Teaching Center Video Processing Automation

A tool created by Brenden Sweetman

**Introduction:**  
One of the responsibilities of Instructional Support Services is to provide the computers and personnel for processing videos of lectures around campus, recorded by the Teaching Center. For years we have been processing the videos using various applications including HandBrake and MP4joiner. These freeware applications have consistently plagued the process with crashes and general software bugs. In an effort to both streamline the process and avoid further annoyances with buggy software I have developed a simple script written in python that utilizes the open source unix-based video processing tool ffmpeg. The following technical documentation provides an in-depth look at how this script works. Please note this is the **technical** documentation for non-technical documentation please see the usage file.

**Python Script:**

The automation script is a relatively simple python3 file containing 3 methods:

**main:**

The main method is responsible for keeping track of all videos that need encoding. The method imports all of the files in the copied folder and removes the junk files. Next, it sorts the list of files. This is extremely important as this is how the script determines if there are multipart videos and the order of these parts. The method then checks for multipart videos by matching the first part of their file names. It then passes a list of all video parts to the errorHandler method. If the list clears the errorHandler then the main method sends the list of files to the cmdHandler method.

**errorHandler:**

The errorHandler method preforms limited error checking based on expected file names. This can only detect some errors. Just because the list of files passes this method does not mean it will work or the files will be in order.

**cmdHandler:**

The cmdHandler builds and executes the ffmpeg command (see the ffmpeg section).

**File Structure:**

The python script relies on a very specific set of files all files are contained in a root folder called “Teaching\_Center\_Video\_Processing.” When troubleshooting please insure all files are present and the folder hierarchy matches the one given below or the script will not run.

**Rules:**

* "Teaching\_Center\_Video\_Processing" must contain 3 further subdirectories: Copied, Final, and Resources
* The "run" bash script and "autoScript.py" must be stored at the base of "Teaching\_Center\_Video\_Processing"
* The "Copied" directory must contain all videos to be processed before the script is run
  + The "Copied" directory also must contain a "Done" directory to store the original files after processing
* The "Final" directory stores the completed videos ready for upload
* The "Resources" directory must contain a copy of the current copyright statement video clip: "copyright.mp4" and a copy of the ffmpeg executable.

**Complete File Structure:**

|Teaching\_Center\_Video\_Processing

|-->autoScript.py

|-->run

|-->|Copied

| |--><Files to be processed>

| |-->|Done

| |--><Finished originals>

|

|-->|Final

| |--><completed videos ready for upload>

|

|-->|Resources

| |-->ffmpeg

| |-->copyright.mp4

**Naming convention:**

This is perhaps the most important step before starting the script. The files MUST be named following this convention. This is the only way the script can recognize multi part files. If the name is off by even one character the script will not work, resulting in many different problems. Please double check all file names before running the script.

All movie video files have 5 mandatory parts delimited with "-":

1)Class

2)Professor

3)Year

4)Month

5)Day

Classes with 2 or more video files will have a 6th part denoting the order of the videos

The File naming convention is given below:

Class-Professor-Year-Month-Day-<part#>.mp4

NOTE: Do not use the “-” character for anything but separating the parts of the file name

Example file name for 1part video:

chem451-mabbs-2017-10-01.mp4

Example file name for multi-part video:

chem451-mabbs-2017-10-01-part1.mp4

chem451-mabbs-2017-10-01-part2.mp4

**ffmpeg:**

The ffmpeg tool is a command line tool for converting video, encoding video and concatenating video. This tool can perform in 1 bash command what used to take 2 programs and 5 of 6 steps. See <https://www.ffmpeg.org/> for more information about ffmpreg and its capabilities. The documentation for ffmpeg can be found here: <https://www.ffmpeg.org/ffmpeg.html>.

**ffmpeg command structure:**

The most basic command structure for ffmpeg is as follows:

$ ffmpeg –i <input file name> <output file name>

* + The –i argument dictates the input file
  + The output file is always the last argument passed and does not have a flag

Ffmpeg automatically determines the video file type based on the file suffix passed in the file names. For example:

$ ffmpeg –i exampleIn.avi exmapleOut.m4v

* + This command will convert the exmapleIn AVI file into a M4V file named exampleOut.

The command constructed by the python script is much more complicated. The script command includes additional options for bitrate and framerate. These options control for framerate compatibility and file size. The script also concatenates multipart videos into one output file.

**Options for bitrate and framerate:**

The framerate is controlled by the –r flag as seen below:

$ ffmpeg –i <input> -r 30 <output>

* The example above locks the output framerate to 30fps.

The bitrate is controlled by the –b:v and –bufsize tags as seen bellow:

$ ffmpeg –i <input> -r 30 –b:v 64k –bufsize 64k <output>

* The example above will lock the output framerate 30fps and output bitrate to 64 kbits/s.

NOTE: Command tags must come between the input file name and the output file name.

**Ffmpeg concatenation:**

The concatenation functionality of ffmpeg is by far the most complex part of the ffmpeg command created by the python script. The concat filter of ffmpeg is a complex filter and requires a couple tags to implement. The easiest way to explain is to go through an example. I will then go through each part of the example and its function.

$ ffmpeg –i input1.mp4 –i input2.avi –filter\_complex\

“[0:v:0] [0:a:0] [1:v:0] [1:a:0] concat=n=2:v=1:a=1”\

–map ”[v]” –map “[a]” output.mp4

* **-i**: Notice there is more than one input file. The files must be provided in order and each needs its own “-i” tag.
* **-filter\_complex**: This tag specifies the use of complex filters. The string following the tag is the arguments passed to the complex filter. Complex filters act on the audio and video input and output streams. Each input file provides a new input video and audio stream. These streams are numbered starting at 0 for the first video and counting up for each additional input video file. In our case we will have only one output file so there is only one output video and one output audio steam (0).
  + **[#:v:0]**: This argument specifies that we want the input video stream (denoted as #) to be sent to the only output video stream (0). We need one of these arguments for every input video file. The python script automaticity creates this depending on how many parts were given by the main method
  + **[#:a:0]:** This argument specifies the audio streams and works the same way as above.
  + **Concat=n=#:v=1:a=1** :This argument implements the concat filter
    - **n=#:** specifies the number of input streams (same as the number of video parts)
    - **v=1:** specifies the number of output video streams
    - **a=1:** specifies the number of output audio streams
* **-map “[v]”:** This argument tells the ffmpeg to map the output video stream from the concat filter to the output file
* **-map “[a]”:** This argument tells the ffmpeg to map the output audio stream from the concat filter to the output file

More information about the concat filter can be found at <https://trac.ffmpeg.org/wiki/Concatenate> under the **Concatenation of Files With Different Codecs** heading

NOTE: the python script uses one additional tag to set the verbosity of ffmpeg when running. The –loglevel tag sets how much info the ffmpeg daemon should share while encoding video. The script sets the log level to “error.” This will only print to the terminal if an error prevents ffpeg from running.

**What does our ffmpeg command look like?:**

Below is an example of the command created by the python script. This is the command for a 2-part video. Please go through each part and make sure you understand each part before trouble shooting the python script.

$ ffmpeg –loglevel error –i /Resources/copyright.mp4\

–i /Copied/chem111-sadtler-2017-10-20-part1.mp4\

–i /Copied/chem111-sadtler-2017-10-20-part2.mp4 \

–filter\_complex “ [0:v:0] [0:a:0] [1:v:0] [1:a:0] [2:v:0]\

[2:a:0] concat=n=3:v=1:a=1” –map “[v]” –map “[a]” –r 29.97\

–b:v 700k –bufsize 700k /Final/chem111-sadtler-2017-10-20.mp4

Understanding all parts of this command is imperative when trying to troubleshoot the python script. In the cmdHandler method the script builds this total command over several steps. Each step acts differently depending on the type of video passed.

**The “run” Bash Command File:**

Congratulations! If you have read this far there is only one more part to this tool: the executable to run it! It is a very simple bash script. The first line stores a variable with the relative path of the “Teaching\_Center\_Video\_Processing” root file. The script then echos the directory name for debugging purposes. Then it calls the script. If anything ever goes wrong with this file the script can still be run by opening the terminal, changing the directory to “Teaching\_Center\_Video\_Processing” and executing:

$ python3 autoScript.py.

Last Update by Brenden Sweetman 10/24/17