7 @koni_raid `ctrl+shift+PrntScr` is amazing. – viveksinghggits Mar 27 '19 at 15:44

- Theme: light + (default light)

1. Introduction

- Shell scripts are interpreted not compiled
- To see installed shells in your machine
  - cat /etc/shells
- sh is borne shell….still used in unix system, the first shell which was used
- bash ...born again shell…. Standard gnu which is intutive and flexible
- to see where bash is located
  - which bash
- #! shebang/hashbang

$_ 1_hello_world.sh

```
#! /bin/bash

echo "Hello world"
```

2. Using Variables and Comments

- Two types of variables
  - system variable (Capital case)
  - user defined variables

$_ 2_Variables&Comments.sh

```
# /bin/bash


# System defined varibales

echo $BASH
echo $HOME
echo $PWD

# User defined variables

name="rashid is learning bash scripting now"
age=34.0
echo "Who is learning linx?" $name # echo statement can be without quotes
echo "Rashid age is: " $age

```

3. Read User Input

$_ 3_read_user_input.sh

```bash
#! /bin/bash

# This will prompt the user to enter on the next line
echo "Please enter name of three students: "
read name1 name2 name3

echo "Your students names are: $name1 , $name2 , $name3"

# OR
# If you want to prompt user to enter on the same line

read -p "Please enter your name here: " name

echo "Your name is $name"
```

```bash
# If you want to prompt user to enter on the same line
read -p "Please enter your name here: " name

# If the information enter are confidential such as passwords then
read -sp "Please enter your password here: " password

echo -e "\nYour name is $name" #flag -e will print next line
echo "Your password is $password"
```

```bash
# If you do not enter variable for receiving information from user

echo "Please enter your name: "
read

echo "Your name is $REPLY"
```

4. Pass Arguments to a Bash Scripting

$_ 4_pass_arguments_to_bash.sh

```
#! /bin/bash

# Passing arguments to bash scripts store in $1, $2 and so on...

echo $1 $2 $3 '> echo $1 $2 $3'
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE bash

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./4_pass_arguments_to_bash.sh rashid yasir abdullah
rashid yasir abdullah > echo $1 $2 $3
```

```
# To print name of the file as well write
echo $0 $1 $2 $3 '> echo $1 $2 $3'

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE bash

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./4_pass_arguments_to_bash.sh rashid yasir abdullah
rashid yasir abdullah > echo $1 $2 $3
./4_pass_arguments_to_bash.sh rashid yasir abdullah > echo $1 $2 $3
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

```
# Pass the arguments from the user as an array

args=("$@") # stores arguments as an array

echo ${args[0]}, ${args[1]}, ${args[2]}

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE bash

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./4_pass_arguments_to_bash.sh rashid yasir abdullah
rashid, yasir, abdullah
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

```
# Pass the arguments from the user as an array

args=("$@") # stores arguments as an array

# echo ${args[0]}, ${args[1]}, ${args[2]}
echo $@ # prints all the arguments stored in the array

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE bash

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./4_pass_arguments_to_bash.sh rashid yasir abdullah yasin
rashid yasir abdullah yasin
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

```
# Pass the arguments from the user as an array

args=("$@") # stores arguments as an array

# echo ${args[0]}, ${args[1]}, ${args[2]}
echo $@ # prints all the arguments stored in the array
echo $# # prints the number of argument passed in the array
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE bash

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./4_pass_arguments_to_bash.sh rashid yasir abdullah yasin
rashid yasir abdullah yasin
4
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

5. Conditional Statements

$_ 5_if_statement.sh

```
Syntax of if statement
if [ expression ]
then
    statement
fi
```

Comparison Operators

Integer Comparison

-eq	is equal to	`if [ "$a" -eq "$b" ]`
-ne	is not equal to	`if [ "$a" -ne "$b" ]`
-gt	is greater than	`if [ "$a" -gt "$b" ]`
-ge	is greater than or equal to	`if [ "$a" -ge "$b" ]`
-lt	is less than	`if [ "$a" -lt "$b" ]`
-le	is less than or equal to	`if [ "$a" -le "$b" ]`
<	is less than (within double parentheses)	`(("$a" < "$b"))`
<=	is less than or equal to (within double parentheses)	`(("$a" <= "$b"))`
>	is greater than (within double parentheses)	`(("$a" > "$b"))`

>= is greater than or equal to (within double parentheses) **(("$a" >= "$b"))**

String Comparison

= is equal to **if ["$a" = "$b"]**

Note the whitespace framing the =.

if ["$a"="$b"] is *not* equivalent to the above.

== is equal to **if ["$a" == "$b"]**

This is a synonym for =.

The == comparison operator behaves differently within a double-brackets test than within single brackets.

```
[[ $a == z* ]]   # True if $a starts with an "z" (pattern matching).
[[ $a == "z*" ]] # True if $a is equal to z* (literal matching).

[ $a == z* ]     # File globbing and word splitting take place.
[ "$a" == "z*" ] # True if $a is equal to z* (literal matching).

# Thanks, Stéphane Chazelas
```

!= is not equal to **if ["$a" != "$b"]**

This operator uses pattern matching within a [[...]] construct.

< is less than, in ASCII alphabetical order

if [["$a" < "$b"]]

if ["$a" \< "$b"]

Note that the "<" needs to be escaped within a **[ ]** construct.

> is greater than, in ASCII alphabetical order

if [["$a" > "$b"]]

if ["$a" \> "$b"]

Note that the ">" needs to be escaped within a **[ ]** construct.

See Example 27-11 for an application of this comparison operator.

-z string is *null*, that is, has zero length

```
 String=''   # Zero-length ("null") string variable.

if [ -z "$String" ]
then
  echo "\$String is null."
else
```

```
  echo "\$String is NOT null."
fi     # $String is null.
```

-n string is not *null.*

The **-n** test requires that the string be quoted within the test brackets. Using an unquoted string with *! -z*, or even just the unquoted string alone within test brackets (see Example 7-6) normally works, however, this is an unsafe practice. *Always* quote a tested string.

```
integer comparison

-eq - is equal to - if [ "$a" -eq "$b" ]
-ne - is not equal to - if [ "$a" -ne "$b" ]
-gt - is greater than - if [ "$a" -gt "$b" ]
-ge - is greater than or equal to - if [ "$a" -ge "$b" ]
-lt - is less than - if [ "$a" -lt "$b" ]
-le - is less than or equal to - if [ "$a" -le "$b" ]
< - is less than  - (("$a" < "$b"))
<= - is less than or equal to - (("$a" <= "$b"))
> - is greater than - (("$a" > "$b"))
>= - is greater than or equal to  - (("$a" >= "$b"))

string comparison
= - is equal to - if [ "$a" = "$b" ]
== - is equal to - if [ "$a" == "$b" ]
!= - is not equal to - if [ "$a" != "$b" ]
< - is less than, in ASCII alphabetical order - if [[ "$a" < "$b" ]
> - is greater than, in ASCII alphabetical order - if [[ "$a" > "$b
-z - string is null, that is, has zero length
```

$_ 5_if_statement.sh

```
# Syntax of if statement
count=10
# if (($count >= 5)) or
if [ $count -ge 5 ]
then
    echo "Count is greater than 5"
fi
```

OR

```
a="rashid"
b="rashid"

if [ "$a" == "$b" ]
then
    echo "both strings are equal"
fi
```

```
if [[ "$a" == "$b" ]]
then
    echo "both strings are equal"
else
    echo "both strings are not equal"
fi
```

```
16
17  word=a
18
19  if [[ $word < 'b' ]]
20  then
21      echo "a is smaller than b"
22  fi
23
```

- use double brackets when you are using angle brackets (>,<) i.e., (( )) in case of comparing integers, and [[]] in case of comparing strings
- **Convention** that I will follow….
  - For strings, always use [[ ]] with comparison operators like ==, > etc
  - for integers, always use [] and use urinary operator like -eq, -gt etc.

```
17  word=a
18
19  if [[ $word > 'b' ]]
20  then
21      echo "a is smaller than b"
22  else
23      echo "a is not greater than b"
24  fi
```

```
17  a=5
18
19  if [ $a -gt 5 ]
20  then
21      echo "a is greater than 5"
22  elif [ $a -eq 5 ]
23  then
24      echo "a is equal to 5"
25  else
26      echo "a is smaller than 5"
27  fi
```

6. File Test Operators

```bash
#  File test operators | To check if file exit or not

echo -e "Please enter name of the file: \c" # This will keep the cursor on the same line
read file_name

if [ -e $file_name ] # to check if file exists
then
    echo "The file: $file_name, exists."
else
    echo "The file $file_name, DOES NOT exists."
fi
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE bash

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./6_file_test_operators.sh
Please enter name of the file: 6_rashid.txt
The file: 6_rashid.txt, exists.
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

```bash
#  File test operators | To check if file file is regular or not

echo -e "Please enter name of the file: \c" # This will keep the cursor on the same line
read file_name

if [ -f $file_name ] # to check if file is regular
then
    echo "The file: $file_name, exits and it is a regular file."
else
    echo "The file $file_name, either DOES NOT exists or it is NOT a regular file."
fi
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE bash

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ mkdir test
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./6_file_test_operators.sh
Please enter name of the file: test
The file test, either DOES NOT exists or it is NOT a regular file.
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./6_file_test_operators.sh
Please enter name of the file: 6_rashid.txt
The file: 6_rashid.txt, exits and it is a regular file.
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

```bash
#  File test operators | To check for the directory

echo -e "Please enter name of the the directory: \c" # This will keep the cursor on the same li
read dir_name

if [ -d $dir_name ] # to check if file is regular
then
    echo "The directory: $dir_name, exits."
else
    echo "The directory: $dir_name, DOES NOT exist."
fi
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE bash

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./6_file_test_operators.sh
Please enter name of the the directory: test
The directory: test, exits.
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

- Two types of files
  - character special file: normal file (**-b**)
  - block special file: binary file (**-c**)
- To check whether file is empty or not (**-s**)
- To check whether file has read, write, executable permission (**-r**, **-w, -x**)

```
#  File test operators | To check whether the file is empty or not

echo -e "Please enter name of the the directory: \c" # This will keep the cursor on the same li
read file_name

if [ -s $file_name ] # to check if file is empty
then
    echo "The file: $file_name, is not empty"
else
    echo "The file: $file_name, is empty"
fi
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE bash

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./6_file_test_operators.sh
Please enter name of the the directory: 6_rashid.txt
The file: 6_rashid.txt, is empty
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

7. How to append output of the end of the file

```
# Appending the output to the end of the file

echo -e "Please enter name of the file: \c"
read file_name

if [ -f $file_name ]
then
    echo "$file_name found"
    if [ -w $file_name ]
    then
        echo "$file_name has write permission"
    else
        echo "$file_name DOEST NOT have write permission"
    fi
else
    echo "$file_name NOT found"
fi
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./7_appending_to_file.sh
Please enter name of the file: 6_rashid.txt
6_rashid.txt found
6_rashid.txt has write permission
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

- cat > : will over-write the file
- cat >> : will append the data at the end of the file

```
# Appending the output to the end of the file

echo -e "Please enter name of the file: \c"
read file_name

if [ -f $file_name ]
then
    if [ -w $file_name ]
    then
        echo "Enter your text into the file, to quit press ctrl+d"
        cat >> $file_name
    else
        echo "$file_name DOES NOT have write permission"
    fi
else
    echo "$file_name NOT found"
fi
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./7_appending_to_file.sh
Please enter name of the file: 6_rashid.txt
Enter your text into the file, to quit press ctrl+d
I will InshaAllah explore it and make gui of VSpipe.
in UK
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

8. Logical 'AND' Operator

```
# AND operator with if statement

age=15

if [ $age -gt 18 ] && [ $age -lt 30 ]
then
    echo "The age of the student is between 18 and 30"
else
    echo "The student is either younger than 18 or older than 30"
fi
```

```
# AND operator with if statement

age=15

# if [ $age -gt 18 ] && [ $age -lt 30 ] or
if [ $age -gt 18 -a $age -lt 30 ]
then
    echo "The age of the student is between 18 and 30"
else
    echo "The student is either younger than 18 or older than 30"
fi
```

```
# AND operator with if statement

age=15

# if [ $age -gt 18 ] && [ $age -lt 30 ] or
# if [ $age -gt 18 -a $age -lt 30 ] or
if [[ $age -gt 18 && $age -lt 30 ]]
then
    echo "The age of the student is between 18 and 30"
else
    echo "The student is either younger than 18 or older than 30"
fi
```

9. Logical 'OR' Operator

```
age=11

# if [ $age -gt 18 -o $age -lt 30 ] or
# if [[ $age -gt 18 || $age -lt 30 ]] or

if [ $age -gt 18 ] || [ $age -lt 12 ]
then
    echo "condition true if the age is either less than 12 or greater than 18"
else
    echo "condition is false if age is between 12 and 18"
fi
```

10. Perform Arithmetic Operations

```
# Adding two numbers

n1=10
n2=5

echo $(( n1+n2 ))
```

```
# Adding two numbers

n1=10
n2=5
# echo $(( n1+n2 )) OR

echo $( expr $n1 + $n2 )
```

```
# Adding two numbers

n1=10
n2=5

n3=$(( n1+n2 ))
echo $n3
```

```
# multiplication

n1=10
n2=5
# echo $(( n1*n2 )) OR

echo $( expr $n1 \* $n2 )
```

11. Floating point math operations in bash | bc command

- **echo "10.5+5" | bc**

```
# Arithmetic operation of decimal numbers

n1=10.5
n2=5

echo "10.5/5" | bc # will print 2
echo "scale=2;10.5/5" | bc # will print 2.10
```

```
# Arithmetic operation of decimal numbers

n1=10.5
n2=5

echo "$n1+$n2" | bc
```

```
# To calculate square root
num=9

echo "scale=2; sqrt($num)" | bc
```

- echo "scale=2; 3^3" | bc
-

12. The Case Statement / 13. The Case Statement Examples

```
# Syntax
case expression in
    pattern1 )
        statements ;;
    pattern2 )
        statements ;;
    ...
esac
```

```

vehicle=$1

case $vehicle in
    "car" )
        echo "The rent of $vehicle is 100 USD" ;;
    "cab" )
        echo "The rent of $vehicle is 60 USD" ;;
    "van" )
        echo "The rent of $vehicle is 80 USD" ;;
    "truck" )
        echo "The rent of $vehicle is 150 USD" ;;
    * )
        echo "Unknown vehicle" ;;
esac
```

```
# To evaluate expression entered by the user

echo -e "Please enter a character: \c"
read user_input

case $user_input in
    [a-z] )
        echo "User entered $user_input, a to z" ;;
    [A-Z] )
        echo "User entered $user_input, A to Z" ;;
    [0-9] )
        echo "User entered $user_input, 0 to 9" ;;
    ? )
        echo "User entered $user_input, special character" ;;
    * )
        echo "User entered $user_input, Unknown input" ;;
esac
```

14. Array Variables

```
# Declaraing array
os=('ubuntu' 'windows' 'mac os')

echo ${os[0]} # to print first value; indexes: 0,1,2.....
echo ${os[@]} # to print all the elements of array
echo ${!os[@]} # to print indexes of the array
echo ${#os[@]} # to print length of the array
```

```
# To add elements in the array
os=('ubuntu' 'windows' 'mac os')

os[3]='fedora' # this will add next element in the array
os[2]='openSuSE' # this will update the exisiting value in the array
echo ${os[@]}
```

```
# To remove element from the array
os=('ubuntu' 'windows' 'mac os')

unset os[1] # it will remove 'windows' from the array
echo ${os[@]}
```

```
# You can add element at any index of the array
os=('ubuntu' 'windows' 'mac os')
os[6]='fedora'

echo ${os[@]}
echo ${!os[@]}
echo ${#os[@]}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./14_array_variables.sh
ubuntu windows mac os fedora
0 1 2 6
4
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

- Above code means that some indexes in the array may be left UN-initialized and gaps in the array are OK.

```
# Bash permits array operations on variables also even if the variables are not explicitly declared as an array

mystr=abcdefgh

echo ${mystr[@]} # prints the whole string
echo ${mystr[0]} # prints the whole string
echo ${mystr[1]} # does not print anything
echo ${!mystr[@]} # indexes of the string
echo ${#mystr[@]} # length of the string
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE bash

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./14_array_variables.sh
abcdefgh
abcdefgh

0
1
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

15. WHILE Loops

```
# Syntax of While loop

while [condition]
do
    command1
    command2
    command3
done
```

```
n=1

while [ $n -le 10 ]
do
    echo $n
    n=$((n+1))
done
```

```
n=1

while [ $n -le 10 ] # OR (( $n <= 10 ))
do
    echo $n
    # n=$((n+1)) OR
    # ((n++)) #post increment
    ((++n)) #pre increment

done
```

16. Using sleep and open terminal with WHILE Loops

```
n=1

while [ $n -le 10 ]
do
    echo $n
    n=$((n+1))
    sleep 1 # prints with delay of 1 sec
done
```

```
n=1

while [ $n -le 3 ]
do
    # echo $n
    n=$((n+1))
    gnome-terminal & # opens three terminal windows
done
```

17. Read a file content in bash

```

# Read content of the file

while read p # p = var in which file content will be stored line by line
do
    echo $p
done < 6_rashid.txt
```

```
# Read content of the file

# while read p # p = var in which file content will be stored line by line
# do
#     echo $p
# done < 6_rashid.txt

# OR

cat 6_rashid.txt | while read p
do
    echo $p
done
```

```

# Reading file using IFS (Internal Feel Separator)

while IFS=' ' read -r line # -r prevents back slash escape from being interpretted
do                  # use can use any variable instead of line
    echo $line
done < /etc/host.conf

```

18. UNTIL loop

```

# UNTIL loop: almost similar to while loop with little difference
# it is kind of opposite to while loop
# syntax

until [ condition ] # when this condition false, then its commands are executed
do
    command1
    command2
    command3
done

```

```
n=1

until [ $n -gt 10 ] # it is executed until this condition is false
do
    echo $n
    n=$((n+1))
done
```

19. FOR loop

```
# for loop Syntax
for VARIABLE in 1 2 3 4 5 6 .. N
do
    command1
    command2
    command3
done

# OR -----------------

for VARIABLE in file1 file2 file3
do
    command1 on $VARIABLE
    command2
    command3
done

# OR -----------------

for OUTPUT in $(linux-command-here)
do
    command1 on $OUTPUT
    command2 on $OUTPUT
    command3
done

# OR -----------------

for ((EXP1, EXP2, EXP3))
do
    command1 on $OUTPUT
    command2 on $OUTPUT
    command3
done
```

```
for var in 1 2 3 4 5
do
    echo $var
done
```

```

# for var in 1 2 3 4 5
for var in {1..10}
do
    echo $var
done
```

```
# {start..end..increment}
for var in {1..10..2}
do
    echo $var
done
```

- echo ${BASH_VERSION}

```
for (( var=0; var <= 10; var++ ))
do
    echo $var
done
```

20. FOR loop to execute commands

```

# for command in list_of_command
for command in ls pwd date
do
    echo ----------$command-----------
    $command # to execute the command
done
```

```

# to print all the directories in home directory

for item in ~/*
do
    if [ -d $item ]
    then
        echo $item
    fi
done
```

21. Select Loop

```
# Select loop to print menu --> Syntax

select var in list
do
    command1
    command2
    command3
done
```

```
select name in Rashid Yasir Abbas Yasin
do
    echo $name is selected
done
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./21_select_loop.sh
1) Rashid
2) Yasir
3) Abbas
4) Yasin
#? 1
Rashid is selected
#? 2
Yasir is selected
#?
```

```
select name in Rashid Yasir Abbas Yasin
do
    case $name in
    Rashid)
        echo "learning bash scripting" ;;
    Yasin)
        echo "learning python" ;;
    Yasir)
        echo "doing business" ;;
    *)
        echo "ERROR! Please provide number 1..4"
    esac
done
```

22. Break and continue statement

```
# Break statement: to exit the loop before its normal execution

for (( i=0; i<=10; i++ ))
do
    echo $i
    if [ $i -eq 5 ] # it will quit the loop when i=5
    then
        break
    fi
done
```

```

# Continue statement: it will skip the current iteration of the loop

for (( i=0; i<=10; i++ ))
do
    if [ $i -eq 3 ] || [ $i -eq 6 ] # it will skip the iteration when i=3, i=6
    then
        continue
    fi
    echo $i
done
```

23. Functions

```
# function syntax

function name(){
    commands
}

# OR ---------

name(){
    commands
}
```

```
# function definition
function helloWorld(){
    echo "Hello world !!!"
}

helloWorld # function call
```

```
# Passing parameters in function

function displayDetail(){
    echo "Hell $1"
}

displayDetail 'Rashid' # function call
```

24. Local Variables

$_ 24_local_variable.sh

```bash
#! /bin/bash
# Local and global variables
function print(){
    name=$1
    echo $name
}

name="Yasir"
echo "The name is $name, BEFORE function call print()"
print 'Rashid'
echo "The name is $name, AFTER function call print()"

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./24_local_variable.sh
The name is Yasir, BEFORE function call print()
Rashid
The name is Rashid, AFTER function call print()
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

```bash
# Local and global variables
function print(){
    local name=$1
    echo $name
}

name="Yasir"
echo "The name is $name, BEFORE function call print()"
print 'Rashid'
echo "The name is $name, AFTER function call print()"

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./24_local_variable.sh
The name is Yasir, BEFORE function call print()
Rashid
The name is Yasir, AFTER function call print()
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

25. Function Examples

```
# function to check whether a file exists or not

is_file_exit(){
    local file_name=$1
    [[ -f $file_name ]] && return 0 || return 1 # if condition
}

usage() {
    echo "You need to provide an argument"
    echo "usage: $0 file_name"
}

[[ $# -eq 0 ]] && usage

if ( is_file_exit $1 )
then
    echo "File exists"
else
    echo "File does not exist"
fi
```

26. Read only command

```

# read only variable
# Cannot be overwritten
var=5

readonly var
var=4
echo "var = $var"
```

```
10
11  # # read only function
12  # # Cannot be overwritten
13
14  function hello(){
15      echo "Hello world !!!"
16  }
17
18  readonly -f hello
19
20  function hello(){
21      echo "Hello world Again !!!"
22  }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./26_read_only_var.sh
./26_read_only_var.sh: line 22: hello: readonly function
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

```
23
24  # To see all read only variables
25  readonly
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE bash

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./26_read_only_var.sh
declare -r BASHOPTS="checkwinsize:cmdhist:complete_fullquote:extquote:force_fignore:globasciiranges:hostcomplete:inter
active_comments:progcomp:promptvars:sourcepath"
declare -ar BASH_VERSINFO=([0]="5" [1]="0" [2]="17" [3]="1" [4]="release" [5]="x86_64-pc-linux-gnu")
declare -ir EUID="1000"
declare -ir PPID="2983"
declare -r SHELLOPTS="braceexpand:hashall:interactive-comments"
declare -ir UID="1000"
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

```
24  # To see all read only variables
25  readonly -p
26
27  # To see all read only functions
28  readonly -f
```

27. Signals & Trap

- To kill the process
  - kill -9 pid
- '$$' to print pid of the script itself
- to check about different signals
- man 7 signal
- simicolon is used to combine two commands

$_ 27_signal_and_trap.sh

```
#! /bin/bash

trap "echo Exit command is detected" 0

echo "hello word"

exit 0

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$ ./27_signal_and_trap.sh
hello word
Exit command is detected
rashid@rashid-pc:~/5_BashScripting/Bash_Practice$
```

```

# to remove the file when a specific signal found
file_name=/home/rashid/5_BashScripting/Bash_Practice/6_rashid.txt

# 0: exit signal; 2: cntrl+c signal; 15: sigterm signal
# this will remove the file when either of these signals are found
trap = "rm -f $file_name; exit" 0 2 15
```

- to see trap that you have defined
- type trap
- to remove trap
  - trap - name_of signal/number_of_signal e.g. trap – 0 2 15

28. Debug a bash script

- bash -x ./script_name.sh OR
- #! /*bin/*bash -x (in the script) OR
- type set -x (in the script), it will start debugging from the point where you define this
- type set +x (in the script), it will deactivatedebugging from the point where you define this

•