GETTING STARTED WITH DEBUGGING SCRIPTS

➢ DEBUGGING SHELL SCRIPTS:

- Computer programmers, like everybody else, are not perfect. This means the programs they write sometimes have small errors, called "bugs," in them.
- Debugging, in computer programming and engineering, is a multistep process that involves identifying a problem, isolating the source of the problem, and then either correcting the problem or determining a way to work around it.
- In most of the programming languages **debugger tool** is available for debugging. A debugger is a tool that can run a program or script that enables you to examine the internals of the script or program as it runs.
- In the shell scripting we don"t have any debugger tool but with the help of command line options (**-n, -v** and **-x**) we can do the debugging.

METHODS OF ENABLING SHELL SCRIPT DEBUGGING MODE:

- **-v (short for verbose)**: Tells the shell to show all lines in a script while they are read, it activates verbose mode.
- **-n (short for noexec or no execution)**: Instructs the shell read all the commands, however doesn't execute them. This option activates syntax checking mode.
- **-x (short for xtrace or execution trace)**: Tells the shell to display all commands and their arguments on the terminal while they are executed. This option enables shell tracing mode.

DISABLING THE SHELL (-N OPTION):

- The -n option, shot for **noexec** (as in no execution), tells the shell to not run the commands. Instead, the shell just checks for syntax errors.

run the script with -n option:

```
$sh -n script.sh
```

Note: It displays only syntax errors.

DISPLAYING THE SCRIPTS COMMANDS (-V OPTION):

- The **-v option** tells the shell to run in **verbose mode**. In practice, this means that shell will echo each command prior to execute the command. This is very useful in that it can often help to find the errors.

execute the script with -v option:
$sh -v script.sh

NOTE: In the above script output, gets mixed with commands of the scripts. But however, with **-v option**, at least you get a better view of what the shell is doing as it runs your script.

Combining the -n & -v Options:
We can combine the command line options (-n & -v). This makes a good combination because we can check the syntax of a script while seeing the script output.

Execute the script with –nv option
$sh -nv script.sh

TRACING SCRIPT EXECUTION (-X OPTION):

- The **-x** option, short for **xtrace** or **execution trace**, tells the shell to echo each command after performing the substitution steps. Thus , we can see the values of variables and commands. Often, this option alone will help to diagnose a problem.
- In most cases, the -x option provides the most useful information about a script, but it can lead to a lot of output.

  Example 1:
  ```
  #!/bin/bash
  clear
  for f in *
  do
  file $f
  done
  ls
  For Execute: $sh +x script.sh
  ```

  Example 2:
  ```
  $ vi filesize.sh
  #!/bin/bash
  for filesize in $(ls -l . | grep "^-" | awk '{print $5}')
  do
  let totalsize=$totalsize+$filesize
  done
  ```

echo "Total file size in current directory: $totalsize"
For Execute: **$ ./filesize.sh**
Total file size in current directory: 652
Execute Shell script with debug option:
$ bash -xv filesize.sh

USING SET SHELL BUILT-IN COMMAND:

- The third method is by using the set built-in command to debug a given section of a shell script such as a function. This mechanism is important, as it allows us to activate debugging at any segment of a shell script.
- We can turn on debugging mode using **set** command in form below, where option is any of the debugging options.

```
$set option
```

To enable debugging mode, use:
```
$set -option
```

To disable debugging mode, use:
```
$set +option
```

we can disable all of them at once:
```
$set -
```

This script enables shell tracing (the -x option):
```
$set -x
$sh script.sh
```

Enabling debugging using set:
```
$set -x
$./script.sh
```

Disabling Debugging Using set:
```
$set +x
$sh script.sh
```

➢ DEBUGGING COMMON BASH SHELL SCRIPTING ERRORS:

- Bash or sh or ksh gives various error messages on screen and in many case the error message may not provide detailed information.

Skipping to apply execute permission on the file

bash: ./hello.sh: Permission denied

End of file unexpected Error

If you are getting an End of file unexpected error message, open your script file and and make sure it has both opening and closing quotes.

Missing Keywords Such As fi, esac, ;;, etc.

If you missed ending keyword such as fi or ;; you will get an error such as as "xxx unexpected". So make sure all nested if and case statements ends with proper keywords.

Moving or editing shell script on Windows or Unix boxes

Do not create the script on Linux/Unix and move to Windows. Another problem is editing the bash shell script on Windows 10 and move/upload to Unix server. It will result in an error like command not found due to the carriage return (DOS CR-LF)