Session 6 - Wrapping it up.

The Plan

- Get user input
- Internal Field Separator (IFS)
- Execution: background, parallel, conditional
- Reviewing existing code
- Codequiz
- Q and A

Get user input

Interactive from keyboard (user):

```
read -p "Message" var1 var2 ... varN
```

See: Getting User Input Via Keyboard (cyberciti.biz)

- -p “Message”: Display a message to the user
- var1: 1st input (word) is assigned to variable var1
- var2: 2nd input (word) is assigned to variable var2
- varN: . . . and so on ;)

Hello Who?

```
read -p "What is your name? " NAME
echo "Hello $NAME."
```

```
read -p "What is your name? " FIRST LAST
echo "Hello $LAST $FIRST."
```

Read: More modes

- `read -p`: Show a message (prompt)
- `read -a`: Assign words to array
- `read '-d '`: Set other delimiter (e.g. space)
- `read -t n`: Timeout after n seconds
- . . .

See: `$ help read` or this article on computerhope.com

Internal Field Separator ($IFS)

> “A string treated as a list of characters that is used for field splitting, expansion of the ’*’ special parameter, and to split lines into fields with the read utility."

See: Shell Command Language (POSIX)

$IFS and lists

Changing the separator to ',' (comma):

```
LIST="one,two,three"

IFS=','
for ITEM in $LIST; do
    echo "item: $ITEM"
done
```

…but it could be any character! :)

$IFS and read

Reading a textfile line by line:

```
I=1
while IFS= read -r LINE
do
  echo "$I: $LINE"
  I=$((I + 1))
done < mylist.txt
```

mylist.txt:

```
one
two something: \"
three and additional
four
```

Background Execution

Just add an ampersand at the end:

```
$ my_program &
```

Background / Foreground / Suspend

- `Ctrl+Z`: Suspend Job
- `fg [JOB_ID]`: Move to foreground
- `bg [JOB_ID]`: Run in background
- `jobs`: List background jobs

See: 5 Examples to Manage Unix Background Jobs

Parallel Execution

```
ffmpeg -i day.ts -f segment -segment_time 3600 \
    -c:a copy out_%02d.ts

for FILE in out*.ts; do
    MP3_OUT=$(basename "$FILE" .ts).mp3

    # Run FFmpeg separately for each segment.
    # In the background.
    ffmpeg -i $FILE -c:a mp3 -b:a 192k MP3_OUT &
done
```

Parallel Execution

. . . or GNU-parallel-ize it!

```
ls *.ts | parallel ffmpeg -i {} -c:a mp3 -b:a 192k mp3/{}.mp3
```

Conditional Execution

```
# If 'first' succeeds, 'second' will never be executed:
$ first || second

# Only run 'second' if 'first' is successful:
$ first && second
```

You can use the exit status of a program (success or not) and execute a chain of programs after each other - but conditional.

Using Libraries

Including/importing code from other files:

```
source functions.sh

# read_fps() is declared in functions.sh:
read_fps "$VIDEO_IN"
```

See: Import/Source Files in Bash (Dave Eddy)

Reviewing Existing Code

Codequiz

- What are *variables* for?
- What are *functions* for?
- When to use a *loop*?
  - `while` vs `for`?
- How to use parameters?
- What is `eval`?
- What are `<`, `|` and `>` for?
- What do `&&` and `||` mean?

Questions and Answers

It's *your* turn!

- Fin -