# Bash Scripting 2

PREPARED BY ENES ERTEN

## Contents

## Input Output

- Reading Files by lines :

```
#!/bin/bash

echo "Enter a filename to read:"
read FILE

while read -r SUPERHERO; do
	echo "Superhero Name: $SUPERHERO"
done < "$FILE"
```

## Input Output

- File Descriptors and Handles: In Unix and Unix-like computer operating systems, a file descriptor is a unique identifier for a file or other input/output resource, such as a pipe or network socket. File descriptors typically have non-negative integer values, with negative values being reserved to indicate "no value" or error conditions.

# exec [integer number]<>$FILE –> defining file descriptor and assigning to a number.

# command >&[integer number] -> appends the file that points file address (descriptor).

#exec 5<&-     ->  Closing a file descriptor

## Input Output

- File Descriptor script:

```
FILE=$1 # give the name of file in the command line
exec 5<>$FILE # '5' here act as the file descriptor

# Reading from the file line by line using file descriptor
while read LINE; do
   echo "$LINE"
done <&5

# Writing  to the file using descriptor
echo "Adding the date: `date`" >&5
exec 5<&- # Closing a file descriptor
```

## Input Output

- IFS and Delimiting: A variable **IFS** (Internal Field Separator) is being used to specify a particular **delimiter** for string division. IFS is an variable that specifies delimiter.

## Input Output

```
#!/bin/bash

echo "Enter filename to parse: "
read FILE

echo "Enter the Delimiter: "
read DELIM
IFS="read -r $DELIM"
COUNT=1

while read -r CPU MEMORY DISK; do
  echo "$COUNT system resource Consumptions"
  echo "CPU: $CPU"
  echo "Memory: $MEMORY"
  echo "Disk: $DISK"
  COUNT=`expr $COUNT + 1 `
done <"$FILE"
```

## Debugging and Error Handling

- Debugging in Bash Scripts:

  # bash -x script.sh -> debug entire script

  # add set +x and set –x to script to debug a specific part of the script

## Functions

- Functions: Functions are blocks of reusable code; used when you need to do the same tasks multiple times.

```
function myFunction() {
   # Code Goes Here
}
```

- Unlike other languages, calling a function in Bash does not entail using parentheses. Ex: myFunction

# Functions

- Positional Parameters: In functions, it's possible to use positional parameters as arguments. To use positional parameters, you must first reference them within your function. Once defined, you can use your function with arguments that take on the place of the parameters:
- Return Codes: Each function has an exit status, and functions have their own method of dealing with exit statuses. Return codes are simply exit statuses for functions. By default, the return code of a function is simply the exit status of the last command executed within the function

## Functions

```
function functionName() {
# Code Goes Here
return <Return Code>
}
```