# Shell Script Programming:

Q. What is a Shell?

- Shell is an application program which acts as an interpreter between user and system, i.e. it accept command in a text format from user, interpret it and pass it to the system for execution.
- Shell is a CLI (Command Line Interpreter).

- In a single Linux system, there may exists more than one shell programs, but at a time only one shell program may be active.
- Shell Program: "sh" compatible shell programs:
  - bsh - bourne shell
  - bash - bourne again shell
  - csh - c shell
  - zsh -
  - sh -
  - ksh - korn shell by david korn
  - etc...

- Command to check available shell programs in a system:
```
$ls -l /bin/*sh
```
- Command to check name of active shell program:
```
$echo $SHELL
```
SHELL is a shell variable which always contains an absolute path of currently active shell program.

editor	: vim editor / vs code editor / gedit
bash shell	: shell program which will going to execute script

# "vim editor": vi improved editor /text editor

- it is an "application program" used for editing text files, and mostly used as a source code editor.
- $vim filename
- "vim" command opens a file in vi editor, if file exists already, if file does not exists, then new file gets opened for editing.
- vi editor works in two modes:

1. "command mode" : we can give commands
2. "insert mode / edit mode" : we can insert / edit data into the file only in insert mode.

- when we open any file in vi editor, bydafult it gets opened in a command mode
- if we want switch from command mode to insert mode we need to press key "i"
- if we want switch from insert mode to command mode we need to press "esc" key, and in command mode we can give commands in front colon

:q	=> quit without save
:w	=> to save file contents
:wq	=> save and quit
:wqa	=> save and quit all files if multiple files opened at once in vi editor
:next OR :n	=> switch to next file if multiple files opened at once in vi editor
:prev	=> switch to prev file if multiple files opened at once in vi editor
:y	=> copy line and move cursor whereever you want to paste it and press key "p"

:m, ny => copy lines from line no. "m" to line no. "n" and move cursor whereever you want to paste it and press key "p"

:d => cut the line move cursor whereever you want to paste it and press key "p"

:m, nd => delete lines from line no. "m" to line no. "n"

[alt+u] => short cut key for undo in vi editor

- we can write one time settings for vi editor in a congiguration file named as ".vimrc" and this file must exist at user's home dir
- filename starts with dot indicates its a hidden file
- we can open or create .vimrc file in user's home dir and can write following one time settings for vi editor:

```
$vim ~/.vimrc
OR goto user's home dir $cd ~ and $vim .vimrc
```

```
set autoindent
set smartindent
set tabstop=2
set cindent
set number
```

"steps to write shell script program":
step-1: write a script by using any editor and save the file/program by giving dot extension as .sh for sake of convenience.

```
	$vim script_01.sh
```

step-2: assign execute perms for the script by using below command

```
	$chmod +x script_01.sh
```

step-3: execute script

```
	way-1  => $./script_01.sh
	way-2  => $bash script_01.sh
```

- by means of executing script at once on terminal, set of commands inside it gets executed by the bash shell automatically in sequence.

- Shell Script Programming Language is "typeless", hence no need of prior definition of any variable,
- In Shell Script Programming Language, "Variable" is a like a container which will going to contains any type of value and type of value can be decided during runtime.

- Shell Script Programming Language, there are 3 types of quotes:
1. single quotes [' '] => we mention char constant in a single quotes
2. double quotes [" "] => we mention string constant in a double quotes
3. back quotes [` `] => if we want assign/store an output of any command to any variable or we want use output of any command we need to mention that command in back quotes.

# DAY-02:

year must be completely divisible by 4 => yr % 4 == 0

year should not be a century year
esception : after every 400 years there is a century year which is a leap year

```
if( yr % 4 == 0 )
{
        //it should not be a century year
        if( yr % 100 != 0 )
                print "year is a leap year"
        else if( yr % 400 == 0 )
                print "year is a leap year"
        else//if year is a century year and not divisible by 400
                print "year is not a leap year"
}
else
        print "year is not a leap year"
```

+ if statement:

```
if [ condition ]
then
        statement/s
fi
```

- statement/s inside if block executes only if condition is true.

+ if-else statement:

```
if [ condition ]
then
        statement/s - 1
else
        statement/s - 2
fi
```

- if condition is true then statement/s-1 inside if block executes otherwise statement/s-2 will executes.

+ if-else ladder

```
if [ condition ]
then
        statement/s-1
elif [ condition ]
then
        statement/s-2
```

```
elif [ condition ]
then
        statement/s-3
else
        statement-s-4
fi
```

# operators in Shell Script Programming Language:

# relational operators:

-gt	=> greater than
-ge	=> greater than or equal to
-lt	=> less than
-le	=> less than or equal to
-eq	=> equal to equal to
-ne	=> not equal to

# logical operators:

-a	=> logical AND
-o	=> logical OR

```
if( ( yr % 4 == 0 ) && ( yr % 100 != 0 ) || ( yr % 400 == 0 ) )
        year is a leap year
else
        year is not a leap year
```

# loops:
- there are 3 types of loops:

1. while loop

```
while [ condition ]
do
        statement/s
done
```

- statement/s inside while loop gets executes till condition is true, as soon as condition becomes false while loop gets break.

2. until loop

```
until [ condition ]
do
	statement/s
done
```

- statement/s inside until loop gets executes till condition is false, as soon as condition becomes true until loop gets break.

3. for loop

```
for var_name in collection
do
	statement/s
done
```

- statement/s inside for loop execute no. of elements in a collection, number of time, in each iteration value of loop counter will be taken sequentially from collection

# test commmands:

-e filepath	=> to check filepath is valid or not
-f filepath	=> to check filepath is a regular file or not
-d filepath	=> to check filepath is a directory or not
-x filepath	=> to check filepath is having execute perms or not
-w filepath	=> to check filepath is having write perms or not
-r filepath	=> to check filepath is having read perms or not

```
ls
{
	class_work.txt  file2.txt     script_02.sh  script_04.sh  script_06.sh
	file1.txt       script_01.sh  script_03.sh  script_05.sh  script_07.sh
}
```

# Positional parameters in shell script programming langauge <=> command line arguments in C:
- parameters which we passed to the script while executing it from terminal are referred as positional parameters.

argv[0] : name of an executable file	<=> $0 : name of the script
argv[1] : first argument	<=> $1 : first pos param
argv[2] : second argument	<=> $2 : second pos param

.
.

.

argc : argument counter <==> $# : no. of pos params = count

- bydefault argc = 1 <==> bydefault $# => no. pos params = 0

$* => all positional parameters => collection/array of all pos params

+ "function":
- function is an indepedent block of statements which does a specific task
- In C, we need to first declare a function for compiler before its call, and then we need to write definition

```
<return_type> <function_name>( function signature );


<return_type> <function_name>( function signature )
{
	//function body:
	//statement/s
}
```

- statement/s inside function definition gets executed only after giving call to that function.
- In C, to give call to any function min we required 2 things:

	1. function name => function name itself is an starting addr of block of function definition
	2. () => function call operator

```
fun();
```

- we have to write function definition at the begining of the script
- general syntax:

```
function <function_name>( )
{
	//function body
	//statement/s
}
```

- function => keyword
- function_name => identifier
- ( ) followed by function name
- we can write actual logic of function inside curly braces

- statement/s inside function gets executed only after giving call to that function

- if we use/mention $1, $2, $3, ... globally (i.e. outside any function) in the script => it will be treated as positional params.

- if we use/mention $1, $2, $3, ... inside any function in a script => it will be treated as arguments passed to the function : formal params

- there are 2 ways by which we can give call to the function as per our requirement:

# way-1: if we do not expect that function should return value

function_name arg1 arg2 arg3

arg1, arg2, arg3 => actual parameters

```
table : table of 25 is :
25
50
75
100
125
150
175
200
225
250
```

- depends on way of function calling behaviuor of echo statement/command is changed
- in way-1 of function calling, echo command is used for printing purpose
- in way-2 of function calling, echo command is used for storing purpose

```
table={
	"table of 25 is :"
	"25"
	"50"
	"75"
	"100"
	.
	.
	"250"
}
```

# switch control block

```
case $choice in
	0)
```

```
                statement/s
                ;;      # break


        1)
                statement/s
                ;;      # break

        2)
                statement/s
                ;;      # break

        *)      #default case
                statement/s
                ;;      # break
esac
```

- shell script is a simple text file in which we can write :
    - set of commands
    - set commands with some instructions in shell script programming langauge syntax
    - call to other binaries

- by means of executing script once, set of commands, set of commands with some instructions written in scripting lanaguage syntax and binaries gets executed implicitly, i.e. we can use shell script to achieve automation.

```
 /home/sunbeam/feb2022/DAC/OS/syscalls/process_mgmnt/myshell.out
```

# A few examples of applications shell scripts can be used for include:
- Automating the code compiling process.
- Running a program or creating a program environment.
- Completing batch.
- Manipulating files.
- Linking existing programs together.
- Executing routine backups.
- Monitoring a system

"UNIX Shell - By Sumitba Das"