# Snoober's Guide to OBS Studio

## Introduction

helo im snobar. This guide is intended for those who wish to use OBS Studio to stream online and/or record video to a local file (optionally to upload online post-recording).

Part 1 will discuss recording and streaming. You will learn the differences between encoders, codecs, and file formats as well as different bitrate modes and what to choose when recording and streaming. After applying these settings in OBS, you will learn how to choose your audio settings as well as some optional tweaks you can make to your microphone.

Part 2 will discuss editing. You will learn what software tools you will need as well as get a quick overview of available free video editing software. You will then learn how to use two optional editing workflows: *intermediate codecs* and *proxies*. Finally, you will learn about exporting your video to a proper delivery codec.

Part 3 will finish the guide off by discussing your options for uploading to online video hosting services.

## Part 1: Recording + Streaming

Which encoder and settings you use depends on what you want to do:

1. Record locally only
2. Stream online only
3. Stream + save a local recording with the same quality as the stream
4. Stream + save a local recording of higher quality than the stream

Depending on your hardware, you likely have multiple encoders to choose from. You have a software encoder called "**x264**" (this uses the CPU to encode). You may also have hardware encoders. For instance, if you have an Nvidia GPU you will have **NVENC** encoder which uses your GPU to encode.

If you are going to stream and record with high quality (option 4 from above), you most likely want to use different encoders utilizing different hardware in order to split the work.

## Encoders vs codecs vs file formats

It's important to understand the difference between these three terms.

Encoders are algorithms that encode data so that the data ends up adhering to a *codecs* "rules" for how the data should be stored. There can be different encoders that use

different algorithms to encode to the same codec. For instance, both x264 and NVENC are encoders which encode to H.264 codec.

Codecs are the "rules" for how the data is stored. For media such as video, audio, and images, codecs often use compression to save bandwidth. H.264 is the most common codec for videos. VP9 is also becoming more common (.webm file format uses the VP9 video codec). *(H.264 and VP9 are both delivery codecs, designed for the final format of the video. There are different codecs that are designed for editing which will be discussed later).*

File formats are containers that "wrap" the codec and determine things like how metadata is written and read. File formats can support various codecs, and codecs can be supported by various file formats. For instance, the following are some file formats that support H.264 codec: .mp4, .avi, .mkv, .flv, .mov.

# Which encoder(s) to choose?

This guide will discuss two encoders: x264 and NVENC:

### x264

x264 is a free software encoder. Being a software encoder it utilizes the CPU. x264 is included with your OBS installation. x264 encodes to the H.264 codec.

### NVENC

NVENC (NVidia ENCoder) is a proprietary hardware encoder that comes with Nvidia GPU's and uses the GPU to do the encoding. It also encodes to the H.264 codec.

---

x264 is said to be a little more efficient than NVENC. This means that for a given quality x264 will use less bandwidth (bandwidth here refers to both data uploaded to a streaming service as well as filesize if the video is saved locally). Note that this does *not* mean that NVENC results in lower quality video, just that NVENC uses more data for a given quality.

**This makes x264 the choice for streaming**, as you are limited by your upload rate so bandwidth here is important. NVENC, however, often has less of a performance hit for your frames-per-second as games today tend to be CPU-bottlenecked. For most set-ups that means **it's probably the better choice to use NVENC for local recording**.

If you are streaming and simply want to keep a local copy without an improvement in quality over your stream, then you can opt to "use stream encoder" in the recording settings in OBS (shown later). In this case, your PC won't be doing any extra work besides encoding the stream, but you'll be missing out on potential quality increase that is obtainable in your local recording.

# Bitrate modes

Bitrate is simply the amount of data used per a unit of time for encoding videos. Bitrate is measured in bits/second. How exactly the encoder decides how much bitrate to use is a very different choice for streaming versus recording locally. This guide will talk about two bitrate modes: CBR and CRF/CQP.

## CBR - constant bitrate

Constant bit rate is about what it sounds like. Pick an amount of bits per second and the encoder will aim to use that many bits constantly.

It's possible to choose a bitrate where relatively simple scenes look fine but more complex scenes will lose quality as there isn't enough bitrate available. So quality can fluctuate in CBR mode.

Additionally, picking a bitrate that is too high will use unnecessary resources to encode as well as unnecessary amount of bandwidth.

## CRF - constant rate factor

You can think of CRF as constant quality. You set a desired quality for CRF mode and the encoder adjusts the bitrate to achieve that quality.

So simple scenes will lower the bitrate, saving you processing power as well as bandwidth. Complex scenes will use high bitrates but will still retain whatever quality you set the CRF to.

NVENC uses its own version of CRF called CQP. It's not known exactly how it works as NVENC is not open-source but it is equivalent to x264's CRF mode in the practical sense.

CRF/CQP quality values range from 0-51 with lower numbers being of higher quality (lower is less compression, 0 being lossless). Around 18 is said to be "visually lossless". Any quality increase by setting CRF lower than ~18 should not be noticeable by the human eye.

*(x264 also has a bitrate mode called CQP which is similar to CRF but not to be confused with NVENC's CQP mode. There is no reason to use x264's CQP over CRF, which is why OBS doesn't offer it as an option. For the curious: with CQP you can set the quantization parameter which refers to how much compression is used. CRF changes the quantization parameter for you, increasing compression for lower complexity scenes where higher bitrate isn't needed. This saves you bandwidth while still maintaining a constant quality.)*

A final note is that while it's possible to encode in lossless mode with H.264, the codec really isn't designed for it. I strongly recommend against recording in lossless mode. Even high-end systems will most likely not be able to playback lossless H.264 videos smoothly and file sizes will be huge. And as said earlier, there is no visually-detectable increase in quality between lossless and ~18 CRF.

### Which bitrate mode to use for streaming? Which to use for recording?

#### The simple answer is use CBR for streaming, CRF/CQP for recording.

Twitch recommends CBR for streaming as having a constant bitrate is important for technical reasons. Viewers may have trouble watching if the bitrate is fluctuating too much.

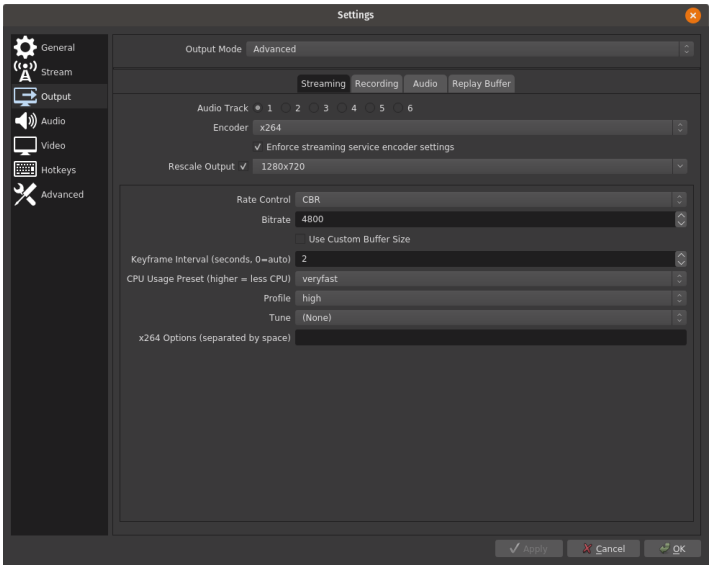
For CBR, pick a bitrate that is ~80% of your upload rate. Try a few different speedtests to get your upload rate (ISP's tend to "lie" by prioritizing speedtest traffic, so use a variety of speedtests if you want an accurate result).

For CRF/CQP, start with 18 as I recommend above for "visually lossless". You may need to increase this value (decreasing quality) if your CPU/GPU struggles.

## Output and Video Settings

Set **"Output Mode"** to advanced then take a look at the following settings:

### Streaming Output Settings




See https://stream.twitch.tv/encoding/ for streaming guidelines. These guidelines will tell you what resolutions+framerates you can likely get away with for your bitrate. It also has a few other settings recommendations.

This is where you most likely want to pick x264 encoder as it being somewhat more efficient than NVENC means a given bitrate will give you better quality.

We will come back to **rescaling** down below under **"Video Settings"**.

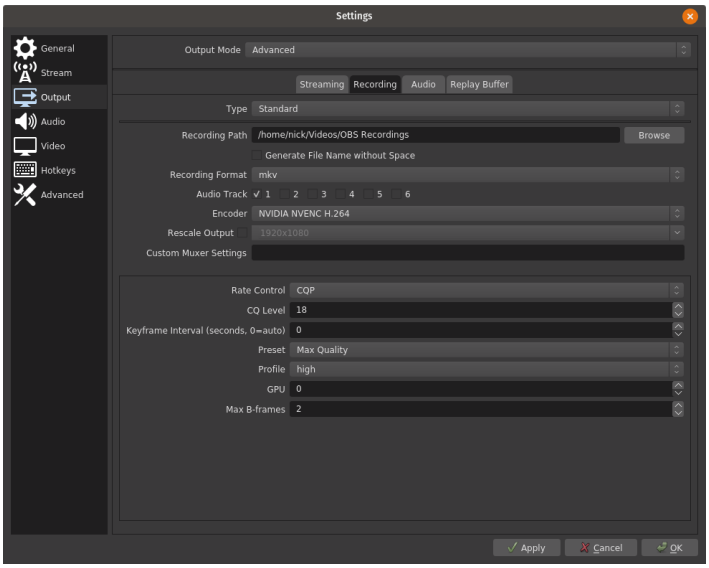
You can see that I use 4800 (kbps) for my bitrate which is about ~80% of my upload rate determined by various speedtests.

See the twitch link above for keyframe and profile recommendations.

For **CPU Usage Preset** start with veryfast. "Faster" settings, such as "ultrafast", are *easier* on your CPU and in CBR mode will result in lower quality video output. You may try to set this to a "faster"/easier setting if your PC lags while recording (alternatively you may need to downscale to a lower resolution).

---

## Recording Output Settings




For **recording format** it's probably best to choose .mkv. You may need to *remux* your videos to .mp4 later. This will be explained later in the guide but is relatively straight forward and quick. *Why don't we choose .mp4 here for our recording format?* The problem with .mp4 is that if the recording gets interrupted (program crashes, computer crashes, etc.) then your whole video file is lost. With .mkv this is not the case.

Choose your **encoder**. This is where you would pick "use stream encoder" if you want to stream and record with the exact same quality. If you are streaming and recording but would like better quality recordings then NVENC is the way to go for your recording encoder. If you are recording and *not* streaming then NVENC is still probably the best choice as it will usually result in less of an FPS hit than using x264 (even with the same quality recordings).

We are using CQP set to 18 for **rate control** as discussed earlier. You may need to set this value higher (=lower quality) if your PC has trouble.

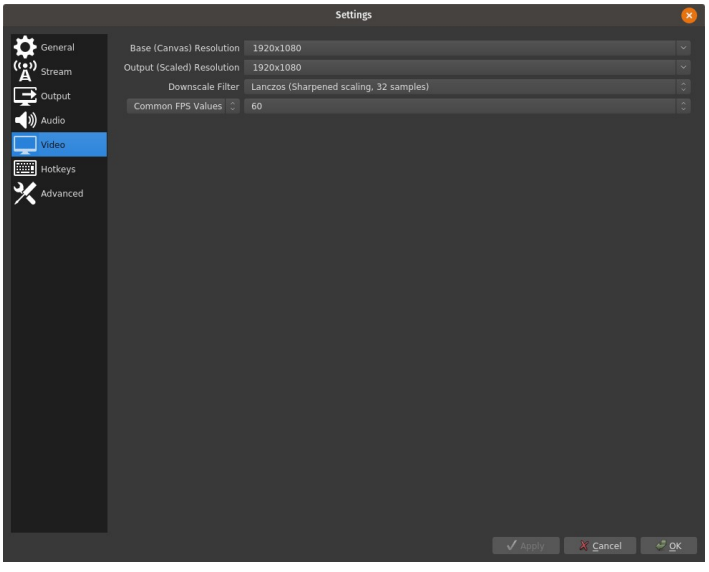
Start with **max quality** preset if you have a decent PC. This may be the first setting you try to lower if your recordings and/or PC are laggy.

**GPU** setting is used if you have multiple GPU's in your PC (you can select which GPU to use with this value).

I leave **keyframe interval** and **max b-frames** at default. These two concepts are beyond the scope of this guide, but if you want to know more you may want to start here.

---

## Video Settings




Set the **base (canvas) resolution** equal to your game resolution.

If you are streaming and recording at different output resolutions then set the **output (scaled) resolution** here in the **video** settings to your *highest output resolution* between streaming and recording (most likely your recording output resolution). Then go back to **output** settings and check **rescale output** and select the resolution to rescale to.

If you are only streaming or only recording then set the **output (scaled) resolution** here in **video** settings to your desired output resolution (this might be the same as your base resolution if you are not downscaling at all). In this case, don't rescale the resolution in **output** settings.

**Lanczos** is generally the best downscale filter.

**Video** settings is also where you may set your desired **FPS**.

---

# Audio

In the **output** settings under the "Audio" tab I recommend 160kbps. You can try higher bitrates for local recordings and see if there is any noticeable difference. Twitch recommends a maximum of 160kpbs since many mobile devices will not support higher audio bitrates.

In the main **audio** settings you can select either 41kHz or 48kHz sample rate (shouldn't make any noticeable difference).

You may find you need to set some filters for your microphone. On the **mixer** dock (if hidden go to View --> Docks --> and check Mixer) click the cog wheel next to your microphone and click filters. You can experiment with some filters. For instance, I use a gain filter to adjust my mic volume (relative to desktop audio) and a noise gate to try to avoid picking up keyboard sounds (or any sound that isn't my voice). You'll have to fine-tune these settings yourself as they will need to be different for every set-up.

Additionally, you *can* do some fancy things with audio tracks. For instance, you can have the game volume, background music, discord output, and your own microphone output all on different tracks. This kind of set-up would allow you to do a lot of editing with the audio (for example you could mute the discord track if your friend started yelling or something). Doing this requires other software, however, and would require a whole other guide...(if I write one eventually I'll link it here, otherwise look up guides for using "Voicemeeter" software).

# Record!

Try recording some gameplay and see how it turns out.

If your PC is laggy during recording, you will need to try changing some of the settings mentioned earlier. Perhaps a faster CPU usage preset, lower quality (higher CRF/CQP value), or downscale to a lower resolution.

# Other Tips

I recommend using an OBS plug-in called infowriter. This allows you to set up a hotkey that will log a timestamp to a text file of where you are currently in your recording. It's great when you are recording for long periods of time and don't want to have to search through hours of your video just to find that 30-second clip.

Set up some other hotkeys as well. You can set up hotkeys to show/hide sources. For instance, one of my sources is just a cropped capture of my music streaming window (this shows the song name/artist in the corner of the video). I can use a hotkey to show/hide it. You can set up all sorts of complicated scenes/sources and switch them around with hotkeys if needed.

# Part 2: Editing

So you have your OBS recording set-up and need to move onto editing. Most editing that you want to do probably involves making short clips that occur during your recordings. Of course you can take editing as far as you want, but this guide will not go into using the actual editing software (which is different depending on which software you choose). A quick overview of available free editing software will given, but then this guide will move onto *editing workflows* and how to implement them. This should be helpful especially for those that proceed to get more deeply involved in editing your videos.

# Software

FFmpeg - FFmpeg is a free, open-source, command-line tool for doing all kinds of work involving decoding/encoding videos. It's the leading standard tool for this kind work (OBS actually uses it under the hood). The command line may be a little uncomfortable for those not familiar with command line interface (CLI) programs. There are a number of GUI frontends available for FFmpeg that you can choose to use instead. The most popular being HandBrake.

*Note: If you are on Windows you need to add FFmpeg to your PATH environment variable. If you are unsure how to do this see this wikihow.*

Editing software - You will need some program to do the actual editing. There is a decent amount of free editing software available. Here are a few popular choices:

- Kdenlive | Windows, Mac, Linux | Open-source
    - This is my favorite. It has it’s own internal support for proxies and render previews (can generate a proxy format for sections of your timeline while you edit).
    - Kdenlive is primarily made for Linux but also supports Windows and Mac.
- Shotcut | Windows, Mac, Linux | Open-source
    - Another free open-source option similar to Kdenlive.
- DaVinci Resolve | Windows, Mac, Linux | Proprietary
    - This is by far the most sophisticated and complete video editor you can get for free. It's professional-grade software. It's simply more than most hobbyist will need and will take a bit to learn. You will be forced to follow a "professional" work-flow (regarding use of intermediate codecs or proxies, discussed below) which for editing a simple clip may be overkill. If you want to dive into video editing a bit more than simple edits then DaVinci Resolve is the way to go. For most of us it may be too involved.
- Others that I have not tried
    - Lightworks | Windows, Mac, Linux | Proprietary
        - only supports up to 720p in the free version
    - Blender | Windows, Mac, Linux | Open-source

        - designed for 3D modeling but can do video editing as well
    - Openshot | Windows, Mac, Linux | Open-source

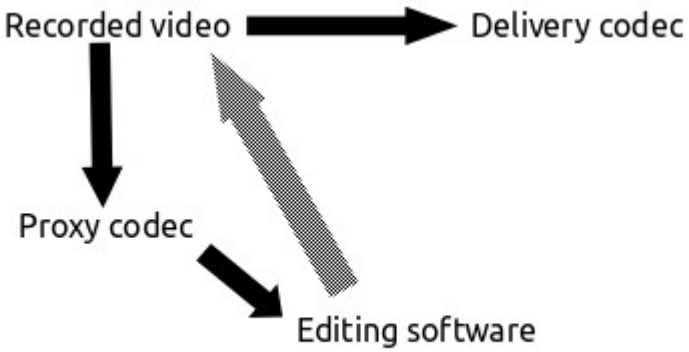
# Intermediate Codecs & Proxies

After your recording, your video is in H.264 codec (it may be .mkv, .mp4, or another file format that you have chosen, but it is encoded in H.264 codec). H.264 is really awful for editing. This is because H.264 compresses the data a TON. This is great for filesizes and uploading to a stream but it means that your CPU has to do a lot of work to decode the video. Playing back video is still smooth even with all of the work that your CPU has to do (unless you are using a slow PC) as H.264 is designed for playback (learn how H.264 compresses data with GOP). Editing, however, requires jumping back and forth. For H.264 this is SLOW.

There are two solutions to this problem. One is to use **intermediate codecs**. Intermediate codecs are codecs that are designed for editing. They use very little compression and while not technically lossless they lose very little quality over multiple encodings (so little that they are used professionally). The workflow for using intermediate codecs looks like:

Recorded video ➔ Intermediate codec ➔ Delivery codec

The other solution is to use a **proxy**. Using a proxy involves encoding to an editor-friendly codec (intermediate codec) except that when you render to a delivery codec after editing, you render directly from the recorded video instead of from the proxy. So the proxy is only used to "mark" your time-dependent edits, but those edits are then directly applied to the recorded video. This also means you can encode with lower quality to your intermediate codec when using it as a proxy since any quality loss won't affect the delivery video. Here is what the workflow looks like when using a proxy:




You can see that the editing software uses the proxy so that you can move around the video quickly while editing, but then your edits are ultimately applied to the original source material (in our case the recorded video).

## Should I use intermediate/proxy workflows?

It does add extra time and complexity to include the extra step of encoding your media to an intermediate/proxy codec. If you are only going to do simple edits (say just cut a clip and maybe add a fade in+out) it probably isn't worth the time to use intermediate/proxy codecs. However, if you plan on doing more than quick, simple edits I would definitely recommend using an intermediate codec or proxy.

A last thing to consider is that it's definitely "less annoying" to work with intermediate/proxy material over H.264 material in the editor.

## How to use FFmpeg to generate an intermediate codec or proxy

There are three commonly used intermediate codecs: DNxHD/DNxHR, Cineform, and Prores. This guide is going to show you how to encode to DNxHD/DNxHR. DNxHR is essentially the same as DNxHD except with additional support for >1080p resolutions. It also uses some different command arguments in FFmpeg.

DNxHD has various profiles and you must select one to convert to. See a nicely organized list on wikipedia. As an example, let's say that we have recorded a video in OBS that is 1080p and 60 FPS. There are four choices that DNxHD offers for 1080p/60fps media:

| Profile | Bits | Megabits per second |
|---|---|---|
| Avid DNxHD 440x | 10 | 440 |
| Avid DNxHD 440 | 8 | 440 |
| Avid DNxHD 290 | 8 | 291 |
| Avid DNxHD 90 | 8 | 90 |

440x is for 10-bit depth media also called "deep color" or "HDR". We aren't recording in 10-bit (your game most likely is not in 10-bit and your monitor most likely won't support 10-bit). That leaves us with 440, 290, and 90. These all refer to the bitrate (in Mbps). You will notice that 90-440 Mbps is MUCH higher than the bitrates you are streaming with in CBR as well as the resulting bitrates from recording in CRF/CQP mode. That's due to the fact that H.264 has compressed the video data and DNxHD decompresses the data and leaves it (mostly) decompressed. You might be confused since you probably know that H.264 is a *lossy* codec (unless you set CRF/CQP = 0). However, it also compresses a lot of data with *lossless* methods similar to when you make a zip archive. So DNxHD is "un-zipping" the lossless data and leaving it "un-zipped".

With that said, DNxHD 90 even with it's 90 Mbps is going to hurt the quality of the video quite a bit. That's because this profile is designed for *proxies*. So if you plan to follow the **intermediate workflow** outlined earlier you need to choose between DNxHD 290 and 440 (for 1080p/60fps video). *Will you lose quality between 290 and 440?* That depends on how complex your video is. If you have the disk space (and you should only keep these

intermediate files temporarily until you are finished editing unless you really don't mind using up storage space) then you might as well use 440.

You'll notice that every resolution/fps configuration has four profiles that correspond to the ones above just with different bitrate values. So in every configuration you will have the option of a high bitrate 10-bit, high bitrate, medium bitrate, and low bitrate (for proxies).

Here's the FFmpeg command to convert to DNxHD 440 for a recording called "obs-recording.mkv":

```
ffmpeg -i "obs-recording.mkv" -c:v dnxhd -b:v 440M -c:a pcm_s16le "out.mxf"
```

This tells FFmpeg the input file name is "obs-recording.mkv", the video codec is DNxHD, the video bitrate is 440 Mbps, the audio codec is pulse-code modulation signed 16-bit little endian (what?? see below), and the output name we would like is "out.mxf". The format needs to be something that DNxHD supports which includes ".mxf" or ".mov".

If you want to use DNxHR instead of DNxHD you use a profile switch instead of specifying the bitrate (if you work with >1080p resolution then you must use DNxHR). The profile choices correspond to our bitrate options and are as follows:

- dnxhr_444
- dnxhr_hqx
- dnxhr_hq
- dnxhr_sq
- dnxhr_lb

*The extra option, dnxhr_444 is for 10-bit video only. The 444 refers to a different chroma-subsampling which is not going to be discussed in this guide.*

Here's an example of our command with DNxHR syntax:

```
ffmpeg -i "obs-recording.mkv" -c:v dnxhd -profile:v dnxhr_hq -c:a pcm_s16le "out.mxf"
```

The encoder for both DNxHD and DNxHR should properly detect that the video is 1080p and 60 fps. If not, you can add these arguments to force FFmpeg to encode to 1080p/60fps:

```
-vf scale=1920x1080 -r 60
```

**What is pcm_s16le?** This means pulse-code modulation signed 16-bit little endian. *What??* This is one of those things that you can just blindly use when you are encoding to DNxHD, as PCM audio handles re-encodings much better than AAC. My simple explanation of PCM is that it is a way to describe an analog audio signal digitally. 16-bit is the bit depth. The rest ("signed", "little endian") describe how it's formatted in binary. You could optionally use 24-bit with "pcm_s24le" and it *might* make a slight difference in retaining quality.

Now you can use this intermediate format for your editing software.

If instead you wanted to use a **proxy-workflow** then you could optionally make the proxy a lower quality. So in our case with 1080p/60fps video we could use DNxHD 90 (or DNxHR with the dnxhr_lb profile). We could also *downscale* to a lower resolution to make things even faster. For example we could downscale to 720p by adding the argument "-vf scale=1280x720" to our FFmpeg command (in this case you would need to choose from different bitrates if using DNxHD from the wikipedia list). Remember, none of this quality loss will result in our delivery format if we are following the proxy-workflow.

How exactly to implement a proxy-workflow depends on which software you use. Some don't explicitly support proxies, but you can often "trick" your software into using proxies by swapping a proxy with your source file while you edit, then swapping it back out when you render. If you use DaVinci Resolve you can generate proxies within the software (Resolve calls it "optimized media").

## Is there a way to convert to an intermediate codec for a section of video around my clip without needing to convert the whole (possibly hours long!) recording?

Yes! Fortunately there is *seeking* support in FFmpeg. You can specify a start time with the "-ss" option (this must be used *before* the input option, "-i"). You can then specify either a duration with "-t" or an ending time with "-to" (note that the timestamps get reset to 0 using these options so keep that in mind while using "-to"). You will want to pick start and end times that have some extra room around your clip to work with.

You will notice that with this method the beginning of your "cut" will look "garbled". This is due to how *keyframes* work; keyframes are essentially a full snapshot while other frames store only the differences between themselves and the keyframes. So the cut video looks "garbled" until it gets to the next keyframe. This is why you should specify some extra time around your desired clip (especially at the start time). However, sometimes this will still cause problems with certain editors. One solution is to force FFmpeg to find the nearest keyframe when making cuts. To enable this behavior include the "-noaccurate_seek" option in your FFmpeg command (must be *before* the input switch, "-i").

Lastly, when I have used seeking and converted to another codec simultaneously I have experienced issues with audio sync. Therefore, I recommend using the “copy” option for codecs, which tells FFmpeg to not re-encode (we will do that after).

Here's an example FFmpeg command that cuts a 1 minute and 30 seconds section of video starting at 10 minutes at the nearest keyframes:

```
ffmpeg -ss 00:10:00.0 -noaccurate_seek -i “obs-recording.mkv” -t 00:01:30.0 -c:v copy -c:a copy “seek-cut.mkv”
```

The resulting cut won't be exactly 1 minute and 30 seconds since FFmpeg will have cut at the nearest keyframes instead.

Now we can convert to our intermediate codec:

```
ffmpeg -i “seek-cut.mp4” -c:v dnxhd -b:v 440M -c:a pcm_s16le “intermediate.mxf”
```

Here is a full guide on seeking in FFmpeg.

# Remuxing

If you ever need to change the *file format* but not the *codec* then you can perform a remux. For instance, your editing software might not recognize a certain file format but it can recognize the same codec wrapped in a different file format. This might happen in a situation where you have opted to not use intermediate/proxy codecs and have recorded in .mkv. Some software will recognize H.264 in .mp4 format but not in .mkv. We can remux the file by using the "copy" argument for our codecs as above but specify a different file format in the output name:

ffmpeg -i "obs-recording.mkv" -c:v copy -c:a copy "out.mp4"

This process is very fast as there is no re-encoding (and no quality loss). You can combine it with *seeking* to cut out a section of video without re-encoding (as described above). This is also a great way to save a section of recorded video that you intend to edit at a later time, as you don’t need to hold onto the full recording.

You can also remux with OBS by going to "File ➔ Remux Recordings".

# Delivery Format

Once you're finished editing in your software of choice you will want the video in a delivery format. You most likely want an .mp4 video with H.264 codec or a .webm video with VP9 codec. Most software can export directly to these formats but it's usually better to export to an intermediate codec and then use FFmpeg to convert to delivery format.

See what kinds of formats your software can export to and decide if you want to take the extra step to export it to an intermediate codec (DNxHD for example) before finally converting to delivery.

Let's say that we again have 1080p/60fps video and have exported to DNxHD with a .mxf format. Let's convert to H.264:

ffmpeg -i "export.mxf" -c:v libx264 -crf 18 -c:a aac -b:a 160k "delivery.mp4"

This command will use CRF mode to achieve a constant-quality (set to 18 in this example) and AAC audio codec. AAC is really the best choice for .mp4 videos. For the audio bitrate select whatever you recorded in for OBS.

You may find that the resulting video is black. This is most likely because DNxHD has changed the *pixel format* to yuv422p while our video was recorded in yuv420p (unless you have changed an advanced setting in OBS which I do not recommend changing). Pixel format refers to different methods of *chroma subsampling* which will not be discussed in

this guide. To force FFmpeg to use yuv420p use the option "-pix_fmt yuv420p" in your FFmpeg command. Our command becomes:

```
ffmpeg -i "export.mxf" -c:v libx264 -crf 18 -pix_fmt yuv420p -c:a aac -b:a 160k "delivery.mp4"
```

You can also use .webm format and VP9 codec instead of .mp4 and H.264 (or you could output to both, of course). Why webm? One reason is that VP9 codec compresses the data more than H.264. This means it takes (quite a lot) longer to encode but uses less filesize. The other reason depends on if you plan on uploading your video to an online service and which service you plan to upload to. Some services result in better quality if you upload one codec versus another.

Encoding in VP9 has quite a few differences from encoding to H.264. I will provide the most similar FFmpeg command to what we did for H.264, but if you want to know more see Google's VP9 Encoding Guide (Google developed the VP9 codec), as well as FFmpeg's.

Here's the closest equivalent to our H.264 example:

```
ffmpeg -i "export.mxf" -c:v libvpx-vp9 -b:v 0 -crf 31 -c:a libopus -b:a 160k "delivery.webm"
```

Notice that you must set the video bitrate to 0 if you want to use CRF mode (with H.264 you can leave the bitrate argument out). Also, the CRF scale is different than in H.264. For VP9, this value depends on the resolution as well as the desired quality. See the "Quality" section of Google's VP9 Guide.

# Part 3: Uploading Online

Most of us will want to share clips online after editing them out of our recorded footage. There are many online options but some of the popular ones include:

- YouTube
- Gfycat
- Streamable
- Vimeo
- Twitch

Every service will re-encode your video according to their specifications. Which video codec is best to upload depends on which service you are uploading to. Generally, I find it's best to upload in the same codec that the site's final re-encoded version will be. Gfycat videos are VP9/webm, for instance, so in my experience uploading VP9/webm videos results in better quality for Gfycat.

YouTube provides their own recommended upload encoding settings. You'll notice they recommend H.264 codec. YouTube re-encodes to H.264 except in two cases: you have at least X followers/subscribers or your video is greater than 1080p. Some users have found that it is better to "trick" YouTube into using VP9 codec by upscaling their videos to >1080p (you could do this by using the "-vf scale=..." command from earlier). The trick

works in the sense that it will be in VP9, but in my experience the video ends up being the same or even slightly worse quality than H.264. It appears to me that the increase in quality is only present in videos for affiliates with enough followers/subscribers, and not for videos that are simply upscaled in resolution.

# That's it!

I hope this guide was useful and more people are able to get a proper recording set-up for their games (or whatever else they wish to record). Leave a comment if you notice any mistakes, have suggestions, or have any opinions on this guide!

*Snoober*

## Resources Linked

- Twitch Streaming Encoding Guidelines
- Wikipedia "Group of Pictures"
- OBS plug-in Infowriter
- FFmpeg
    - HandBrake
    - FFmpeg installation guide (Windows)
- Editing software:
    - Kdenlive
    - Shotcut
    - DaVinci Resolve
    - Lightworks
    - Blender
    - OpenShot
- List of DNxHD resolutions (profiles)
- Seeking in FFmpeg
- Google's VP9 encoding guide
- FFmpeg's VP9 encoding guide
- YouTube's recommended upload settings

*Update on Jan 6th, 2020:* Removed Hitfilm from list of recommended software. Kdenlive noted as a favorite with a little more detail about the software.