ffmpeg

1. 1. How to split video into 30 seconds chunks. This example creates 3 segments from the beginning of video starting at time 0 seconds of original video. i.e. <30s><30s><30s><... long big chunk>.

   Code:

```
ffmpeg -ss 00:00:30 -vsync 0 -t 00:00:30 -i webcam_2012-03-18_00_33_58.mp4 -vcodec copy ·

ffmpeg -ss 00:01:00 -vsync 0 -t 00:00:30 -i webcam_2012-03-18_00_33_58.mp4 -vcodec copy ·

ffmpeg -ss 00:01:30 -vsync 0 -t 00:00:30 -i webcam_2012-03-18_00_33_58.mp4 -vcodec copy ·
```

1. 1. How to extract a still JPEG image from a video at the 20 second spot.

   Code:

```
ffmpeg -i webcam_2012-03-18_00_33_58.mp4 -r 0.1 -t 20 image%3d.jpg
```

1. 1. Here are some other examples to save you time:

   19-FFmpeg commands for all needs

   2. Good articles and examples here:

   FFMpeg – The swiss army knife of Internet streaming – part I

   FFMpeg – The swiss army knife of Internet streaming – part II

   FFMpeg – The swiss army knife of Internet streaming – part III

   FFMpeg – The swiss army knife of Internet streaming – part IV

   3. Screencasting: How to capture your desktop screen and audio.

   Nice examples here.

   4. How to overlay text on the video at a specific window position.

Code:

```
ffmpeg -i sub_video3.mp4 -vf \
       drawtext="fontfile=/usr/share/fonts/truetype/ttf-dejavu/DejaVuSerif.ttf: \
       text='Text to write is this one, overlaid':fontsize=20:fontcolor=red:x=100:y=100"
       with_text3.mp4
```

1. 1. How to grab audio from your desktop/laptop microphone.

   Code:

   ```
   ffmpeg -f alsa -ac 2 -i pulse mic_test.mp3
   ```

1. 1. Two examples that overlays text over video which timestamps the output video to start at the specified time (timecode).
   Code:

   ```
   #Begin at 1:00

   ffmpeg -y -i in_video.mp4 \
          -vf "drawtext=fontcolor=white: fontsize=16: \
                        fontfile=/usr/share/fonts/truetype/ttf-dejavu/DejaVuSans.ttf: \
                        box=1:boxcolor=black@0.3:x=50:y=20: \
                        timecode='00\\:01\\:00\\;02':rate=30000/1001" \
           out.mp4

   #Begin at 3:59, with a specified codec

   ffmpeg -i in_video.mp4 \
          -vf "drawtext=fontfile=/usr/share/fonts/truetype/ttf-dejavu/DejaVuSans.ttf: \
                        x=10: y=20: fontsize=32: boxcolor=black@0.5:box=1: \
                        rate=30000/1001:timecode='00\\:03\\:59\\;27'" \
           -r 30000/1001 \
           -vcodec mpeg4 \
           nostra_timecode.mp4
   ```

1. 1. A example that overlays text over video which shows updating timestamps on the output video.
   Code:

   ```
   ffmpeg -i video.mp4 -vf drawtext="fontfile=/usr/share/fonts/truetype/ttf-dejavu/DejaVuSar
                           text='%T %D': x=10: y=10: fontsize=24: fontcolor=black" \
                           -vcodec libx264 -preset fast -crf 34 -threads 0 \
   ```

```
                        strftime.mp4
```

1. 1. A example that overlays text over an mjpeg input stream generated with mjpeg_streamer from a usb webcam.
   Code:

```
ffmpeg -fflags +genpts -t 600 -f mjpeg -r 8 -s 640x480 \
       -i  http://localhost:8080/?action=stream -vcodec mpeg4 \
       -vf drawtext="fontfile=/usr/share/fonts/TTF/mitra.ttf:x=70:y=455: \
                    text='\%H\:\%M\:\%S | \%a \%d/\%b/\%Y | S500ATV | camera 0': \
                    fontcolor=0xFFFFFFFF:fontsize=18: \
                    shadowcolor=0x000000EE:shadowx=1:shadowy=1" \
       -b 1500000 -r 8 \
       video_file.avi
```

1. 1. If you want to add subtitles to a video file (in_video.avi) with audio to a new output video with subtitles. Note that since video had audio we are not passing the audio file we just map audio to the 2nd (0:1) component of the output. Input had 2 components, 0 and 1. Output has three (3) components, 0, 1, and 2. So, we are mapping the 1st input component to the 1st and 2nd output components (0:0) and (0:1). We also map the 2nd input component to the 3rd out put component (1:2).
   Code:

```
ffmpeg -i in_video.avi     -c:v libx264 -vpre ipod640 -s 480x240 -b 256k -map 0:0 \
                           -c:a libfaac -ar 48000 -ab 128k -ac 2 -map 0:1 \
       -i in_subtitles.srt -c:s copy -map 1:2
       out_video_with_titles.m4v

#That is, follow this type of command format:

ffmpeg -i Input_1.avi -c:v copy -map 0:0 \
       -i Input_2.wav -c:a copy -map 1:1 \
       -i Input_3.srt -c:s copy -map 2:2
       Output.mp4
```

1. ffmpeg: the mother of all command-lines

   Nice complex command explained!

Re: HOWTO: Proper Screencasting on Linux

> **Note:** *This was the original post not done by me, but i will try to update if anything new comes up*
>
> Demo Video http://www.youtube.com/watch?v=Ewxm6T6rXP0
> While many screencasting tools exist on Linux, none of them is able to really pull a high-quality screencast. I've tried almost all existing tools such as recordmydesktop, xvidcap, istanbul, wink etc.. and all of them produced poor results. Horrible video/audio quality, bad sync, lots of limitations, and some even simply segfault while doing the recording. The real solution and the ultimate answer to all your screencasting needs on Linux has been there for quite a long time. It is called FFmpeg. I am surprised to see that many people would just overlook this awesome piece of software.
>
> ## What is FFmpeg?
>
> From the official website:
>
> Quote:
> FFmpeg is a complete, cross-platform solution to record, convert and stream audio and video. It includes libavcodec – the leading audio/video codec library.
> In this tutorial, I'll suppose you're using Ubuntu Linux Karmic or Lucid, but the tutorial should also work for other versions with slight modifications.
>
> ## Preparation:
>
> For this tutorial, you'll need FFmpeg (preferably a recent revision) compiled with "--enable-x11grab" and the following libraries:
>
> * libx264
> * libfaac
> * libvpx
> * libvorbis
> * libxvid
> * libmp3lame
> * libtheora
>
> Because the version of FFmpeg that is available in the repositories is not compiled with these libraries, you need to compile it yourself. There is an easy-to-follow guide for compiling FFmpeg on ubuntu with those libraries here:

http://ubuntuforums.org/showthread.php?t=786095
At this point, you should have the latest FFmpeg built with the needed encoding libraries. Let's go to our main subject:

Screencasting:

We will do screencasting in 2 steps:

* Capture a lossless, crystal-clear video stream with lossless audio.
* Encode the resulting lossless file to a compressed version suitable for internet use.

The reason we're doing it in a 2-step scheme (instead of just encoding directly as we capture) is that compressing a video with good quality takes some time and processing power that isn't possible on the fly. The idea is to use the first step to only capture lossless audio/video feeds as fast as possible and store them, then use the 2nd step to do the real compression and get a better result. The 2nd step is needed because the losslessly-captured file is tremendously large to be used on the web.

Step 1:

Just for the record, basic ffmpeg syntax is:

Code:
ffmpeg [input options] -i [input file] [output options] [output file]
Learn by example, fire up your terminal application and enter the following command:

Code:
ffmpeg -f alsa -ac 2 -i pulse -f x11grab -r 30 -s 1024x768 -i :0.0 -acodec pcm_s16le -vcodec libx264 -preset ultrafast -crf 0 -threads 0 output.mkv

Then press Enter to start the capturing process. To stop recording, go back to the terminal and press q.

In the above command, we capture audio from pulse (pulseaudio sound server) and encode it to lossless raw PCM with 2 audio channels (stereo). Then, we grab a video stream from x11 at a frame rate of 30 and a size of 1024×768 from the display :0.0 and encode it to lossless h264 using libx264. Using -threads 0 means automatic thread detection. If your distribution does not use the pulseaudio sound system, see FAQ section. The resulting streams will be muxed in a Matroska container (.mkv). The output file "output.mkv" will be saved to the current working directory.

Before we move to step 2, let’s look at things you can change in step 1. Obviously, the most important of which is the resolution. If you want to capture your entire desktop, then you have to enter the screen resolution you’re working at. It is also possible to capture a specific area of the screen by specifying a capture size that is smaller than the resolution. You can optionally offset this area by adding +X,Y after :0.0 which means it will look something like this:

Code:
-s 800x600 -i :0.0+200,100
This tells it to capture a rectangle of 800×600 with an X offset of 200 pixels and a Y offset of 100 pixels (the offset starting point is the top-left corner of the screen). Note that if you offset the capture area out of the screen, it will give you an error.

Another thing you can change is the video frame rate (FPS). In the example above we used -r 30 which means capture at 30 FPS. You can change this value to whatever frame rate you want.

Step 2:

Now that we have the lossless file, let’s encode it to suit our needs. From here on, it all depends on what you’re planning to do with your screencast, but we’ll provide some basic examples and from there you move on.

Example 1:

Code:
ffmpeg -i output.mkv -acodec libfaac -ab 128k -ac 2 -vcodec libx264 -preset slow -crf 22 -threads 0 our-final-product.mp4
In the above example, we encode the audio to AAC at a bitrate of 128k with 2 audio channels (stereo). We encode the video to the high quality H.264 video compression standard. We use the preset "slow" and a CRF value of 22 for rate control. The output file will be named “our-final-product.mp4″ and will be muxed in an .mp4 container. Note that FFmpeg determines the container format of the output file based on the extension you specify (i.e. if you specify the extension as .mkv, your file will be muxed into an .mkv matroska container). You can tweak the CRF value to get different results. The lower you set the CRF value, the better your video’s quality will be, and consequently the file size and encoding time will increase, and vice-versa.

Example 2:

WebM is a new, high-quality, free/open format that can be played in modern web browsers that support the HTML5 <video> tag. Since libvpx does not have a constant quality mode yet, we're doing the encode in 2 passes:

Pass 1:
Code:
ffmpeg -i output.mkv -an -vcodec libvpx -b 1000k -pass 1 our-final-product.webm
Pass 2:
Code:
ffmpeg -i output.mkv -acodec libvorbis -ab 128k -ac 2 -vcodec libvpx -b 1000k -threads 2 -pass 2 our-final-product.webm
Change the bitrate (-b 1000k) to control the size/quality tradeoff. Also, change the number of threads (-threads 2) to suit the number of threads your CPU has. If your CPU is not multi-threaded, you can omit the -threads option completely. If you have a modern web browser, you can open the file and play it natively inside it. A WebM file consists of VP8 video and Vorbis audio mulitplexed into a .webm container (which is basically a subset of the Matroska container, aka .mkv).

Example 3:

Code:
ffmpeg -ss 00:00:10 -t 00:07:22 -i output.mkv -acodec libvorbis -ab 128k -ac 2 -vcodec libx264 -preset slow -crf 22 -threads 0 our-final-product.mkv
Usually when you start screencasting, there will be a few moments of “getting ready” that you may want to cut out of your final product. Same thing near the end. It is possible to only encode a specific range of the original lossless file you captured using the -ss and -t options. These options can come before the input file option (i.e. before -i output.mkv) or after it. If they are specified after it, seeking will be more accurate but a lot slower. In the above example, we specified the starting point of the encoding of our final product after 10 seconds from the start of the original input file using -ss 00:00:10. We also specify the duration of the encoding to be 7 minutes and 22 seconds, because we want to effectively cut something at the end we don’t want to show. If there isn’t anything you want to hide at the end, you can just omit the -t option altogether. We use Vorbis and H.264 for the audio and video respectively and mux the entire thing to an .mkv container. This mix of Vorbis/H.264/Matroska is my favorite .

Example 4:

Code:
ffmpeg -i output.mkv -acodec libmp3lame -ab 128k -ac 2 -vcodec libxvid -qscale 8 -me_method full -mbd rd -flags +gmc+qpel+mv4 -trellis 1 -threads 0 our-final-product.avi
Here we have a typical avi with xvid and mp3. The options from “-me_method full” to “-trellis 1″ are encoding parameters for libxvid. You can see that we used libmp3lame to encode the audio with the same options we used for the other

examples. For video, tweaking the value of -qscale will give different results. Smaller value means higher video quality but increased file size and encoding time (Similar to libx264's -crf in the first example) . The output file is muxed into an .avi container.

Example 5:

Code:
ffmpeg -i output.mkv -acodec libvorbis -ab 128k -ac 2 -vcodec libtheora -b 1000k our-final-product.ogg
In the above example, we have a completely free file , with vorbis for audio and theora for video. The video is encoded at 1000k bitrate and the output file is muxed to an .ogg container. The audio quality should be great since vorbis is a great audio codec, but the video quality will not be as good. Theora lags behind in almost every aspect of video compression. See the WebM example above (example 2) for a better free and open alternative.

There are many tricks you can use with step 2. You can record your entire desktop at step 1 and then crop it in step 2 to only cover the area you worked on. You can also resize your screencast in step 2. Dealing with these situations and many others is beyond the scope of this tutorial.

FAQ:

Q: How do I get the exact size and coordinates of a specific window I want to capture?
A: Use a command called "xwininfo". Basically, you run this command and then click on the window that you want to capture. It will then print the window information to the terminal. This command prints a lot of information, but what you need are the following lines:

Absolute upper-left X:
Absolute upper-left Y:
Width:
Height:

If the command, for example, prints:

Absolute upper-left X: 383
Absolute upper-left Y: 184
Width: 665
Height: 486

Then, you will adapt it to FFmpeg like this:

-s 664x486 -i :0.0+383,184

Note that we used 664 instead of 665 for the width since ffmpeg only accepts resolutions divisible by 2.
You can use the following command line combination with "xwininfo" to only print the information you'll be needing:

Code:
xwininfo | grep -e Width -e Height -e Absolute

Q: How can I control PulseAudio input? (e.g. capture application audio instead of mic)
A: Install "pavucontrol". Start recording with ffmpeg. Start pavucontrol. Go to the "Recording" tab and you'll find ffmpeg listed there. Change audio capture from "Internal Audio Analog Stereo" to "Monitor of Internal Audio Analog Stereo".
Now it should record system and application audio instead of microphone.

This setting will be remembered. The next time you want to capture with FFmpeg, it will automatically start recording system audio. If you want to revert this, use pavucontrol again to change back to microphone input.

Q: What are the recommended codecs/container to use if I'm planning to upload my screencast to YouTube?
A: YouTube recommends uploading clips using H.264 for the video and AAC for the audio in an .mp4 container (as in example 1 of step 2). This is the safest format to upload into because some codecs do have issues with YouTube. Refer to this page for other recommended codecs and containers for YouTube.

Q: What do I do if my system does not use the PulseAudio sound server?
A: Most recent Linux distributions have PulseAudio installed by default, but if your system does not have PulseAudio, then try replacing "-f alsa -ac 2 -i pulse" with something like:

-f alsa -ac 2 -i hw:0,0

Many users of this guide reported success with the above options. You might have to change the 0,0 to match that of your sound device. You could also try:

-f alsa -ac 2 -i /dev/dsp

Other users reported success with:

-f oss -ac 2 -i /dev/dsp

Basically there are many ways to do it, and it depends on your system's sound configurations and hardware.

Q: How can I pause/resume screencasting?
A: That isn't possible with ffmpeg yet, but you can use mkvmerge to achieve the same result. You can install this program from your distribution's package management system under the package name mkvtoolnix, or download it from the official website. This program allows you to concatenate mkv files with identically-encoded streams to a single file. Meaning that if you want to make a pause from screencasting, you'll just stop recording part 1, then start recording part 2 as another file, and finally concatenate (i.e. add) these 2 parts into a single screencast ready for final compression. You can do this for as many parts as you like. Example:

Code:
mkvmerge -o complete.mkv part1.mkv +part2.mkv +part3.mkv +part4.mkv
This command pruduces a video file named "complete.mkv" that has all the parts put together in the order they were specified into on the command line.

However, note that all the parts you want to concatenate must have exactly the same size/framerate/encoding parameters, otherwise it won't work. You can't add files with different encoding options to each other.

Final note, before you start recording part 2 of your screencast, be sure to change the output file name, or else your previous precious work might be overwritten.

Q: I want to select an area of the screen with my mouse and get ffmpeg to record it. Is it possible?
A: Yes! There is a simple tool called "xrectsel" that you can use to select an area of the screen by drawing with your mouse, and it will print your selection coordinates which you can then use in ffmpeg. This tool comes as an auxiliary tool with the "FFcast2" bash script that wraps around ffmpeg's screencasting abilities

https://bbs.archlinux.org/viewtopic.php?id=127570

https://github.com/lolilolicon/FFcast2#readme
Q: How do I hide the mouse cursor?
A: Add "+nomouse" after ":0.0" to look like this:
Code:
:0.0+nomouse
(thanks again FakeOutdoorsman)

Q: I have a problem, error message, or ffmpeg doesn't work the way I expected it to do. How do I get help?

A: The best way to get help with ffmpeg is to login to the official IRC support channel for ffmpeg #ffmpeg @ irc.freenode.net, because 1) you will get an instant reply or engage in discussion about your problem and provide realtime feedback 2) the developers of ffmpeg know more about ffmpeg and its problems than we do (and they almost always hang around there and help people). If you want to ask for help here, you're most welcome. We'll try our best to support you, but bear in mind that the answer might come late, or we may not be able to help you with your problem.

Credits

verb3k
Dark_Shikari
FakeOutdoorsman

http://blog.devinrkennedy.com/2009/1...ng-ffmpeg.html

Updates:
Added FFcast2 reference link
changed syntax for new presets format on step1 (Thanks FakeOutdoorsman)
Removed all broken links, other junks

Revision #2
Created 2023-01-05 19:51:38 UTC by willi
Updated 2024-11-02 15:34:58 UTC by willi