What is Bash?

Bash is Bourne Again Shell. It is a shell in linux. shell is a software that interpretes your commands and runs your command on operating system.

Bash scripts end with .sh

Following are the scripts that cover features of Bash Scripting Language. In the following scripts description of the scripts are given as comments in the scripts. Comments in Bash script start with #.

```
#!/bin/bash

echo "Hello World"
```

```
#!/bin/bash

student="Raman"

echo "Hello ${student}"
```

```
#!/bin/bash

rollno=1

name="Raman"

echo $rollno

echo $name

echo $PATH
```

```
# to print shell variable PATH


#!/bin/bash

echo $PATH
```

```
#!/bin/bash

# to print home variable

echo $HOME
```

```
#!/bin/bash

#to print name of current user who is logged in

echo $USER
```

```
#!/bin/bash

#to print hostname

echo $HOSTNAME
```

```
#!/bin/bash
#to print HOST TYPE
echo $HOSTTYPE
```

```
#!/bin/bash
#change first letter of name variable as a lowercase letter
name=RaMAN
echo ${name,}
```

```
#!/bin/bash
#change all letters of name variable as a lowercase letters
name=RaMAN
echo ${name,,}
```

```
#!/bin/bash
#change first letter of name variable as a uppercase letters
name=raman
echo ${name^}
```

```
#!/bin/bash
#change all letters of name variable as a uppercase letters
name=raman
echo ${name^^}
```

```
#!/bin/bash
#to find length of name variable
name=raman
echo ${#name}
```

```
#!/bin/bash
#to find slice or find portion of name variable
name="raman deep"
echo ${name:0:8}

#to start slicing from letter at index 3
echo ${name:3}
```

```
#to start slicing from left side

echo ${name: -3}


#to start slicing from left side

echo ${name: -3:2}
```

```
#!/bin/bash

ls
```

---

parameter expansion : ${paramater}

command expansion : $(command)

aritmetic expansion : $((expression))

```
#!/bin/bash

# saving command output in a variable, example of command substitution


a=$(date)

echo "The date is $a"
```

```
#!/bin/bash

# example of aritmetic expansion


echo $(( 5 + 10 ))


x=10

y=20


echo $(($x + $y))


echo $((x + y))
```

```
echo $((x - y))

echo $((x * y))

echo $((x / y))


echo $(( (2+3)*4))

echo $((2**5))

echo $((23%10))
```

```
#!/bin/bash

#to find 5/2 with decimal numbers using bc and scale command


echo "scale=2; 5/2" | bc
```

```
#!/bin/bash

#to print home directory using tide ~


echo ~
```

```
#!/bin/bash

#using ~ to switch between present working directory and old working directory

#Tilde expansion within the shell is useful when writing scripts that need to work across multiple directories


cd ~-
```

```
#!/bin/bash

#example of brace expansion


echo {jan,feb,mar,apr,may,jun}

echo {1..10}

echo {10..1}

echo {a..z}

#to put gap between numbers of 3 from 1 to 30

echo {1..30..3}
```

```
#Quoting in Bash
```

Use the backslash to remove special meaning from the next character

Use single quotes to remove special meaning from all the characters within them

Use double quotes to remove special meanings from all except dollar signs ($) and backticks(')

#!/bin/bash

#ls command to list all directories and all files within that directory

ls *

#? character in globbing

ls ?ile?.sh

to use square bracket to specify more than one character [12] means file1.sh and file2.sh

ls file[12].sh

to use square bracket to specify more than one character [1-9] means file1.sh to file9.sh

ls file[1-9].sh

#!/bin/bash

#Data Streams

#Stream 0 = Standard Input (stdin)

#Stream 1 = Standard Output (stdout)

#Stream 2 = Standard Error (stderr)

#create a file file2.txt in the same directory and add text to it This is file2

cat < file2.txt

```
#output of above command will be This is file2

#redirecting output to output.txt

echo "this is some output" > output.txt

#redirecting hello to hello.txt, 1> means you output "hello" to standard output stream
echo "hello" 1> hello.txt

#appending hello to hello.txt, 1>> means you append output "hello" to standard output stream
echo "hello" 1>> hello.txt
```

```
#!/bin/bash

#Example of Positional Parameters

echo "Name is $1"
echo "Address is $2"
echo "Email is $3"
```

```
#!/bin/bash

#Example of Special Parameter $#

echo "Number of Parameters passed to script are $#"

#Example of $0 parameter to print name of script

echo "Name of script is $0"
```

```
#!/bin/bash

#Example of special parameter $@
```

```
echo "All the positional parameters are :  $@"

#Example of special parameter $*
IFS=,
echo "All the positional parameters separated with the first letter of IFS variable are :  $*"
```

```
#!/bin/bash

read var1 var2
echo "Value of variable 1 is $var1"
echo "Value of variable 2 is $var2"

#-p option is used to give prompt to user

read -p "What is your name " name
echo "You entered your name $name"

#-t option is used to give time to user to respond , time is given in seconds, if the time expires script goes to next command

read -t 5 -p "What is your name " name
echo "You entered your name $name"

#-s option is used to hide the data entered by the user on the screen

read -s -p "What is your name " name
echo "You entered your name $name"
```

```
#!/bin/bash

#read command will store the value in variable $REPLY

read
```

```
echo $REPLY
```

```
#!/bin/bash

#select command is used to give users options to select, users will be option with numbers from
# 1 to 7 if user selects 5 day will be friday and after that select command will end

select day in mon tue wed thu fri sat sun;
do
echo "The day of the week is $day"
break
done
```

```
#!/bin/bash

#select command is used to give users options to select, users will be option with numbers from
# 1 to 7 if user selects 5 day will be friday and after that select command will end
#PS3 variable is used to give user a prompt

PS3="What is the day of the week?"
select day in mon tue wed thu fri sat sun;
do
echo "The day of the week is $day"
break
done
```

```
#!/bin/bash

#; list operator waits for previous command to be completely run

# && list operator will make second command run if the first command was successful

# || list operator makes second command runs only if the first one failed
```

& list operator runs command asynchronously or in the background

---

```
#!/bin/bash

#test command is placed in square brackets

[ 2 -eq 2 ]  ;  echo $?

#for above command result of test command will be 0

[ 1 -eq 2 ]  ;  echo $?

#for above command result of test command will be 1

[ 1 -ne 2 ]  ;  echo $?

#for above command result of test command will be 0

# -gt can be used for greater than
# -lt can be used for less than
# -geq can be used for greater than or equal to
# -leq can be used for less than or equal to
#these operators work for integers
```

```
#!/bin/bash

a=hello
b=goodbye
[[ $a = $b ]] ; echo $?

#output will be 1 because value in variable a and b are not equal

[[ $a != $b ]] ; echo $?
```

```
#output will be 0

#how to check whether a variable is empty or not

[[ -z $c ]] ; echo $?

#output will be 0 because variable c is empty

# -n means you are checking for non-empty string
c=anything
[[ -n $c ]] ; echo $?

#Test File operators
[[ -e today.txt ]] ; echo $?

#output of the above command is 1 because today.txt does not exist

#-f checks for file and -d operator checks for directory
# -x checks whether a file is a script or not
# -r for readable file and -w for writable file
```

```
#!/bin/bash

# example of if statement

if [ 2 -gt 1 ]; then
        echo test passed
fi
```

```
#!/bin/bash

# example of if else statement
```

```
if [ 2 -eq 1 ]; then
        echo test passed
else
        echo test failed
fi
```

```
#!/bin/bash

# example of if elif statement

if [ 2 -eq 1 ]; then
        echo test passed
elif [ 1 -eq 1 ]; then
        echo second test passed
else
        echo test failed
fi
```

```
#!/bin/bash

# example of if statement and && operator

a=$(cat file1.txt)
b=$(cat file2.txt)
c=$(cat file3.txt)
if [ $a = $b ] && [ $a = $c ]; then
        echo "three files match"
else
        echo "Files do not match"
fi
```

```
#!/bin/bash
```

```
# example of || operator
marks=80
if [ $marks -eq 80 ] || [ $marks -eq 90 ]; then
        echo "marks are either 80 or 90"
else
        echo "marks are neither 80 or 90"
fi
```

```
#!/bin/bash

read -p "Please enter a number : " number
case "$number" in
        [0-9]) echo "you have entered a single digit number";;
        [0-9][0-9]) echo "you have entered a two digit number";;
        [0-9][0-9][0-9]) echo "you have entered a three digit number";;
        *) echo "you have entered a number that is more than three digits"
esac
```

```
#!/bin/bash

read -p "Enter a number to print numbers from 0 to the number : " num
i=0
while [ $i -lt $num ]; do
        echo $i
        i=$(( $i+1 ))
done
```

```
#!/bin/bash

#read a file with filename passed as argument

while read line; do
        echo "$line"
```

```
done < "$1"
```

```
#!/bin/bash

#example of indexed array

numbers=(1 2 3 4)
echo $numbers
#output will be 0 because when specifying only array name we get element at index 0

echo ${numbers[2]}
#output will be 3 as 3 is present at index 2

echo ${numbers[@]}
#output will be whole array

echo ${numbers[@]:1}
#output will be 2 3 4

echo ${numbers[@]:1:2}
#output will be 2 3

#adding element 5 to this array
numbers+=(5)
echo ${numbers[@]}

#output will be 1 2 3 4 5

#unset can be used to delete element based on its index
unset numbers[2]
echo ${numbers[@]}
```

```
#change an element based on its index
numbers[0]=a
echo ${numbers[@]}
```

```
#!/bin/bash

#read contents of a file in an indexed array

#suppose there is a file days.txt that contains name of days with dayname in every line

readarray -t days < days.txt
echo ${days[@]}

#Output will be Monday Tuesday Wednesday Thursday Friday Saturday Sunday
```

```
#!/bin/bash

#example of simple for loop

for n in a b c;
do
  echo $n
done
```

```
#!/bin/bash

#example of range based for loop

for n in {1..5};
do
   echo $n
done
```

```
#!/bin/bash
```

```
#script to print numbers from 1 to 5 with a gap of 2

for n in {1..5..2};
do
   echo $n
done
```

```
#!/bin/bash

#iteration of array using for loop

s=("football" "cricket" "hockey")
for n in ${s[@]};
do
   echo $n
done
```

```
#!/bin/bash

#example of C styled for loop

n=7
for (( i=1 ; i<=$n ; i++ ));
do
   echo $i
done
```
