Cut: in detail

Русский

The section cuts camera files **without re-encoding**. The streams are copied as they are, no generation is lost, the metadata and the timecode move into the result. This is not a rival to an editing program but an exact trimmer of what is extra, with a guarantee that the camera material reaches the edit as it was shot.

Sound files are cut by the same section and with the same three buttons. Sound has no keyframes: WAV, FLAC and ALAC are cut **on the sample**, compressed sound by default on the bounds of its packets, without re-encoding, and with Exact on the sample, at the price of one more encoding. This is described in the section «Sound files» below.

Everything written here is checked by a run on real files; the numbers in the text are measured, not expected. The measurements are collected in `MEASUREMENTS_RU.md` (Russian), the compatibility tables are in `TRIM_MATRIX_EN.md`, the player keys are in `MPV_HOTKEYS_EN.md`.

Working cycle

1. Put the takes or the sound files into the Source (`input`).
2. Choose the file with the badge: the arrows on the left and on the right step through the folder, and the button with the table shows what `ffprobe` knows about the file before the run.
3. `Open in the player`: the take opens in mpv in a separate window.
4. Find the points: `,` and `.` step exactly one frame, `Shift+←` and `Shift+→` exactly one second, `I` sets A, a second press sets B. The piece between the points plays in a loop, and the cut is heard before it is made.
5. `Take A and B`: both points move into the fields of the panel. Or `Mark IN` and `Mark OUT`, if it is handier to keep your hands on the panel: they take the position where the player stands and move the loop point there too.
6. Choose what to do with the piece between the points, and press `RUN`.

The points can also simply be typed on the keyboard: `00:01:28,400` or just seconds. This way needs nothing but digits, and it works even if the player is not installed.

Several pieces: the `+` button

To the right of the last row stands `+`. It is gray until the row has everything it needs: for `KEEP` one point is enough, for `CUT OUT` both are needed, for `SPLIT` one point. A press adds the next row and clears points A and B in the player, so that the next piece is searched from a clean slate. The player always writes into the last row. An empty row is simply not counted, so there is no need to delete it. The limit is 20 rows and 30 cut points.

To the right of OUT in each `KEEP` and `CUT OUT` row is the duration of the piece: how much will remain or how much will be cut out. A single IN in `KEEP` counts to the end of the file.

Spinners in the point fields

Each point field has `▲▼` on its right. The field stays one line of text, so the timecode is still copied and pasted whole. An arrow changes the part where the cursor is: hours, minutes, seconds or the last part, which steps by one frame of the file and lands on the frame grid (for a sound file, by one millisecond). The carry between seconds and minutes is automatic. Without the cursor in the field, the seconds step. Holding speeds it up: each repeat is one more than the previous one: 1, 2, 3… Checked: at 25p `00:02:01,013` became `00:02:01,040` after one frame step up, and `00:02:00,960` after two steps down; one second of holding on the seconds gave 1, 3, 6, 10, 15, 21.

The player and the selected file

The arrows of the badge change the file in the section, not in the player. So the player buttons first ask mpv which file it shows, and move the points only if it is the file selected in the section. Otherwise there is a warning with the name of the file in the player and a hint to press `Open in the player`.

`Open in the player` with another file clears points A and B: in mpv they are a setting of the player, not a property of the file, and they survive a change of file. Measured: after an ordinary change of file the 1–3 s loop stayed on the new file.

What to do with the piece between the points

Three buttons. No more are needed: which edge goes away is already said by which fields are filled.

`KEEP`

What is between the points stays; the rest goes.

Filled	What happens
IN only	the start goes: the clapperboard, the approach, the false start
OUT only	the end goes: the walk-off and the camera stop
both points	both edges go, the middle stays
nothing	the section refuses: «Nothing would be trimmed: no cut point is set.»

`CUT OUT`

The other way round: the piece between the points disappears, and the start and the end are joined into one file. A slip of the tongue, a phone call, an assistant walking through the frame.

It works in three FFmpeg passes, and none of them re-encodes: the head up to IN, the tail from the keyframe after OUT, the join by copying packets. The intermediate pieces live in `workspace` and are deleted however the run ends.

Rows are variants, not one editing timeline

In `KEEP` and `CUT OUT` each row added with the plus is a separate run on the same take and a separate file: `name_keep01`, `name_keep02` or `name_cut01`, `name_cut02`. The number is the number of the row on the screen. The rows know nothing about each other, so they can overlap in any way: one moment can go into two clips, and from the whole take each file has its own piece cut out. With one filled row there is no suffix.

Checked on a 30-second take (a keyframe every second): rows 5–12.5 and 8–20 in `KEEP` gave two files of 188 and 300 frames, in `CUT OUT` 550 and 450 frames: the tail starts with the keyframe after OUT.

`SPLIT`

Nothing is thrown away. N points cut the file into N + 1 parts: `name_part1`, `name_part2` and so on; from ten parts on, the number is written with a zero, `name_part01`, so that the parts go in order in Explorer. Each point has one field; `+` adds the next one.

Points are taken only forward. `Take A` and `Mark here` write the position of the player into the last field only if it is past the previous points; otherwise the panel says where the player stands and which

point it has not passed. A slider moved back by accident spoils nothing. The engine itself puts the points typed by hand in order. If two points land on one keyframe, the run stops and names both: it does not write an empty part.

Checked: four points at 4, 11, 17 and 23 s on the same take gave five parts, together exactly 750 frames of 750; two points on a 30-second WAV gave three parts, together 1,440,000 samples of 1,440,000.

Each point lands on the **nearest** keyframe, and this frame is the shared join of two neighboring parts: one ends before it, the next starts with it. So the parts placed one after another give the source frame for frame, and the sound sample for sample.

Two independent cuts, «remove the end» and «remove the start», meet only if the point itself stands on a keyframe. Measured on a 30-second take (750 frames, a keyframe every second): a point at 10.2 s puts **5 frames into both parts** this way, a point at 10.7 s leaves **7 frames in neither**. Split cuts each join once, and in all five cases checked (a point on a keyframe, after it, before it, two points, two points in reverse order) the frame hashes of the parts in a row matched the list of the source, and the raw samples of the parts in a row matched the 1,440,000 samples of the source byte for byte.

If both points land on one keyframe, the middle part would be empty: the section refuses and asks to move the points apart.

Why the start lands on a keyframe

A copied video stream cannot start on an arbitrary frame: the frames between keyframes are described through their neighbors, and without a reference frame there is nothing to decode them with. So the start of a piece lands on the nearest keyframe, and the journal says where it actually landed:

```
cut in  00:00:09.000 -> keyframe 00:00:08.400 (-0.6 s)
```

The end is counted differently: in frames, by the exact fraction of the rate, so it lands exactly on the requested frame:

```
cut out 00:00:21.000 -> frame 300 of the piece (00:00:12.000)
result  300 frames, 00:00:12.000 (+0.000 s from the plan), timecode 01:00:09:00
```

When the middle is cut out, this rule works the other way round. The tail starts with the **first keyframe after OUT**, not with the nearest one: if it started earlier, a part of what was asked to be

removed would stay in the result. How much more than requested went away is said in words:

```
tail at 00:00:09.000 (first keyframe past the piece, +0.7 s)
[WARNING] 0.7 s more than asked for was removed. Starting earlier would leave
          part of the removed piece in the result.
```

If a keyframe stands exactly at the point, nothing extra goes away.

The frame rate comes from the file and is not guessed

`ffprobe` gives the exact fraction: `24000/1001` for 23.976 and `24/1` for a true 24.000; a camera shoots both. The fraction goes into the calculation as a fraction, so on an hour-long file there is no accumulated error.

If the file is shot with a **variable frame rate** (a phone writes 30 in one field and something in between in another), the frame count would lie: the section switches to counting by time and says so. The end then lands inside a packet, not on a frame.

Sound

The sound is copied together with the picture. In two cases a copy is not exact enough, and the section rewrites PCM in the same format and the same bit depth: bit for bit the same sound, but cut on the sample:

- **MXF**: the chunks are large, and a copy would drag about 8 ms of sound from the other side of the cut;
- **the join** (`CUT OUT`): the excess does not just hang at the end, it falls exactly on the join. Measured on a 25 fps take: without this the join drifted by 72 ms, then by 16 ms; with it, exactly 1,200,000 samples in 25.000 s.

`Audio channel` can take the first, the second, or fold both into mono: a camera often writes the shotgun and the lavalier as two mono channels. Choosing a channel **rebuilds** the sound: for PCM this is free, for compressed sound it costs a generation, and the section warns about it.

What else travels with the material

A camera writes more than picture and sound, and what is dropped silently a person would find only in the edit. So there are two ticks here, both on by default.

`Telemetry` is a data stream. A GoPro writes its telemetry there. Such a stream is held by **MOV and MP4**; **MKV does not hold it at all** («Only audio, video, and subtitles are supported for Matroska»), MXF does not either, and the section says so before the run instead of failing in the middle.

`Subtitles` are rare in camera files, but a DJI writes its flight data as subtitles. MOV, MP4 and MKV take them.

Subtitles are not copied through the cut but are rebuilt for it. Packet copying was tested, and it went wrong at every boundary: with a cut by the number of frames FFmpeg dropped a line running across OUT; a line that started before IN kept its full length; and the `CUT OUT` join turned a 0.5–1.5 s line into one microsecond and moved the next line a second earlier.

So the section takes the text of each line out into SRT (on the file's own clock, `-copyts`), trims the lines to the pieces kept, shifts them onto the new time scale and mixes the finished track in as a second input of the same FFmpeg call that cuts the picture. The picture is still copied. The result:

- a line running across IN or OUT keeps its own part inside the piece;
- the piece starts with the keyframe before IN, and the lines are counted from it too, together with the picture;
- in `CUT OUT` the lines of the tail move close up to the head; a line that was cut only by the removed piece is whole again;
- in `SPLIT` a line across a join goes to both parts as its two halves.

Measured on an 8-second 25p take (lines 0.5–1.5, 2.5–3.5 and 5.0–7.5 s): `KEEP` 0–7 gave 0.5–1.5, 2.5–3.5 and 5.0–7.0; `CUT OUT` 2–4 gave 0.5–1.5 and 3.0–5.5; IN 1.2 (keyframe 1.0) gave 0.0–0.5, 1.5–2.5 and 4.0–6.0.

Each container writes text in its own form (measured on FFmpeg 8.0.1): MP4 and MOV write `mov_text`, MKV writes SRT; another format does not go there as a copy. So `mov_text` for MKV, and SRT, ASS, WebVTT for MP4 are rewritten: the words and the timings travel, the ASS styling does not, and the journal says so. Picture subtitles (PGS, DVD) stay behind: a picture of text cannot be trimmed to a piece without redrawing it. The language of the track travels.

The data stream (telemetry) does not pass through the `CUT OUT` join, and the journal says so before the run; `KEEP` and `SPLIT` carry it over.

The timecode track is a data stream too, and it is left behind on purpose: FFmpeg cannot name its codec, MP4 stops on such a stream («Could not find tag for codec none»), and the timecode itself travels as a value anyway. The journal says so in words, with the name of the track:

```
[WARNING] The data stream (tmcd) has no codec FFmpeg can name, so it cannot be
          copied. A timecode track reads exactly like this and is written as a
          value instead.
```

What was carried over is written too:

```
carried over: telemetry, subtitles
```

The rule is shared by the panel and the command line (`AUDION_TRIM_DATA`, `AUDION_TRIM_SUBTITLES`), and their files match byte for byte.

The result of a run and stopping

The status at the bottom of the window and the report tell the truth about the run. `Done` comes only if no piece failed and no row was rejected. An FFmpeg error and rejected points (for example, IN later than OUT) give `Failed`, a stop by the button gives `Stopped`. A result that is not overwritten because `Overwrite` is off is a skip on request, not an error.

A stop does not touch what has not started yet: a file of the next piece with the same name is from an earlier run, and with `Overwrite` off it is not this run's file. Only the unfinished file of the piece that was being written at the moment of the stop is removed.

Timecode

Editors sync cameras with each other and with the recorder by timecode, so it moves into the result and shifts by the length of the cut-off start:

```
timecode 01:00:00:00 -> 01:00:09:00
```

`Zero` instead of `Shift` resets the timecode of the result to zero. If the container cannot hold `tmcd`, the section says so before the run, not after it.

Sound files

WAV, FLAC, MP3, M4A, Opus, OGG and the other sound formats stand in the same badge, with a notes icon instead of a film strip, and are cut with the same three buttons. For a sound file the section hides what does not apply to it (the container, the video timecode, telemetry, subtitles) and shows its own fields.

Exact cut

The field «Cut» = «Exact». By default it is «Lossless» (below): «Exact» encodes compressed sound once more. Here the sound is decoded, cut **exactly on the sample** and written in the format chosen with the buttons below. There is no intermediate file: decoding, cutting and writing go in one FFmpeg pass.

A point becomes the nearest sample: 10.2345 s at 48 kHz is sample 491,256. The journal prints both numbers:

```
Split take.wav into 3 parts on the sample:
    at 00:00:10.235 (sample 491256)
    at 00:00:20.000 (sample 960005)
[1/3] take.wav [part1] | PCM_S24LE 48000 Hz 2 ch | 00:00:30.000 | -> WAV
    piece   00:00:00.000 (sample 0) .. 00:00:10.235 (sample 491256)
    keeping 491256 samples (00:00:10.235)
    writes  PCM_S24LE as it was, sample for sample
    result  491256 samples, 00:00:10.235 (+0 from the plan)
```

`result` is not a number from the file header but as many samples as the decoder gave: the headers of compressed formats differ from the sound (AAC has 29 samples fewer, Opus 312 more).

What comes out, by measurements on a 30-second stereo recording of 1,440,000 samples:

Source → result	What matched
WAV → WAV, FLAC → FLAC	the piece is **bit for bit** equal to the same span of the source; three parts in a row are the whole source
MP3, AAC, Vorbis → WAV	the piece is bit for bit equal to the same span of the **decoded** source
anything → MP3, Opus	the length is exact to the sample

Source → result	What matched
anything → M4A (AAC)	the encoder appends a tail: +243 samples to a piece of 468,749
anything → FLAC, ALAC	bit for bit, as long as the source is not deeper than 24 bits

Format of the piece

The buttons are the same set as in the AUDIO section:

Button	What is written
`As is`	WAV, FLAC, ALAC, WavPack, AIFF as themselves, sample for sample. MP3, AAC, Opus, Vorbis with the same codec once more, at the source bitrate: **one generation of compression**, and the journal says so
`WAV`	the decoded sound at the bit depth of the source; a compressed source is stored as float, exactly as it is decoded
`FLAC`, `ALAC`	lossless, up to 24 bits; a float or 32-bit source is stored in 24 bits, with a warning
`M4A / AAC`	AAC at the bitrate below
`MP3`	LAME: VBR V0, Insane 320 or by bitrate; a rate above 48 kHz is brought to 48 through soxr
`Opus`	at most 256 kbps; Opus works at 48 kHz, other rates are converted through soxr

For a compressed source, lossless means `WAV` or `FLAC`: everything heard after decoding is kept, and nothing more is lost after that.

`As is` refuses where FFmpeg cannot write the format back: for APE, TAK and DSD the section offers FLAC or WAV.

What travels with the sound

- **Broadcast WAV.** A recorder writes into the `bext` chunk a description, the originator, the date, the time and `time_reference`: the number of samples from midnight at which the take started. This is the sound timecode, and it shifts the same way as the video timecode: the three parts of a split at the points 491,256 and 960,005 got 172,800,000, 173,291,256 and 173,760,005. Without an explicit request FFmpeg loses the whole `bext` chunk, and it gives four of its fields under names

that it does not read itself when writing; the section passes them under the right names. Checked by reading the bytes of the chunk.

- **iXML** (track names, scene, take): FFmpeg neither reads nor writes it. The section warns: `bext` and the time travel, iXML stays behind.
- **The cover** travels into MP3, M4A, FLAC and ALAC; WAV and Opus do not take it, and the journal says so.
- In FLAC, MP3 and M4A `time_reference` travels as a tag, shifted too.
- Of several sound tracks, the first one is cut, and this is said.

Lossless - the default

Nothing is decoded or encoded: the packets of the compressed sound are copied untouched, and no generation is spent. The price: the point moves to the **nearest packet boundary** (MP3 1,152 samples, AAC 1,024, Vorbis up to 1,024, Opus 960), and the join sounds different from the source. How much is measured for each codec by decoding the parts and comparing them with the decoded source:

Codec	Container	What the join costs
MP3	`mp3`	the first 34 ms after the join (1,632 samples): a frame borrows bits from the previous one; a part that runs to the end of the file but does not start at the start carries the trailing padding of the encoder, less than a frame
AAC	`m4a`, `m4b`	the first 21 ms after the join (1,024 samples): the frames overlap
Vorbis	`ogg`, `oga`	the part is copied one packet earlier (the decoder throws away the first packet) and is bit for bit after that; the join of a splice costs 21 ms
Opus	`opus`, `ogg`, `oga`	about half a second, while the decoder recovers; a part that does not start at the start loses the pre-skip, 312 samples

Without the corrections it is worse, and this is measured too: MP3 split into three parts loses 4,625 samples and shifts the sound after the first join (FFmpeg writes into each part a header with the encoder delay of the source, and the decoder cuts it off again); the AAC join comes out 1,024 samples longer and shifted as a whole, unless the clock of the join is moved back by the length of the primer.

WAV, FLAC, ALAC and AIFF are cut on the sample with «Lossless» too: they have nothing to lose, and the journal says just that. Codecs and containers that are not in the table are not offered in this mode: the section asks for «Exact».

```
Copying song.m4a packet by packet (Exact cut is off):
    point 00:00:10.235 -> packet boundary 00:00:10.240 (+5.5 ms, sample 491520)
    point 00:00:20.000 -> packet boundary 00:00:20.011 (+10.6 ms, sample 960512)
[1/1] song.m4a | AAC 48000 Hz 2 ch | 00:00:30.000 | -> M4A
    remove  00:00:10.240 (sample 491520) .. 00:00:20.011 (sample 960512)
    keeping 971008 samples (00:00:20.229)
    writes  AAC packets, copied untouched
```

When the section refuses: before the run

Case	What it says
Camera RAW in MXF (Sony X-OCN, ARRIRAW)	FFmpeg does not read such a stream at all, so there is nothing to cut; the vendor's program is named
MP4 + ProRes, DNxHR, FFV1, HuffYUV, UTVideo, v210, CineForm	the container does not take such video; MOV or MKV is offered
MOV + AV1 or VP9	the same
MXF + HEVC, AV1, VP9	MXF does not hold these
MXF + compressed sound	MXF takes only PCM
MOV + FLAC, Opus, Vorbis	the container does not take such sound
A channel that the file does not have is requested	it says how many channels the file actually has
Both points are empty	«Nothing would be trimmed: no cut point is set.»
Split: both points on one keyframe or sample	the middle part would be empty; move the points apart
Sound file `As is`: APE, TAK, DSD and others that FFmpeg does not write	FLAC or WAV is offered
MP3 from a file with three or more channels	MP3 holds one or two channels
Packet copying for a codec or container that was not measured	it asks to turn exact cut on

Readable video extensions: `avi`, `m2ts`, `m4v`, `mkv`, `mov`, `mp4`, `mts`, `mxf`, `ts`. Sound: `aac`, `ac3`, `aif`, `aiff`, `amr`, `ape`, `au`, `awb`, `caf`, `dff`, `dsf`, `dts`, `eac3`, `flac`, `m4a`, `m4b`, `mka`, `mp3`, `mpc`, `mpc8`, `oga`, `ogg`, `opus`, `snd`, `spx`, `tak`, `tta`, `w64`, `wav`, `wv`.

The player

mpv starts as **a separate program** and listens for commands over a channel, the same way `ffmpeg.exe` is used here. It is installed once: `install\Install-Portable-mpv.cmd`.

What this gives:

- exact seeking: it lands on the requested frame, not on the nearest keyframe;
- a step of exactly one frame in both directions;
- playback **backwards** (`b`);
- the A/B loop, which is the cut itself, heard before the cut;
- the same camera formats that the program itself reads: inside mpv is the same FFmpeg.

The window can be kept on a second monitor; it survives a change of file, because reopening is a command, not a restart. If you close it by hand, the panel sees this, and the `Open in the player` button raises it again. The player closes by itself together with the program.

The installer puts next to the player the license text (`LICENSE.GPL`, `COPYRIGHT`; the build archive does not have them) and the sources the player is built from: mpv itself at the commit of the build, the FFmpeg built into it (this is **another** FFmpeg, not the one the program works with), and the build scripts, which list the versions of all the other libraries. About 24 MB; what was taken from where is written in `Tools\mpv\SOURCES.txt`.

The settings are in `config\mpv`. There is the shared player configuration of the Audion programs, and below it a short block of corrections for cutting: interpolation is off (it draws frames that are not in the file, and a point cannot be set by them), seeking is always exact, a file opens at the place it was opened to.

The keys are in `MPV_HOTKEYS_EN.md`.

From the command line

```
Scripts\ff-trim-start.cmd     remove the start
Scripts\ff-trim-end.cmd       remove the end
Scripts\ff-trim-both.cmd      keep the middle
Scripts\ff-trim-middle.cmd    cut out the middle and join
```

The points are set with environment variables: `AUDION_TRIM_FILE`, `AUDION_TRIM_START`, `AUDION_TRIM_END`; the container, sound, channel and timecode with `AUDION_TRIM_CONTAINER`, `AUDION_TRIM_AUDIO`, `AUDION_TRIM_AUDIO_CHANNEL`, `AUDION_TRIM_TIMECODE`.

The wrappers cut video; sound files are cut from the panel.

The arithmetic is shared with the panel, and this is checked, not claimed: the same cut, made by a wrapper and by the panel, gave **byte-for-byte identical files**.

The neighboring section: lossless rotation

Rotation of the picture lives in the PACKAGE section, because it is not a cut but a change of how the file is shown. The camera was turned on its side for a vertical frame, but it recorded an ordinary horizontal file and said nothing about it: here the rotation is written into the container, and the editing program opens the clip the right way up.

The pixels are not touched: a file of 6,085,307 bytes became 6,085,304, the frame hashes matched, the decoder gives 540×960 instead of 960×540. MP4, MOV and MKV keep the rotation; MXF does not, and the section refuses instead of giving an unrotated file. It is offered for camera H.264 and HEVC.