

---

---

# Variables and Loops in FFmpeg Scripts

— Variables, Array Variables,  
and For Loops —

---

---

# Agenda

- Variables / For Loops
- Piping
- Mapping



# Variables/For Loops

IT & Software > Other IT & Software > FFmpeg

## Deploy Variables and For Loops in Bash FFmpeg Scripts

Maximize your productivity for testing and production

5.0 ★★★★★ (3 ratings) 15 students

Created by [Jan Ozer](#)

🕒 Last updated 1/2023 🌐 English

Course URL: [https://bit.ly/var\\_4loops](https://bit.ly/var_4loops)

Click here for free: [https://bit.ly/variables\\_free](https://bit.ly/variables_free)

Coupon Code = SME\_2023

- Free for 30 days
- Download test files and scripts

# Lesson 1: Introduction

- Why variables
- What variables are
- About scripting languages
- Executing a Bash script

# Why Variables?

- FFmpeg command lines quickly become quite long and unwieldy.
- When variations of the same command line are needed (*for example to generate multiple rungs in a ladder, for multiple files*), it can quickly become error-prone to re-write them, or copy-paste-modify them
- A script makes it possible to write the command once, and quickly and easily modify just the relevant parameters within it, providing consistency and speed.

# What Variables Are

## Command Line (Before)

```
ffmpeg -y -i input.mp4 -c:v libx264 -vf scale=-1:1080 -b:v 3M -maxrate 6M -bufsize 6M -g 48 -preset ultrafast output.mp4
```

## Variables - After

```
height=1080  
bitrate="3M"  
maxrate="6M"  
bufsize="6M"  
gop=48  
preset="ultrafast"
```

```
ffmpeg -y -i input.mp4 -c:v libx264 -vf scale=-1:${height} -b:v ${bitrate} -maxrate  
${maxrate} -bufsize ${bufsize} -g ${gop} -preset ${preset} output.mp4
```

# Why Variables?

## Command Line (Before)

```
ffmpeg -y -re -i Freedom.mp4 \  
-y -c:v libx265 -g 60 -preset medium -tune psnr -b:v 6M -maxrate 6M -bufsize 12M  
Freedom/Freedom_x265_medium_6M_PSNR.mp4 \  
-y -c:v libx265 -g 60 -preset medium -tune psnr -b:v 5M -maxrate 5M -bufsize 10M  
Freedom/Freedom_x265_medium_5M_PSNR.mp4 \  
-y -c:v libx265 -g 60 -preset medium -tune psnr -b:v 4M -maxrate 4M -bufsize 8M  
Freedom/Freedom_x265_medium_4M_PSNR.mp4 \  
-y -c:v libx265 -g 60 -preset medium -tune psnr -b:v 3M -maxrate 3M -bufsize 6M  
Freedom/Freedom_x265_medium_3M_PSNR.mp4
```

# Why Variables?

## Variables After

```
#variables
preset=medium
tune=psnr
GOP=60
bitrate1=6.0M
bufsize1=12.0M
bitrate2=5.0M
bufsize2=10.0M
bitrate3=4.0M
bufsize3=8.0M
bitrate4=3.0M
bufsize4=6.0M

input="Freedom.mp4"
base=$(basename ${input%.*})
outputfolder="./outputs/$base"
mkdir -p $outputfolder

ffmpeg -re -i ${input} \
-c:v libx265 -g ${GOP} -preset ${preset} -tune ${tune} -threads 8 -bf 3 -b:v
${bitrate1} -maxrate ${bitrate1} -bufsize ${bufsize1}
${outputfolder}/${base}_x265_${preset}_${bitrate1}_${tune}.mp4
-c:v libx265 -g ${GOP} -preset ${preset} -tune ${tune} -threads 8 -bf 3 -b:v
${bitrate2} -maxrate ${bitrate2} -bufsize ${bufsize2}
${outputfolder}/${base}_x265_${preset}_${bitrate2}_${tune}.mp4
-c:v libx265 -g ${GOP} -preset ${preset} -tune ${tune} -threads 8 -bf 3 -b:v
${bitrate3} -maxrate ${bitrate3} -bufsize ${bufsize3}
${outputfolder}/${base}_x265_${preset}_${bitrate3}_${tune}.mp4
-c:v libx265 -g ${GOP} -preset ${preset} -tune ${tune} -threads 8 -bf 3 -b:v
${bitrate4} -maxrate ${bitrate4} -bufsize ${bufsize4}
${outputfolder}/${base}_x265_${preset}_${bitrate4}_${tune}.mp4
```

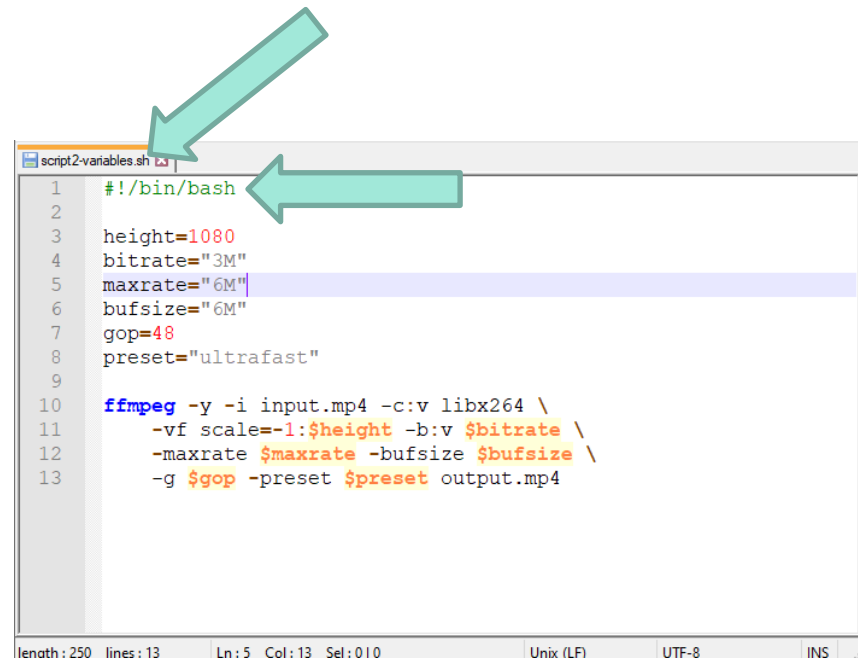
# Scripting Languages

- There are many scripting languages
- This lesson covers Bash scripting for Linux & MacOS
  - All scripts tested on an AWS Ubuntu instance
- **Windows:**
  - **Cannot use normal command window**
  - **Must use PowerShell (not covered in this session)**

⚠ Disclaimer: We wrote all examples to be as easy to read as possible. Bash often has more advanced mechanisms that may be more efficient/much less verbose, but also more challenging to understand. For this course, we assumed that simpler is better

# What is a Shell script?

- A text file
- Typically topped with the Shebang (`#!/bin/bash`)
  - Tells the kernel which interpreter to use to run the commands in the file
- With a `.sh` extension



```
1  #!/bin/bash
2
3  height=1080
4  bitrate="3M"
5  maxrate="6M"
6  bufsize="6M"
7  gop=48
8  preset="ultrafast"
9
10 ffmpeg -y -i input.mp4 -c:v libx264 \
11     -vf scale=-1:$height -b:v $bitrate \
12     -maxrate $maxrate -bufsize $bufsize \
13     -g $gop -preset $preset output.mp4
```

The screenshot shows a text editor window titled 'script2-variables.sh'. The script content is as follows:

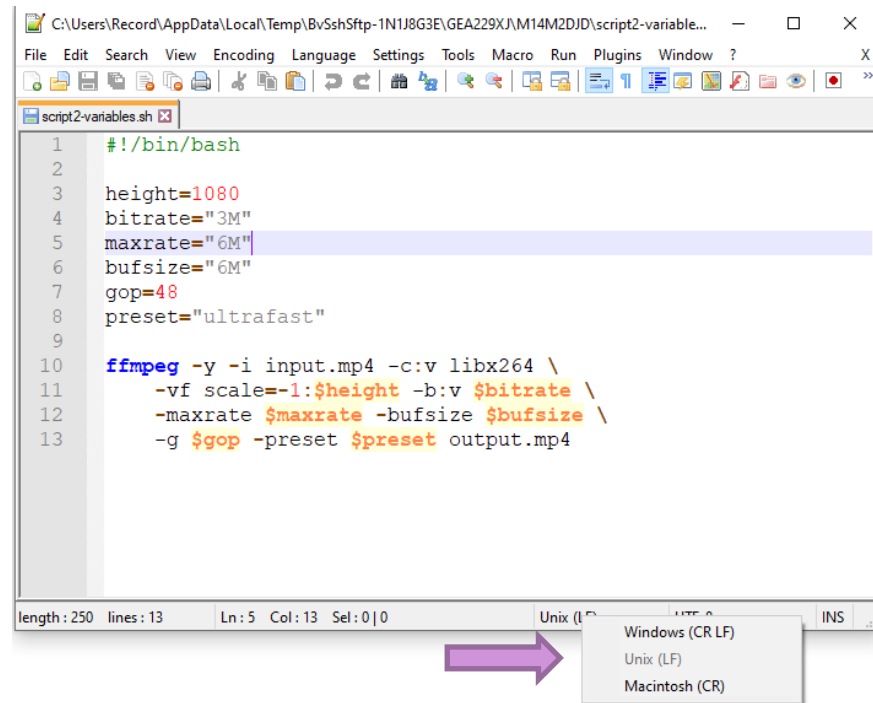
```
1  #!/bin/bash
2
3  height=1080
4  bitrate="3M"
5  maxrate="6M"
6  bufsize="6M"
7  gop=48
8  preset="ultrafast"
9
10 ffmpeg -y -i input.mp4 -c:v libx264 \
11     -vf scale=-1:$height -b:v $bitrate \
12     -maxrate $maxrate -bufsize $bufsize \
13     -g $gop -preset $preset output.mp4
```

At the bottom of the window, a status bar displays: 'length: 250 lines: 13 Ln: 5 Col: 13 Sel: 010 Unix (LF) UTF-8 INS'. Two green arrows are overlaid on the image: one points to the shebang line (line 1) and the other points to the first variable assignment 'height=1080' (line 3).



# Writing Scripts

- Can write scripts in any text editor
- Script editor like Notepad ++ does some advanced formatting and error checking
- If you're creating scripts on one OS (Windows) and executing on Linux, be sure to save script in Unix format
  - Otherwise, hidden line continuation characters can cause funky errors



```
1  #!/bin/bash
2
3  height=1080
4  bitrate="3M"
5  maxrate="6M"
6  bufsize="6M"
7  gop=48
8  preset="ultrafast"
9
10 ffmpeg -y -i input.mp4 -c:v libx264 \
11     -vf scale=-1:$height -b:v $bitrate \
12     -maxrate $maxrate -bufsize $bufsize \
13     -g $gop -preset $preset output.mp4
```

length : 250 lines : 13 Ln : 5 Col : 13 Sel : 0 | 0 Unix (LF) Windows (CR LF) Unix (LF) Macintosh (CR)

# Target script

We will build a script that will:

- Create a ladder of H.264 MP4 rungs at different resolutions and bitrates
- Make it simple to define a GOP size and x264 preset
- Build such a ladder for more than one input file
- Name the output files automatically, on the basis of the rung spec and input file name

The main FFmpeg command that we will use:

```
ffmpeg -y -i input.mp4 -c:v libx264 -vf scale=-1:1080 -b:v 3M -maxrate 6M -bufsize 6M -g 48 -preset ultrafast output.mp4
```

# Executing a simple script

- Add that simple FFmpeg command line in a file called **script1-minimal.sh** (in the same folders as input.mp4)
- Before you run a Bash script, you must authorize its execution by typing the following line in the terminal

```
chmod +x script1-minimal.sh
```

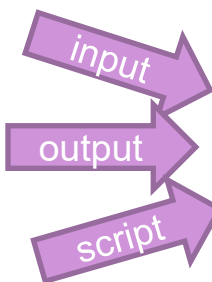
- Then, run the script




```
./script1-minimal.ps1
```

# script1-minimal.sh

```
ffmpeg -y -i input.mp4 -c:v libx264 -vf scale=-1:1080 -b:v 3M -maxrate 6M -  
bufsize 6M -g 48 -preset ultrafast output_command.mp4
```

```
ubuntu@ip-172-31-1-146:~/testfiles/Lesson_1$ ls  
input.mp4  output_command.mp4  script1-minimal.sh  
ubuntu@ip-172-31-1-146:~/testfiles/Lesson_1$ chmod +x script1-minimal.sh  
ubuntu@ip-172-31-1-146:~/testfiles/Lesson_1$ ./script1-minimal.sh
```



Remote files					Filter:
/home/ubuntu/testfiles/Lesson_1					
Name	Size	Type	Date Modified	Permissions	
 input.mp4	109,120,392	MP4 Video F...	8/9/2022 9:10 PM	-rw-rw-r--	
 output_command.mp4	3,984,902	MP4 Video F...	1/2/2023 11:26 AM	-rw-rw-r--	
 script1-minimal.sh	129	SH File	1/2/2023 11:14 AM	-rwxrwxr-x	



## Lesson 2: Creating and Using Variables

- Variables allow information to be contained and referenced later in a script
- And to label data with a descriptive name

For example, let's set the codec parameters through variables:

```
ffmpeg -y -i input.mp4 -c:v libx264  
-vf scale=-1:1080 -b:v 3M -maxrate 6M -bufsize 6M -g 48 -preset ultrafast output.mp4
```

height

bitrate

maxrate

bufsize

gop

preset

# Using Variables

script2-variables.sh

height=1080

bitrate="3M"

maxrate="6M"

bufsize="6M"

gop=48

preset="ultrafast"

```
ffmpeg -y -i input.mp4 -c:v h264 \
-vf scale=-1:${height} -b:v ${bitrate} \
-maxrate ${maxrate} -bufsize ${bufsize} \
-g ${gop} -preset ${preset} output.mp4
```

- Define a variable by an alphanumeric name
- Assign a value to the variable with the equal sign (=) **without spaces**
- If the value is a string, surround by quotes ("")
- Reference the variable with the variable name in braces, preceded by a dollar sign
- **Note:** in Bash a command can be written over multiple lines of a script by ending each line (but the last one) with a backslash (\)

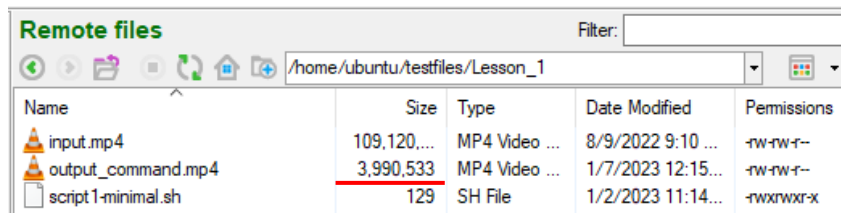




# script2-variables.sh

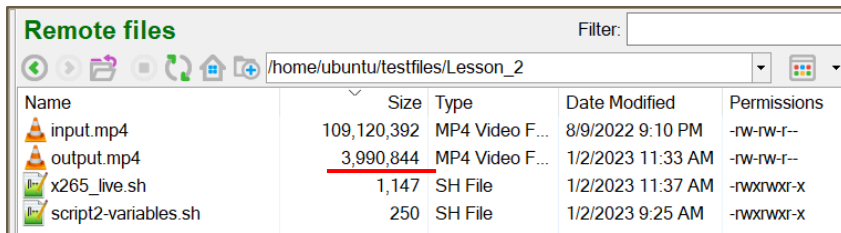
## From Original Command Line

- Run
- Verify files
  - FFmpeg doesn't return the same file each time
  - Some light variations
  - Very, very close



Name	Size	Type	Date Modified	Permissions
input.mp4	109,120,...	MP4 Video ...	8/9/2022 9:10 ...	-rw-rw-r--
output_command.mp4	<u>3,990,533</u>	MP4 Video ...	1/7/2023 12:15...	-rw-rw-r--
script1-minimal.sh	129	SH File	1/2/2023 11:14...	-rwxrwxr-x

## From Variables



Name	Size	Type	Date Modified	Permissions
input.mp4	109,120,392	MP4 Video F...	8/9/2022 9:10 PM	-rw-rw-r--
output.mp4	<u>3,990,844</u>	MP4 Video F...	1/2/2023 11:33 AM	-rw-rw-r--
x265_live.sh	1,147	SH File	1/2/2023 11:37 AM	-rwxrwxr-x
script2-variables.sh	250	SH File	1/2/2023 9:25 AM	-rwxrwxr-x

# Lesson 3: Checking and Debugging Your Strings

Let's separate the building of the command line from its execution

This allows us to:

- Print it on screen (so we can validate it, or troubleshoot the script)
- Add it to a log file (again, for validation and troubleshooting later on)

# Deferred Execution

script3-deferred.sh

```
gop=48
preset="ultrafast"
height=1080
bitrate="3M"
maxrate="6M"
bufsize="6M"
```

```
cmd="
  ffmpeg -y -i input.mp4 -c:v libx264
    -vf scale=-1:${height} -b:v ${bitrate}
    -maxrate ${maxrate} -bufsize ${bufsize}
    -g ${gop} -preset ${preset} output.mp4
"
```

```
echo $cmd
```

```
$( $cmd )
```

- Build the command as a string, surrounded by quotes (").
- It can span multiple lines without special characters needed
- As before, variable values are inserted in the string by referencing variable names surrounded by `${}`
- `echo` prints it on screen
- Execute command by surrounding it in `$()`
- Let's run it



# With and Without Deferred Execution

## Script2-variables.sh

```
ubuntu@ip-172-31-1-146:~/testfiles/Lesson_2$ ./script2-variables.sh
ffmpeg version 4.4.2-0ubuntu0.22.04.1 Copyright (c) 2000-2021 the FFmpeg developers
  built with gcc 11 (Ubuntu 11.2.0-19ubuntu1)
  configuration: --prefix=/usr --extra-version=0ubuntu0.22.04.1 --toolchain=hardened --libdir=/usr/lib/aarch64-linux-gnu --incdir=/usr/include/aarch64-linux-gnu --arch=arm64 --enable-gpl --disable-str
```

## Script3-deferred.sh

```
ubuntu@ip-172-31-1-146:~/testfiles/Lesson_3$ ./script3-deferred.sh
ffmpeg -y -i input.mp4 -c:v libx264 -vf scale=-1:1080 -b:v 3M -maxrate 6M -bufsize 6M -g 48 -preset ultrafast output.mp4
ffmpeg version 4.4.2-0ubuntu0.22.04.1 Copyright (c) 2000-2021 the FFmpeg developers
  built with gcc 11 (Ubuntu 11.2.0-19ubuntu1)
  configuration: --prefix=/usr --extra-version=0ubuntu0.22.04.1 --toolchain=hardened --libdir=/usr/lib/aarch64-linux-gnu --incdir=/usr/include/aarch64-linux-gnu --arch=arm64 --enable-gpl --disable-str
```



Sent to FFmpeg

# Logging to File

script3b-logging.sh

```
gop=48
preset="ultrafast"
height=1080
bitrate="3M"
maxrate="6M"
bufsize="6M"

cmd="
  ffmpeg -y -i input.mp4 -c:v libx264
    -vf scale=-1:${height} -b:v ${bitrate}
    -maxrate ${maxrate} -bufsize ${bufsize}
    -g ${gop} -preset ${preset} output.mp4
"

echo $cmd
echo $cmd >> log.txt

$($cmd)
```

- When followed by redirection operators (**>**), **echo** writes the value of the variable to a file, instead of to screen
- By doubling the redirection operator (**>>**), the string is appended to the file. Otherwise a new file will be written, or overwritten if it already exists
- Let's run it

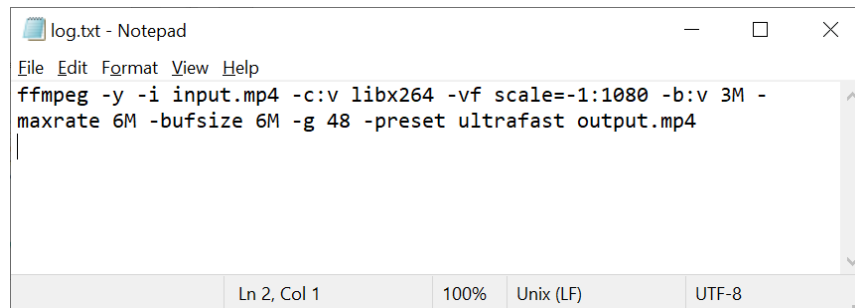
# Here's the Log File

```
gop=48
preset="ultrafast"
height=1080
bitrate="3M"
maxrate="6M"
bufsize="6M"

cmd="
ffmpeg -y -i input.mp4 -c:v libx264
    -vf scale=-1:${height} -b:v ${bitrate}
    -maxrate ${maxrate} -bufsize ${bufsize}
    -g ${gop} -preset ${preset} output.mp4
"

echo $cmd
echo $cmd >> log.txt

$($cmd)
```



The screenshot shows a Notepad window with the following content:

```
log.txt - Notepad
File Edit Format View Help
ffmpeg -y -i input.mp4 -c:v libx264 -vf scale=-1:1080 -b:v 3M -
maxrate 6M -bufsize 6M -g 48 -preset ultrafast output.mp4
```

The status bar at the bottom indicates: Ln 2, Col 1 | 100% | Unix (LF) | UTF-8

- Shows the command sent to FFmpeg
  - not the original script
- Invaluable debugging tool
  - If script isn't working, check the log file
- I include in all my scripts

# Lesson 4: File and Filename Manipulation

- Previous lessons
  - Encode single file from current folder
  - Stored encoded files in current folder
- In this lesson, you'll learn to
  - Retrieve files from different locations
  - Store files in different locations
    - Dynamically name and create folders based upon input file



# Filename Manipulation

script4-filesnames.sh

```
# (... variables as before ...)
```

```
# folders and filenames
```

```
input="./sources/input.mp4"
```

```
base=$(basename "$input%.*")
```

```
outputfolder="./outputs/$base"
```

```
mkdir -p $outputfolder
```

```
output="$outputfolder/${base}_${height}p_  
${bitrate}_${preset}.mp4"
```

```
cmd="
```

```
ffmpeg -y -i ${input} -c:v libx264  
-vf scale=-1:${height} -b:v ${bitrate}  
-maxrate ${maxrate} -bufsize ${bufsize}  
-g ${gop} -preset ${preset} ${output}
```

```
# (...)
```

- We create variable **input** for the input file
- We introduce a new variable – **base** – to name output files
  - **basename** returns the filename (without path)
  - **\${input%.\*}** removes any file extension
  - **\$()** wraps the expression to execute it and store its output in the variable.
- Create variable **Outputfolder** –to store file(s)
- **mkdir -p** creates any named folder and subfolder that don't yet exist
- We build the **output** variable from multiple variables
- Finally, the new variables are used in the command line



Note: lines starting with **#** denote comments. It's a good idea to comment your script to explain (to your future you) what is happening






# script4-filenames.sh

- Files in **/sources** (need folder and input.mp4)
- Script creates **outputs** folders
- Script creates **input** folder (from file name)
- Script produces the file


## Start – Need Sources folder

Name	Size	Type	Date Modified	Permis
 sources	4,096	File folder	1/2/2023 2:24 PM	drwxrw
 script4-filenames.sh	489	SH File	1/2/2023 2:25 PM	-rwxrwx


## Script Creates Output Folder

Name	Size	Type	Date Modified	Permis
 outputs	4,096	File folder	1/2/2023 3:00 PM	drwxrw
 sources	4,096	File folder	1/2/2023 2:24 PM	drwxrw
 script4-filenames.sh	489	SH File	1/2/2023 2:25 PM	-rwxrwx

## Script Creates Base Folder (from file name)

Name	Size	Type	Date Modified	Permis
 input	4,096	File folder	1/2/2023 3:00 PM	drwxrw

## Script Creates and Output Files

Name	Size	Type	Date Modified	Permis
 input_1080p_3M_ultrafast.mp4	3,987,6...	MP4 Video F...	1/2/2023 3:00 PM	-rw-rw-r

# Lesson 5: Encoding Multiple Input Files

- For loops
  - Iteratively encode selected files
- Two ways to select files
  - Array variables - list files directly
  - File lists - point to folders and conditions and OS chooses files

# For Loops


- This script encodes all MP4 files in a folder and stores them in the output folder
  - `Mkdir -p output` – creates the output folder (-p creates parent and doesn't stop if folder already exists)
  - `for ... in ...` iterates through all the items one by one and assigns each to the variable `file` in turn
  - `For file in *.mp4` perform the for loop on all MP4 files in that folder
  - The section of code between `do` and `done` is the set of commands performed in each selected file
  - Output to output folder (otherwise would start cycle again on encoded MP4 files)

## script5-forloop.sh

```
# (...)
mkdir -p output
for file in *.mp4
do
    ffmpeg -y -i $file -c:v libx264 \
    -vf scale=-1:${height} -b:v ${bitrate} \
    -maxrate ${maxrate} -bufsize ${bufsize} \
    -g ${gop} -preset ${preset} \
    output/"${file%.*}_${bitrate}_${preset}.mp4"
done
```

# For Loops

- Script must be in folder with MP4 files
- Will create output folder
- Let's run it
- Quick and dirty way to illustrate **for loops**;  
use this for simple production tasks (like  
encoding multiple files to CRF)
- So, **loop runs** on all selected files
  - Can select files with array variables
  - Or via file lists (like we did here, but more useable)



Name	Size	Type
output	4,096	File folder
BBB.mp4	54,059,529	MP4 Video ...
script5forloop.sh	370	SH File
sintel.mp4	68,710,940	MP4 Video ...
TOS.mp4	47,984,783	MP4 Video ...



# Array Variables

- An array variable is explicitly defined as a list of items, surrounded by parenthesis `()` and separated by spaces
- `for ... in ...` iterates through all the items in that array variable one by one and runs it through the script
- `${files[@]}` returns all elements in the variable as a list and loop through that list
- The section of code between `do` and `done` is the set of commands performed in each iteration
- Review script5-array.sh

```
script5-array.sh

# (...)

# list of source files
inputfolder=./sources
files=("BBB.mp4" "sintel.mp4" "TOS.mp4")

for file in ${files[@]}
do
    input="./sources/${file}"

    # (... code as before to define output filename
    #   and execute the ffmpeg command ...)

done
```

Listed files must be in /sources folder





# Listing the files

- Instead of explicitly listing all assets within the script, we can ask the operating system to create a list
- We can then work directly from that list, instead storing it into an array variable (but we could also do both)

# File list

- `ls` lists all the files in the input folder with an “mp4” file extension
- Surrounding it with `$( )` makes Bash execute that command and store its output into an array
- We can then use the item-based `for` loop as before to iterate through that collection
- Lets run it
  - Show sources
  - Delete output folder

script5-filelist.sh

```
# (...)

# list of source files
inputfolder=./sources
files=$(ls $inputfolder/*.mp4)

for input in ${files[@]}
do

    # (... code as before to define output filename
    # and execute the FFmpeg command ...)

done
```



# Generating a Ladder

- Let's modify the script to generate a full ladder for each input asset
- Configure each rung with a different resolution, bitrate, maxrate and bufsize
- Create multiple array variables, one for each encoding parameter
- Use an index-based loop to select the parameters for each rung
  - Create two **for loops**:
    - One for the rungs
    - One (as before) for the input files

# Lesson 6: Index-based loop

- Create 4 array variables, each with 3 items
- `${!heights[@]}` returns the list of indices in the array
- `for ... in ...` iterates through those indices and assigns each one in turn to the variable `rung`.
- In each iteration, we extract the item from an array by its position. This is done by the expression `${heights[$rung]}` in which the variable between the square brackets is the index of the item.
- This new loop is inside the original loop

## script6-indices.sh

```
# array variables define the rungs
heights=(360 540 1080)
bitrates=(500k 1M 2M)
maxrates=(1M 2M 4M)
bufsizes=(1M 2M 4M)

# (... listing files as before ...)

for input in ${files[@]}
do
    for rung in ${!heights[@]}
    do
        height=${heights[$rung]}
        bitrate=${bitrates[$rung]}
        maxrate=${maxrates[$rung]}
        bufsize=${bufsizes[$rung]}

        output="${outputfolder}/${base}_${height}p_
                ${bitrate}_${preset}.mp4"

        cmd="
            ffmpeg -y -i ${input} -c:v libx264
                -vf scale=-1:${height} -b:v ${bitrate}
                -maxrate ${maxrate} -bufsize ${bufsize}
                -g ${gop} -preset ${preset} ${output}
            "
```

# Lesson 6: Index-based loop

- Note key details
  - Array variables input into dynamic variables
  - Dynamic variables input into command
- Minor details can hose operation
  - With attention, can adopt script to any codec or set of encoding parameters
- Review entire script
- Run script

## script6-indices.sh

```
# array variables define the rungs
heights=(360 540 1080)
bitrates=(500k 1M 2M)
maxrates=(1M 2M 4M)
bufsizes=(1M 2M 4M)

# (... listing files as before ...)

for input in ${files[@]}
do

    for rung in ${!heights[@]}
    do
        height=${heights[$rung]}
        bitrate=${bitrates[$rung]}
        maxrate=${maxrates[$rung]}
        bufsize=${bufsizes[$rung]}

        output="${outputfolder}/${base}_${height}p_
                ${bitrate}_${preset}.mp4"

        cmd="
            ffmpeg -y -i ${input} -c:v libx264
                -vf scale=-1:${height} -b:v ${bitrate}
                -maxrate ${maxrate} -bufsize ${bufsize}
                -g ${gop} -preset ${preset} ${output}
            "
    done
done
```





# Piping


- What is it: using FFmpeg to create a file to pass to another program without storing the file
  - Useful when creating intermediate files would consume too much disk space or would be too time consuming
- Example: x265 encoder needs raw file for input
  - Convert source to raw in FFmpeg
  - Pass to x265 for encoding

# Piping: Raw File to x265 Encoder

```
ffmpeg -y -i Football_short.mp4 -an -f rawvideo file.yuv
```

```
x265_main.exe --input file.yuv --input-res 1920x1080 --preset 5 --fps 29.97 --frame-threads 1 --no-wpp --pools 1 -o football_short.hevc
```

```
ffmpeg -y -i Football_short.mp4 -an -f rawvideo - | x265_main.exe --input - --input-res 1920x1080 --preset 5 --fps 29.97 --frame-threads 1 --no-wpp --pools 1 -o football_short_2.hevc
```



- Procedure:
  - Get command strings working separately
  - Then combine, substituting - (dash) for output file name and input file name and inserting pipe symbol

# Piping: Decode HEVC File to Raw for MSU VQMT

```
ffmpeg -y -i football_short_2.hevc -an -f rawvideo football_short.yuv
```

```
"C:\Program Files\MSU VQMT 14.0 beta\msu_metric_14.1.exe" -in "Football_short.mp4" -in  
football_short.yuv 1920x1080 YUV420p -csv -metr psnr over Y -resize lanczos to orig
```

```
ffmpeg -y -i football_short_2.hevc -an -f yuv4mpegpipe -pix_fmt yuv444p pipe: | "C:\Program  
Files\MSU VQMT 14.0 beta\msu_metric_14.1.exe" -in Football_short.mp4 -in pipe: 1920x1080  
yuv444p -csv -metr psnr over Y -resize lanczos to orig
```

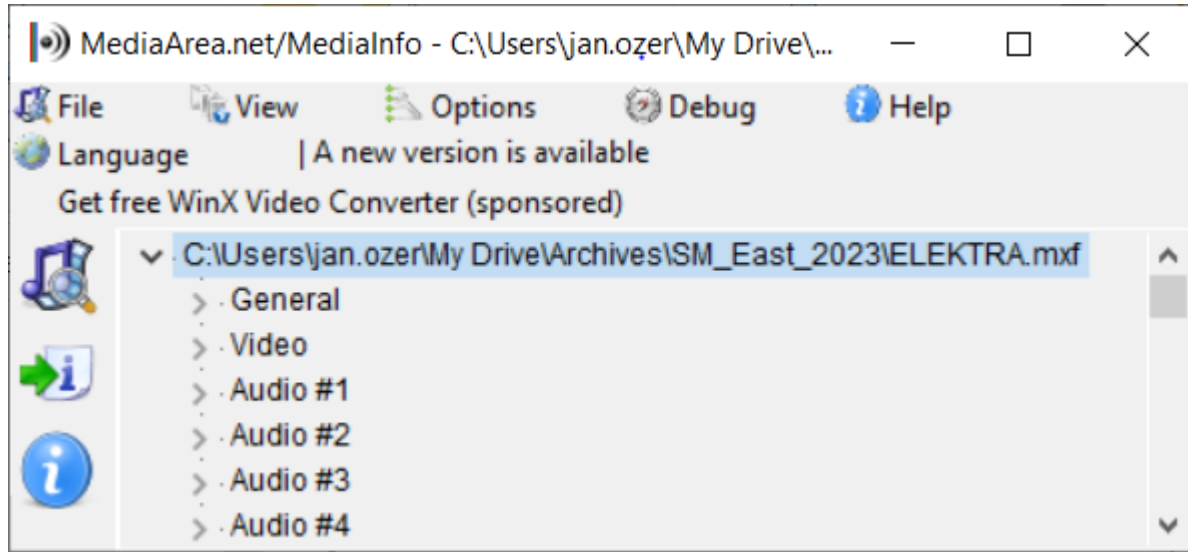
- Procedure:

- Worked fine separately
- Needed special language to make it work
  - Had to use designated format from MSU docs
  - Had to use pipe: rather than -

# Piping

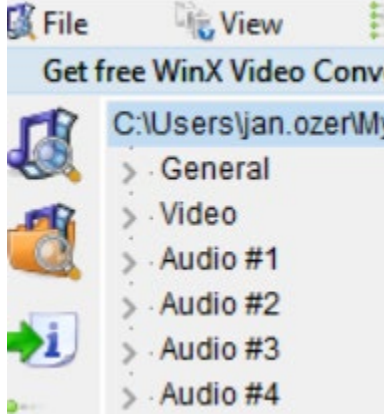
- Each case feels unique
- May need some custom tweaking to make it work
- Few good resources to learn (trial and error)

# Mapping



- Selecting streams within a file to customize an output
  - Can be native streams within a file (audio/video streams)
  - Named streams within an FFmpeg script
- Elektra
  - One video file
  - Audio 1/2 are left/right Spanish
  - Audio 3/4 are left/right English

# Mapping



Can map in  
absolute position

-map 0:0  
-map 0:1  
-map 0:2  
-map 0:3  
-map 0:4

File

Stream

Add video/audio  
designator:

-map 0:v:0  
-map 0:a:0  
-map 0:a:1  
-map 0:a:2  
-map 0:a:3

File

Stream

In both cases, 0 is the first stream

# Creating Separate Language files

## Absolute

```
ffmpeg -y -ss 00:42:55 -i elektra.mxf -map 0:0 -map 0:1 -map 0:2 -t 00:00:40 -c copy  
elektra_Spanish.mxf
```

```
ffmpeg -y -ss 00:42:55 -i elektra.mxf -map 0:0 -map 0:3 -map 0:4 -t 00:00:40 -c copy  
elektra_English.mxf
```

## A/V Designator

```
ffmpeg -y -ss 00:42:55 -i elektra.mxf -map 0:v:0 -map 0:a:0 -map 0:a:1 -t 00:00:40 -c  
copy elektra_Spanish_1.mxf
```

```
ffmpeg -y -ss 00:42:55 -i elektra.mxf -map 0:v:0 -map 0:a:2 -map 0:a:3 -t 00:00:40 -c  
copy elektra_English_1.mxf
```

# Scaling: Mapping to Command String Designators

```
ffmpeg -y -i elektra_English_1.mxf -y ^  
-filter_complex  
"[0:v]split=3[out1080p][out1080p2][in1080p];[in1080p]scale=1280:720:flags=fast_bilin  
ear,split=2[out720p][in720p];[in720p]scale=640:360:flags=fast_bilinear[out360p]" ^  
-map [out1080p] -c:v libx264 -b:v 3.5M -maxrate 3.5M -bufsize 7M -preset ultrafast  
Elektra_1080p_3_5M.mp4 ^  
-map [out1080p2] -c:v libx264 -b:v 1.8M -maxrate 1.8M -bufsize 3.6M -preset ultrafast  
Elektra_1080p_1_8M.mp4 ^  
-map [out720p] -c:v libx264 -b:v 1M -maxrate 1M -bufsize 2M -preset ultrafast  
Elektra_720p_1M.mp4 ^  
-map [out360p] -c:v libx264 -b:v .5M -maxrate .5M -bufsize 1M -preset ultrafast  
Elektra_360p_500K.mp4
```



# Mapping Resources

- FFmpeg Wiki - [bit.ly/FF\\_Wiki\\_Map](https://bit.ly/FF_Wiki_Map)
- FFmpeg Documentation - [bit.ly/FF\\_docs\\_map](https://bit.ly/FF_docs_map)
- Write blog - [bit.ly/Write\\_map](https://bit.ly/Write_map)