Session 5 - Exercise and Refactorize (with Style)

The Plan

- Arrays
- Output formatting
- Hashcode validation.
- Formatting Code. Readability.
- Refactoring Code
- FFmpeg:
  - Seek and Trim
  - Segmenting
  - Concatenating
  - Logo overlay

Arrays

> "An array is a systematic arrangement of similar objects, usually in rows and columns."

See: Array (Wikipedia)

Arrays

```
STUFF=(apple banana orange)

echo "${STUFF[0]}"      # First item in array
echo "${STUFF[*]}"      # All items
echo "${STUFF[@]}"      # All items (
echo "${#STUFF[*]}"     # Number of items in array

# With spaced items, '@' and "" is required.
for ITEM in "${STUFF[@]}"; do
    echo "Current item: $ITEM"
done
```

See: Bash Arrays (Linuxjournal)

Formatting output: printf

> "printf" is used to format strings, according to placeholder masks.

See: Printf format string (Wikipedia)

printf: Basics

```
$ printf "MASK" arg1 arg2 ...
```

- Text = %s
- Decimal = %d
- Float = %.2f (precision 2)
- 0-padded = %05d (5 digits)
- Newline = \n

printf - Strings and numbers

```
$ printf "abc-%s_%d.xxx" "reel_1" 7
```

abc-reel_1_7.xxx

printf - Floating point

```
$ printf "%.2f" 2,12345
```

2,12

printf - Zero padding

```
$ printf "abc-%05d.xxx" 42
```

abc-00042.xxx

Exercise:

DPX - Renumbering

Take a folder of DPX files and rename them to filenames with continuous numbering, according to a given naming rule and a start number.

DPX - Renumbering

```
MASK="reelX_%07d.dpx"

i=1
for f in *.dpx; do
    OUT=$(printf "$MASK" $i)
    echo "$f : $OUT"
    mv $f $OUT

    i=$((i+1))
done
```

Exercise:

Copy files with hashcode

Take input folder as argument and copy DPX files in there into a another folder, but verify that all files were copied bit-proof.

Steps?

- Generate hash for source
- Copy
- Generate hash for target
- Compare source/target hash

Transcode DPX to Video

```
$FFMPEG -f image2 -i NAME_%07d.dpx \
  -framerate 24 \
  -c:v ffv1 -an \
  $VIDEO_OUT
```

See: FFmpeg 'image2' docs

Transcode DPX to Video

Example

```
$FFMPEG -f image2 -framerate 24 -start_number 90137 \
  -i dpx/IA900656_IAETV005215_%07d.dpx \
  -c:v ffv1 -an \
  out.mkv
```

Streamhash (DPX)

```
$FFMPEG -f image2 -framerate 24 -start_number 90137 \
  -i dpx/IA900656_IAETV005215_%07d.dpx \
  -f streamhash -v quiet -hash md5 -
```

0,v,MD5=97de866eb4f492b2c08d222b99fcf1bb

Streamhash (Video)

```
$FFMPEG -i out.mkv
  -f streamhash -v quiet -hash md5 -
```

0,v,MD5=97de866eb4f492b2c08d222b99fcf1bb

Formatting Code / Readability.

- Avoid long lines:
  Aim at 80 characters per line.
- Use space instead of tab.
- Keep conditional blocks short.
- Avoid repetition.
- Avoid hardcoding values.
- Document & comment.

Document and Comment

```
###
# This is an amazing program that does incredible stuff!
# Such as:
#   * this!
#   * that.
#   * etc.
#
# @author: Peter B. (pb@das-werkstatt.com)
# @date: 2020-03-01
```

It's very good practice to add some form of header to your code.

Document and Comment

```
##
# This function wraps the atmosphere of a planet in paper
# and paints it in the given color.
# Returns true if successful.
#
# @param string Planet
# @param integer Color
#
function ColorPlanet {
  ...
}
```

Also for each function.

This documentation example is actually a bit "beyond bash", but it doesn't hurt to maybe already get used to proper in-code function documentation.

If code is properly documented, tools like Doxygen for example, are able to automatically generate code documenation in different output formats (HTML, PDF, etc).

As example, see this page from Carnegie Mellon's Computer Science Department: Coding Style and Doxygen Documentation, and the generated output.

Multi line commands

- Line ending in script = execute line.
- Escaping the EOL character = continue line.
- Escape character: '\' (Backslash)

Readability?

```
$FFMPEG -y -i $VIDEO_IN -i $LOGO -filter_complex "overlay=main_
w-overlay_w-25:25" -c:v libx264 -crf 23.0 -pix_fmt yuv420p -p
reset slow -filter:v yadif=0:-1:0,scale=768:576 -aspect 4:3 -an
$VIDEO_OUT
```

Better.

```
$FFMPEG -y -i $VIDEO_IN -i $LOGO \
  -filter_complex "overlay=main_w-overlay_w-25:25" \
  -vcodec libx264 -crf 23.0 -pix_fmt yuv420p -preset slow \
```

```
-filter:v yadif=0:-1:0,scale=768:576 -aspect 4:3 \
-an \
$VIDEO_OUT
```

Short conditional blocks

- Make it easier to follow the context/flow.
- Reduces risk of Spaghetti Code
- Are easier to maintain.

Avoid repetition

- If value is used more than once: Set variable.
- If similar code repeats: Convert to function.

Avoid hardcoded values

- (How often) will it change?
- Would it be useful to be a parameter?
- Is it better to read as value or named variable?

Avoid hardcoded values

```
exit 4
```

```
ERROR_FileNotFound=4
exit $ERROR_FileNotFound
```

Sometimes it's okay:

```
$FFMPEG -f image2 -i $DPX_IN -framerate $FPS ...
```

```
$FFMPEG -f image2 -i $DPX_IN -framerate 24 ...
```

But choose wisely.

Refactorize!

Let's apply what we've heard and seen so far, to existing code.

More FFmpeg

Seek and Trim

- Seeking: `-ss TIME`
- Duration: `-t TIME`

Seek and Trim

```
$FFMPEG -ss 00:01:23 -i $VIDEO\_IN -t 5 -c copy $VIDEO\_OUT
```

Seek to position 00h, 01m, 23s and extract a duration of 5 seconds.

Content streams are copied in this example, but could also be re-encoded to any other format.

FFmpeg's Time Syntax

Time Syntax Examples:

- **1.2** = 1.2 seconds
- **200ms** = 200 milliseconds
- **200us** = 200 microseconds
- **01:02:03** = 1hour, 2min, 3sec

Segmenting

```
DURATION=5
OUT="out-p%02d.mp4"
$FFMPEG -i $IN -f segment -segment_time $DURATION -c copy -map 0 $OUT
```

See: Segment muxer docs

Concatenate Videos

Attaching separate videos together:

```
$FFMPEG -f concat -safe 0 \
  -i <(for f in *.mp4; do echo "file '$PWD/$f'"; done) \
  -c copy -map 0 concat.mkv
```

See: FFmpeg Wiki: Concatenate

Concat Demuxer

- Source: Codecs must match, container may differ
- Requires text list of source files
- Generate list on-the-fly
- "-safe 0": for absolute paths

See: FFmpeg Wiki: Concatenate

Logo Overlay

```
$FFMPEG -i $VIDEO_IN -i $LOGO \
  -filter_complex "overlay=main_w-overlay_w-25:25" \
  ... \
  $VIDEO_OUT
```

You need to declare the logo as additional input source.

End

No homework! :)