stream from an input file:

```
ffmpeg -i INPUT -map 0:v -map 0:a:2 OUTPUT
```

To map all the streams except the second audio, use negative mappings

```
ffmpeg -i INPUT -map 0 -map -0:a:1 OUTPUT
```

To map the video and audio streams from the first input, and using the trailing `?`, ignore the audio mapping if no audio streams exist in the first input:

```
ffmpeg -i INPUT -map 0:v -map 0:a? OUTPUT
```

To pick the English audio stream:

```
ffmpeg -i INPUT -map 0:m:language:eng OUTPUT
```

Note that using this option disables the default mappings for this output file.

`-ignore_unknown`

Ignore input streams with unknown type instead of failing if copying such streams is attempted.

`-copy_unknown`

Allow input streams with unknown type to be copied instead of failing if copying such streams is attempted.

`-map_channel` [*input_file_id.stream_specifier.channel_id*|`-1`][`?`][`:`*output_file_id.stream_specifier*]

Map an audio channel from a given input to an output. If *output_file_id.stream_specifier* is not set, the audio channel will be mapped on all the audio streams.

Using "-1" instead of *input_file_id.stream_specifier.channel_id* will map a muted channel.

A trailing `?` will allow the map_channel to be optional: if the map_channel matches no channel the map_channel will be ignored instead of failing.

For example, assuming *INPUT* is a stereo audio file, you can switch the two audio channels with the following command:

```
ffmpeg -i INPUT -map_channel 0.0.1 -map_channel 0.0.0 OUTPUT
```

If you want to mute the first channel and keep the second:

```
ffmpeg -i INPUT -map_channel -1 -map_channel 0.0.1 OUTPUT
```

The order of the "-map_channel" option specifies the order of the channels in the output stream. The output channel layout is guessed from the number of channels mapped (mono if one "-map_channel", stereo if two, etc.). Using "-ac" in combination of "-map_channel" makes the channel gain levels to be updated if input and output channel layouts don’t match (for instance two "-map_channel" options and "-ac 6").

You can also extract each channel of an input to specific outputs; the following command extracts two channels of the *INPUT* audio stream (file 0, stream 0) to the respective *OUTPUT_CH0* and *OUTPUT_CH1* outputs:

```
ffmpeg -i INPUT -map_channel 0.0.0 OUTPUT_CH0 -map_channel 0.0.1 OUTPUT_CH1
```

The following example splits the channels of a stereo input into two separate streams, which are put into the same output file:

```
ffmpeg -i stereo.wav -map 0:0 -map 0:0 -map_channel 0.0.0:0.0 -map_channel 0.0.1:0.1 -y out.ogg
```

Note that currently each output stream can only contain channels from a single input stream; you can't for example use "-map_channel" to pick multiple input audio channels contained in different streams (from the same or different files) and merge them into a single output stream. It is therefore not currently possible, for example, to turn two separate mono streams into a single stereo stream. However splitting a stereo stream into two single channel mono streams is possible.

If you need this feature, a possible workaround is to use the *amerge* filter. For example, if you need to merge a media (here `input.mkv`) with 2 mono audio streams into one single stereo channel audio stream (and keep the video stream), you can use the following command:

```
ffmpeg -i input.mkv -filter_complex "[0:1] [0:2] amerge" -c:a pcm_s16le -c:v copy output.mkv
```

To map the first two audio channels from the first input, and using the trailing `?`, ignore the audio channel mapping if the first input is mono instead of stereo:

```
ffmpeg -i INPUT -map_channel 0.0.0 -map_channel 0.0.1? OUTPUT
```

`-map_metadata[:`*`metadata_spec_out`*`] `*`infile`*`[:`*`metadata_spec_in`*`] (`*`output,per-metadata`*`)`

Set metadata information of the next output file from *infile*. Note that those are file indices (zero-based), not filenames. Optional *metadata_spec_in/out* parameters specify, which metadata to copy. A metadata specifier can have the following forms:

*`g`*

global metadata, i.e. metadata that applies to the whole file

*`s`*`[:`*`stream_spec`*`]`

per-stream metadata. *stream_spec* is a stream specifier as described in the Stream specifiers chapter. In an input metadata specifier, the first matching stream is copied from. In an output metadata specifier, all matching streams are copied to.

*`c`*`:`*`chapter_index`*

per-chapter metadata. *chapter_index* is the zero-based chapter index.

`p:program_index`

per-program metadata. *program_index* is the zero-based program index.

If metadata specifier is omitted, it defaults to global.

By default, global metadata is copied from the first input file, per-stream and per-chapter metadata is copied along with streams/chapters. These default mappings are disabled by creating any mapping of the relevant type. A negative file index can be used to create a dummy mapping that just disables automatic copying.

For example to copy metadata from the first stream of the input file to global metadata of the output file:

```
ffmpeg -i in.ogg -map_metadata 0:s:0 out.mp3
```

To do the reverse, i.e. copy global metadata to all audio streams:

```
ffmpeg -i in.mkv -map_metadata:s:a 0:g out.mkv
```

Note that simple `0` would work as well in this example, since global metadata is assumed by default.

`-map_chapters` *input_file_index* (*output*)

Copy chapters from input file with index *input_file_index* to the next output file. If no chapter mapping is specified, then chapters are copied from the first input file with at least one chapter. Use a negative file index to disable any chapter copying.

`-benchmark` (*global*)

Show benchmarking information at the end of an encode. Shows CPU time used and maximum memory consumption. Maximum memory consumption is not supported on all systems, it will usually display as 0 if not supported.

`-benchmark_all` (*global*)

Show benchmarking information during the encode. Shows CPU time used in various steps (audio/video encode/decode).

`-timelimit` *duration* (*global*)

Exit after ffmpeg has been running for *duration* seconds.

`-dump` (*global*)

Dump each input packet to stderr.

`-hex` (*global*)

When dumping packets, also dump the payload.

`-re (`*`input`*`)`

Read input at native frame rate. Mainly used to simulate a grab device, or live input stream (e.g. when reading from a file). Should not be used with actual grab devices or live input streams (where it can cause packet loss). By default `ffmpeg` attempts to read the input(s) as fast as possible. This option will slow down the reading of the input(s) to the native frame rate of the input(s). It is useful for real-time output (e.g. live streaming).

`-loop_input`

Loop over the input stream. Currently it works only for image streams. This option is used for automatic FFserver testing. This option is deprecated, use -loop 1.

`-loop_output` *`number_of_times`*

Repeatedly loop output for formats that support looping such as animated GIF (0 will loop the output infinitely). This option is deprecated, use -loop.

`-vsync` *`parameter`*

Video sync method. For compatibility reasons old values can be specified as numbers. Newly added values will have to be specified as strings always.

`0, passthrough`

Each frame is passed with its timestamp from the demuxer to the muxer.

`1, cfr`

Frames will be duplicated and dropped to achieve exactly the requested constant frame rate.

`2, vfr`

Frames are passed through with their timestamp or dropped so as to prevent 2 frames from having the same timestamp.

`drop`

As passthrough but destroys all timestamps, making the muxer generate fresh timestamps based on frame-rate.

`-1, auto`

Chooses between 1 and 2 depending on muxer capabilities. This is the default method.

Note that the timestamps may be further modified by the muxer, after this. For example, in the case that the format option `avoid_negative_ts` is enabled.

With -map you can select from which stream the timestamps should be taken. You can leave either video or audio unchanged and sync the remaining stream(s) to the unchanged one.

`-frame_drop_threshold` *parameter*

Frame drop threshold, which specifies how much behind video frames can be before they are dropped. In frame rate units, so 1.0 is one frame. The default is -1.1. One possible usecase is to avoid framedrops in case of noisy timestamps or to increase frame drop precision in case of exact timestamps.

`-async` *samples_per_second*

Audio sync method. "Stretches/squeezes" the audio stream to match the timestamps, the parameter is the maximum samples per second by which the audio is changed. -async 1 is a special case where only the start of the audio stream is corrected without any later correction.

Note that the timestamps may be further modified by the muxer, after this. For example, in the case that the format option `avoid_negative_ts` is enabled.

This option has been deprecated. Use the `aresample` audio filter instead.

`-copyts`

Do not process input timestamps, but keep their values without trying to sanitize them. In particular, do not remove the initial start time offset value.

Note that, depending on the `vsync` option or on specific muxer processing (e.g. in case the format option `avoid_negative_ts` is enabled) the output timestamps may mismatch with the input timestamps even when this option is selected.

`-start_at_zero`

When used with `copyts`, shift input timestamps so they start at zero.

This means that using e.g. `-ss 50` will make output timestamps start at 50 seconds, regardless of what timestamp the input file started at.

`-copytb` *mode*

Specify how to set the encoder timebase when stream copying. *mode* is an integer numeric value, and can assume one of the following values:

`1`

Use the demuxer timebase.

The time base is copied to the output encoder from the corresponding input demuxer. This is sometimes required to avoid non monotonically increasing timestamps when copying video streams with variable frame rate.

`0`

Use the decoder timebase.

The time base is copied to the output encoder from the corresponding input decoder.

`-1`

Try to make the choice automatically, in order to generate a sane output.

Default value is -1.

`-enc_time_base[:`*`stream_specifier`*`] `*`timebase`*` (`*`output,per-stream`*`)`

Set the encoder timebase. *timebase* is a floating point number, and can assume one of the following values:

`0`

Assign a default value according to the media type.

For video - use 1/framerate, for audio - use 1/samplerate.

`-1`

Use the input stream timebase when possible.

If an input stream is not available, the default timebase will be used.

`>0`

Use the provided number as the timebase.

This field can be provided as a ratio of two integers (e.g. 1:24, 1:48000) or as a floating point number (e.g. 0.04166, 2.0833e-5)

Default value is 0.

`-shortest (`*`output`*`)`

Finish encoding when the shortest input stream ends.

`-dts_delta_threshold`

Timestamp discontinuity delta threshold.

`-muxdelay` *seconds* (*input*)

Set the maximum demux-decode delay.

`-muxpreload` *seconds* (*input*)

Set the initial demux-decode delay.

`-streamid` *output-stream-index*`:`*new-value* (*output*)

Assign a new stream-id value to an output stream. This option should be specified prior to the output filename to which it applies. For the situation where multiple output files exist, a streamid may be reassigned to a different value.

For example, to set the stream 0 PID to 33 and the stream 1 PID to 36 for an output mpegts file:

```
ffmpeg -i inurl -streamid 0:33 -streamid 1:36 out.ts
```

`-bsf[:`*stream_specifier*`]` *bitstream_filters* (*output,per-stream*)

Set bitstream filters for matching streams. *bitstream_filters* is a comma-separated list of bitstream filters. Use the `-bsfs` option to get the list of bitstream filters.

```
ffmpeg -i h264.mp4 -c:v copy -bsf:v h264_mp4toannexb -an out.h264
```

```
ffmpeg -i file.mov -an -vn -bsf:s mov2textsub -c:s copy -f rawvideo sub.txt
```

`-tag[:`*stream_specifier*`]` *codec_tag* (*input/output,per-stream*)

Force a tag/fourcc for matching streams.

`-timecode` *hh*`:`*mm*`:`*ss*`SEP`*ff*

Specify Timecode for writing. *SEP* is ':' for non drop timecode and ';' (or '.') for drop.

```
ffmpeg -i input.mpg -timecode 01:02:03.04 -r 30000/1001 -s ntsc output.mpg
```

`-filter_complex` *filtergraph* (*global*)

Define a complex filtergraph, i.e. one with arbitrary number of inputs and/or outputs. For simple graphs – those with one input and one output of the same type – see the `-filter` options. *filtergraph* is a description of the filtergraph, as described in the "Filtergraph syntax" section of the ffmpeg-filters manual.

Input link labels must refer to input streams using the `[file_index:stream_specifier]` syntax (i.e. the same as `-map` uses). If *stream_specifier* matches multiple streams, the first one will be used. An unlabeled input will be connected to the first unused input stream of the matching type.

Output link labels are referred to with `-map`. Unlabeled outputs are added to the first output file.

Note that with this option it is possible to use only lavfi sources without normal input files.

For example, to overlay an image over video

```
ffmpeg -i video.mkv -i image.png -filter_complex '[0:v][1:v]overlay[out]' -map
'[out]' out.mkv
```

Here `[0:v]` refers to the first video stream in the first input file, which is linked to the first (main) input of the overlay filter. Similarly the first video stream in the second input is linked to the second (overlay) input of overlay.

Assuming there is only one video stream in each input file, we can omit input labels, so the above is equivalent to

```
ffmpeg -i video.mkv -i image.png -filter_complex 'overlay[out]' -map
'[out]' out.mkv
```

Furthermore we can omit the output label and the single output from the filter graph will be added to the output file automatically, so we can simply write

```
ffmpeg -i video.mkv -i image.png -filter_complex 'overlay' out.mkv
```

To generate 5 seconds of pure red video using lavfi `color` source:

```
ffmpeg -filter_complex 'color=c=red' -t 5 out.mkv
```

`-filter_complex_threads` *nb_threads* (*global*)

Defines how many threads are used to process a filter_complex graph. Similar to filter_threads but used for `-filter_complex` graphs only. The default is the number of available CPUs.

`-lavfi` *filtergraph* (*global*)

Define a complex filtergraph, i.e. one with arbitrary number of inputs and/or outputs. Equivalent to `-filter_complex`.

`-filter_complex_script` *filename* (*global*)

This option is similar to `-filter_complex`, the only difference is that its argument is the name of the file from which a complex filtergraph description is to be read.

`-accurate_seek` (*input*)

This option enables or disables accurate seeking in input files with the `-ss` option. It is enabled by default, so seeking is accurate when transcoding. Use `-noaccurate_seek` to disable it, which may be useful e.g. when copying some streams and transcoding the others.

`-seek_timestamp (`*`input`*`)`

This option enables or disables seeking by timestamp in input files with the `-ss` option. It is disabled by default. If enabled, the argument to the `-ss` option is considered an actual timestamp, and is not offset by the start time of the file. This matters only for files which do not start from timestamp 0, such as transport streams.

`-thread_queue_size` *`size`* `(`*`input`*`)`

This option sets the maximum number of queued packets when reading from the file or device. With low latency / high rate live streams, packets may be discarded if they are not read in a timely manner; raising this value can avoid it.

`-override_ffserver (`*`global`*`)`

Overrides the input specifications from `ffserver`. Using this option you can map any input stream to `ffserver` and control many aspects of the encoding from `ffmpeg`. Without this option `ffmpeg` will transmit to `ffserver` what is requested by `ffserver`.

The option is intended for cases where features are needed that cannot be specified to `ffserver` but can be to `ffmpeg`.

`-sdp_file` *`file`* `(`*`global`*`)`

Print sdp information for an output stream to *file*. This allows dumping sdp information when at least one output isn't an rtp stream. (Requires at least one of the output formats to be rtp).

`-discard (`*`input`*`)`

Allows discarding specific streams or frames of streams at the demuxer. Not all demuxers support this.

`none`

Discard no frame.

`default`

Default, which discards no frames.

`noref`

Discard all non-reference frames.

`bidir`

Discard all bidirectional frames.

`nokey`

Discard all frames excepts keyframes.

`all`

Discard all frames.

`-abort_on` *`flags`* (*`global`*)

Stop and abort on various conditions. The following flags are available:

`empty_output`

No packets were passed to the muxer, the output is empty.

`-xerror` (*`global`*)

Stop and exit on error

`-max_muxing_queue_size` *`packets`* (*`output,per-stream`*)

When transcoding audio and/or video streams, ffmpeg will not begin writing into the output until it has one packet for each such stream. While waiting for that to happen, packets for other streams are buffered. This option sets the size of this buffer, in packets, for the matching output stream.

The default value of this option should be high enough for most uses, so only touch this option if you are sure that you need it.

As a special exception, you can use a bitmap subtitle stream as input: it will be converted into a video with the same size as the largest video in the file, or 720x576 if no video is present. Note that this is an experimental and temporary solution. It will be removed once libavfilter has proper support for subtitles.

For example, to hardcode subtitles on top of a DVB-T recording stored in MPEG-TS format, delaying the subtitles by 1 second:

```
ffmpeg -i input.ts -filter_complex \
  '[#0x2ef] setpts=PTS+1/TB [sub] ; [#0x2d0] [sub] overlay' \
  -sn -map '#0x2dc' output.mkv
```

(0x2d0, 0x2dc and 0x2ef are the MPEG-TS PIDs of respectively the video, audio and subtitles streams; 0:0, 0:3 and 0:7 would have worked too)

## 5.12 Preset files# TOC

A preset file contains a sequence of *option=value* pairs, one for each line, specifying a sequence of options which would be awkward to specify on the command line. Lines starting with the hash ('#') character are ignored and are used to provide comments. Check the `presets` directory in the FFmpeg source tree for examples.

There are two types of preset files: ffpreset and avpreset files.

### 5.12.1 ffpreset files# TOC

ffpreset files are specified with the `vpre`, `apre`, `spre`, and `fpre` options. The `fpre` option takes the filename of the preset instead of a preset name as input and can be used for any kind of codec. For the `vpre`, `apre`, and `spre` options, the options specified in a preset file are applied to the currently selected codec of the same type as the preset option.

The argument passed to the `vpre`, `apre`, and `spre` preset options identifies the preset file to use according to the following rules:

First ffmpeg searches for a file named *arg*.ffpreset in the directories `$FFMPEG_DATADIR` (if set), and `$HOME/.ffmpeg`, and in the datadir defined at configuration time (usually `PREFIX/share/ffmpeg`) or in a `ffpresets` folder along the executable on win32, in that order. For example, if the argument is `libvpx-1080p`, it will search for the file `libvpx-1080p.ffpreset`.

If no such file is found, then ffmpeg will search for a file named *codec_name-arg*.ffpreset in the above-mentioned directories, where *codec_name* is the name of the codec to which the preset file options will be applied. For example, if you select the video codec with `-vcodec libvpx` and use `-vpre 1080p`, then it will search for the file `libvpx-1080p.ffpreset`.

### 5.12.2 avpreset files# TOC

avpreset files are specified with the `pre` option. They work similar to ffpreset files, but they only allow encoder- specific options. Therefore, an *option*=*value* pair specifying an encoder cannot be used.

When the `pre` option is specified, ffmpeg will look for files with the suffix .avpreset in the directories `$AVCONV_DATADIR` (if set), and `$HOME/.avconv`, and in the datadir defined at configuration time (usually `PREFIX/share/ffmpeg`), in that order.

First ffmpeg searches for a file named *codec_name-arg*.avpreset in the above-mentioned directories, where *codec_name* is the name of the codec to which the preset file options will be applied. For example, if you select the video codec with `-vcodec libvpx` and use `-pre 1080p`, then it will search for the file `libvpx-1080p.avpreset`.

If no such file is found, then ffmpeg will search for a file named *arg*.avpreset in the same directories.

# 6 Examples# TOC

## 6.1 Video and Audio grabbing# TOC

If you specify the input format and device then ffmpeg can grab video and audio directly.

```
ffmpeg -f oss -i /dev/dsp -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

Or with an ALSA audio source (mono input, card id 1) instead of OSS:

```
ffmpeg -f alsa -ac 1 -i hw:1 -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

Note that you must activate the right video source and channel before launching ffmpeg with any TV viewer such as xawtv by Gerd Knorr. You also have to set the audio recording levels correctly with a standard mixer.

## 6.2 X11 grabbing# TOC

Grab the X11 display with ffmpeg via

```
ffmpeg -f x11grab -video_size cif -framerate 25 -i :0.0 /tmp/out.mpg
```

0.0 is display.screen number of your X11 server, same as the DISPLAY environment variable.

```
ffmpeg -f x11grab -video_size cif -framerate 25 -i :0.0+10,20 /tmp/out.mpg
```

0.0 is display.screen number of your X11 server, same as the DISPLAY environment variable. 10 is the x-offset and 20 the y-offset for the grabbing.

## 6.3 Video and Audio file format conversion# TOC

Any supported file format and protocol can serve as input to ffmpeg:

Examples:

- You can use YUV files as input:

  ```
  ffmpeg -i /tmp/test%d.Y /tmp/out.mpg
  ```

  It will use the files:

  ```
  /tmp/test0.Y, /tmp/test0.U, /tmp/test0.V,
  /tmp/test1.Y, /tmp/test1.U, /tmp/test1.V, etc...
  ```

  The Y files use twice the resolution of the U and V files. They are raw files, without header. They can be generated by all decent video decoders. You must specify the size of the image with the `-s` option if ffmpeg cannot guess it.

- You can input from a raw YUV420P file:

  ```
  ffmpeg -i /tmp/test.yuv /tmp/out.avi
  ```

  test.yuv is a file containing raw YUV planar data. Each frame is composed of the Y plane followed by the U and V planes at half vertical and horizontal resolution.

- You can output to a raw YUV420P file:

```
ffmpeg -i mydivx.avi hugefile.yuv
```

- You can set several input files and output files:

  ```
  ffmpeg -i /tmp/a.wav -s 640x480 -i /tmp/a.yuv /tmp/a.mpg
  ```

  Converts the audio file a.wav and the raw YUV video file a.yuv to MPEG file a.mpg.

- You can also do audio and video conversions at the same time:

  ```
  ffmpeg -i /tmp/a.wav -ar 22050 /tmp/a.mp2
  ```

  Converts a.wav to MPEG audio at 22050 Hz sample rate.

- You can encode to several formats at the same time and define a mapping from input stream to output streams:

  ```
  ffmpeg -i /tmp/a.wav -map 0:a -b:a 64k /tmp/a.mp2 -map 0:a -b:a 128k /tmp/b.mp2
  ```

  Converts a.wav to a.mp2 at 64 kbits and to b.mp2 at 128 kbits. '-map file:index' specifies which input stream is used for each output stream, in the order of the definition of output streams.

- You can transcode decrypted VOBs:

  ```
  ffmpeg -i snatch_1.vob -f avi -c:v mpeg4 -b:v 800k -g 300 -bf 2 -c:a libmp3lame -b:a 128k snatch.avi
  ```

  This is a typical DVD ripping example; the input is a VOB file, the output an AVI file with MPEG-4 video and MP3 audio. Note that in this command we use B-frames so the MPEG-4 stream is DivX5 compatible, and GOP size is 300 which means one intra frame every 10 seconds for 29.97fps input video. Furthermore, the audio stream is MP3-encoded so you need to enable LAME support by passing `--enable-libmp3lame` to configure. The mapping is particularly useful for DVD transcoding to get the desired audio language.

  NOTE: To see the supported input formats, use `ffmpeg -demuxers`.

- You can extract images from a video, or create a video from many images:

  For extracting images from a video:

  ```
  ffmpeg -i foo.avi -r 1 -s WxH -f image2 foo-%03d.jpeg
  ```

  This will extract one video frame per second from the video and will output them in files named `foo-001.jpeg`, `foo-002.jpeg`, etc. Images will be rescaled to fit the new WxH values.

  If you want to extract just a limited number of frames, you can use the above command in combination with the `-frames:v` or `-t` option, or in combination with -ss to start extracting from a certain point in time.

  For creating a video from many images:

```
ffmpeg -f image2 -framerate 12 -i foo-%03d.jpeg -s WxH foo.avi
```

The syntax `foo-%03d.jpeg` specifies to use a decimal number composed of three digits padded with zeroes to express the sequence number. It is the same syntax supported by the C printf function, but only formats accepting a normal integer are suitable.

When importing an image sequence, -i also supports expanding shell-like wildcard patterns (globbing) internally, by selecting the image2-specific `-pattern_type glob` option.

For example, for creating a video from filenames matching the glob pattern `foo-*.jpeg`:

```
ffmpeg -f image2 -pattern_type glob -framerate 12 -i 'foo-*.jpeg' -s WxH foo.avi
```

- You can put many streams of the same type in the output:

```
ffmpeg -i test1.avi -i test2.avi -map 1:1 -map 1:0 -map 0:1 -map 0:0 -c copy -y test12.nut
```

The resulting output file `test12.nut` will contain the first four streams from the input files in reverse order.

- To force CBR video output:

```
ffmpeg -i myfile.avi -b 4000k -minrate 4000k -maxrate 4000k -bufsize 1835k out.m2v
```

- The four options lmin, lmax, mblmin and mblmax use 'lambda' units, but you may use the QP2LAMBDA constant to easily convert from 'q' units:

```
ffmpeg -i src.ext -lmax 21*QP2LAMBDA dst.ext
```

# 7 See Also# TOC

ffmpeg-all, ffplay, ffprobe, ffserver, ffmpeg-utils, ffmpeg-scaler, ffmpeg-resampler, ffmpeg-codecs, ffmpeg-bitstream-filters, ffmpeg-formats, ffmpeg-devices, ffmpeg-protocols, ffmpeg-filters

# 8 Authors# TOC

The FFmpeg developers.

For details about the authorship, see the Git history of the project (git://source.ffmpeg.org/ffmpeg), e.g. by typing the command `git log` in the FFmpeg source directory, or browsing the online repository at http://source.ffmpeg.org.

Maintainers for the specific components are listed in the file `MAINTAINERS` in the source code tree.

This document was generated using *makeinfo*.