READY-TO-USE PROMPT - FOR GTA6 STREAMS ONLY

Viral Stream Clips

Make short viral 9:16 clips from a YouTube GTA6 live stream or an ended GTA6 stream (VOD)

Find the loudest moments, download only those sections in 1080p60, build split-layout clips (streamer face top 40%, gameplay bottom 60%), verify sync, and save them to Google Drive.

> **Scope:** This prompt is made for **GTA6 streams and GTA6 VODs**. It is not a general prompt for all streams.

How to start - ask first, then build

When the user pastes this skill and wants clips, do **NOT** start downloading anything. Follow this conversation, one question at a time:

1. **Ask for the YouTube video link first.** Ask: "YouTube video ka link do." Wait for the link. Do not guess a video.
2. **Check the link, then ask how many clips and how long each one is.** Once the link is given, check the video (live or ended, duration, top quality - see below) and ask: "Kitne clips nikalwane hain, aur har clip kitne seconds ki ho?" If the user does not give a number, suggest the usual default: **5 clips of 40 seconds each**.
3. Then run the rest of this skill - find the moments, download the sections, build the clips, verify sync, save to Google Drive, and deliver the links. Do not ask again for anything the user already answered.

If the user wants a different layout, length or count mid-way, take the new instruction and continue from the step it changes; do not restart the whole job.

Decide live vs ended first

Get the video facts before anything else:

```
yt-dlp --no-playlist --dump-json <URL>
```

Read title, channel, duration, `live_status`, `is_live`.

> **Ended stream (VOD) is the good case.** Download sections straight from the VOD (format 299+140). Audio and video carry exact PTS, so sync is perfect - both streams start at 0.000000. Prefer this route whenever the stream has ended.

Live stream is the hard case

A live stream's HLS DVR window is only **900s (15 min)** - older footage is not retrievable. `yt-dlp --live-from-start` downloads the whole stream (hours long) and is unusable for clips.

Working method for live: fetch the video + audio HLS playlist URLs (`yt-dlp -g`), take the last N segment URLs, download in parallel, concat with the concat demuxer (`-c copy`). Script: `~/workspace/yt-clips/dl_window.py`.

> **AV sync trap (live HLS only):** video and audio playlists fetched seconds apart cover different absolute windows, so aligning segments by position gives a ~2s audio delay (voice lags lips). Fetch both playlists simultaneously and match segments by media sequence number, or shift the audio by one segment. VOD sections do not have this problem - never apply the shift to VOD clips.

Tell the user plainly when their favourite moments are outside the 15-min DVR window and can only be recovered once the stream ends and becomes a VOD.

Find the viral moments

1. Download the audio-only track first (`yt-dlp -f ba`): ~200 MB for a 4-hour VOD, much faster than video.
2. Run one `astats` pass over the whole audio to get per-second RMS levels, then pick the best windows of the length the user asked for (default 40s), as many as they asked for (default 5), by mean RMS with at least **180s separation** between starts, spread across the stream.
3. The loudest window is often the intro music or a "STREAM STARTING SOON" screen - check every picked window with a frame before building. Replace a non-gameplay window with the next-loudest candidate.
4. Verify each section with `ffprobe` (expect 1920x1080, 60fps, ~40s) and extract a mid-frame (`-ss 20`) to confirm it is real gameplay and to see the streamer's facecam position and any on-screen overlays.

Download the sections in 1080p60

Quality rule - always use the highest quality YouTube offers for that video. Never hardcode a format id. Run `yt-dlp -F <URL>` first and pick the highest resolution, then the highest bitrate.

- Prefer `mp4_dash` for section downloads (it supports `--download-sections` reliably); an HLS (m3u8) format with a higher bitrate is worth trying, but fall back to the best DASH format if sections fail.
- For the CarryisLive VOD the top was 299 (1920x1080 60fps, 4014k DASH); 312 was the same resolution at a higher bitrate (6075k HLS).
- If a stream offers 1440p or 2160p, use that instead - and scale all crop values below proportionally (they are written for a 1920x1080 source).

```
yt-dlp --no-playlist -f "<BEST_VIDEO>+140" --download-sections "*HH:MM:SS-HH:MM:SS" -o <file>.mp4 <URL>
```

yt-dlp downloads only the fragments in range, so each 40s section is ~20-30 MB at 1080p.

- Verify the downloaded section really is the chosen resolution with `ffprobe` (expect 1920x1080 or higher, 60fps). If it is not, re-check `-F` - the format list may have changed or the wrong id was used.
- yt-dlp extraction is flaky (sometimes rc=1, or ffmpeg exits with code 8 on merge). Retry the same command up to **3 times** - it usually succeeds.
- Download sections in parallel where possible; each run takes under a minute.

Locate the facecam

The facecam position differs per streamer. Find it with a frame check or `muse.visual_grounding` (bbox) on a mid-frame before building.

Streamer / Element	Crop (1920x1080 source)
Typical Gamer - facecam, bottom-left, no border	`crop=420:276:10:428`
CarryisLive - full branded box, bottom-left, purple border + star banner + logo bar. Aspect 400:280 almost exactly matches the 720x512 panel, so it fills the panel cleanly.	`crop=400:280:14:406`
Gameplay centre crop (both) - avoids the bottom-left facecam and minimap, and side banners / top-right alerts	`crop=1012:1080:454:0`

> If a new streamer appears, measure the facecam box first and record the crop here; never guess it from another streamer's layout.

Build the 9:16 split clips

Output: 720x1280 (9:16), 60fps, CRF 18, yuv420p, AAC 192k, +faststart. The source is 1080p, so say plainly the clips are HD from a Full HD source - never promise 1080x1920 output; the facecam would go soft from the upscale.

Filtergraph (face top 720x512, gameplay bottom 720x768, stacked):

```
[0:v]split=2[face][game];
[face]crop=<FACE>,scale=720:512:flags=lanczos:force_original_aspect_ratio=increase,crop=720:512,setsar=1[face720];
[game]crop=1012:1080:454:0,scale=720:768:flags=lanczos,setsar=1[game720];
[face720][game720]vstack=inputs=2,format=yuv420p[v]
```

- Build **one clip first** and check its frame before batching the rest - the facecam framing (banner/logo slivers, border) is the thing that goes wrong.
- Script for CarryisLive: `~/workspace/yt-clips/build-carry-clips.sh` (build <section.mp4> <clip.mp4>).
- **Clip length:** whatever the user asked for (default 40s; earlier live-stream clips were 60s). Keep the user's requested length; do not change it silently.

Verify sync and quality before delivery

- ffprobe every finished clip: width 720, height 1280, 60fps, duration matching the request, and audio + video `start_time` both 0.000000.
- Check one mid-frame per clip: face panel framed cleanly, gameplay crisp, no facecam / minimap / banner leaking into the gameplay panel.
- If the user reports lip-sync still wrong on a live-HLS clip, do not claim the shift fixed it - the offset is rarely a single constant. Offer the three honest options: live-record while streaming, wait for the VOD, or DASH fragments matched by sequence number.

Save to Google Drive

1. Create one folder per stream, e.g. "GTA 6 Viral Clips - CarryisLive", with drive files create (folder MIME type, parent root, ignoreDefaultVisibility: true).
2. Upload each clip with `drive +upload <ABSOLUTE_PATH> --parent <FOLDER_ID> --name <FILENAME>`. Uploads ask the user for approval on a card - one approval covers the whole batch. Wait for it; do not re-run uploads that are only waiting on approval.
3. Fetch each file's `webViewLink` with drive files get (fields id, name, webViewLink, size) and deliver the links by clip name and moment, in stream order.
4. Local files stay on disk after any link expires; offer a fresh link on request.

Standing rules for this work

- One short copyright note when the source is another creator's stream (reposting can be claimed) - say it once, then do the work.
- Never log in to Facebook or connect a Facebook account for this work.
- Give a realistic time estimate before a long build and update the user while it runs (audio analysis of a 4-hour VOD takes ~5 min; each clip encode takes ~2 min).
- Record the finished clips in `~/MEMORY.md`: stream URL, channel, moments with timestamps, crop values, Drive folder id, file sizes.

Available connectors

Use `muse.connector_search` to find other connectors and `muse.skill_search` to find related skills. Google Drive uploads use the Google Drive connector. If nothing here fits, look on the web for a maintained API or MCP server and set it up with Custom Connectors and MCPs, or build a custom connector.

Alternatives for better results

- Offer the VOD route whenever a live stream has ended - it is the only way to get exact sync for past moments.
- If the user wants the clips posted instead of just saved, ask which platform and account, and do not post without their explicit approval.
- Offer captions and hashtags for the clips when the user plans to post them on Facebook or Reels.

Quick checklist

1. Ask for the link. Ask for clip count and length.
2. Check live vs ended. Prefer the VOD route.
3. Find loudest moments with audio RMS. Check frames.
4. Download sections at the highest quality. Verify with ffprobe.
5. Measure the facecam. Build one clip and check it.
6. Build all clips at 720x1280, 60fps. Verify sync.
7. Upload to Google Drive. Deliver links in stream order.

SAJID OFFICIAL

Thank you for using this prompt.

Get more premium tools, prompts and digital services at

sajidofficial.com

Sajid Kharal | @TheSajidKharal