To install Ubantu :
Download ubantu from micro store
turn Windows features on or off

tick linux
restart after installation
ID : Kiran_512
Password : SHIVRAY

open ubantu terminal and waitinf or the installation and then set username and the password

Commads :
ls -a : to check all files
ls -l - IT will give the long list with the permission sets
ls -S - it will give us the reverse od

IMPORTANT COMMANDS :

cd ~ => to goto root directory!
cd / => to go to home directory!

mkdir - create file
rmdir - remove file
pwd - present working directory
ls /mnt/ - to check the drive on device -- works from any locations not the space ls_space/mnt
clear - used to clear screen
cd file_name- used to change the directory
cd .. - to get back one folder
cd Desktop/ - to go to desktop
touch file_name.ext - to create the file
nano file_name.txt - to open file in nano terminal
cat text.txt - to display the cointent inside the text file on the terminal

date
date +%a/b/x/y/m/A/B/X/Y/M to get the specific - date +23, date +2022, date +45
cal
cal [[month] year] - cal 3 2025

copy command
cp <source file name> <destination file name>

remove directory

Remove directory
rmdir foleder_name

in case opf nested folder insdie the folder then we can use below command
In this case the command will first delet the nexted folders and then it will delete the

rm -rf folder_name

man cal

cmd [- option] [arguments]
here cmd is compulsory and the option and the arguments are not compulsory.

cat > file_name.ext = to add content inside the file // to read and write content
cat file_name.ext = to view the content // to read the content
cat >> file_name.ext = used to append the file

to count the wors and lines in file

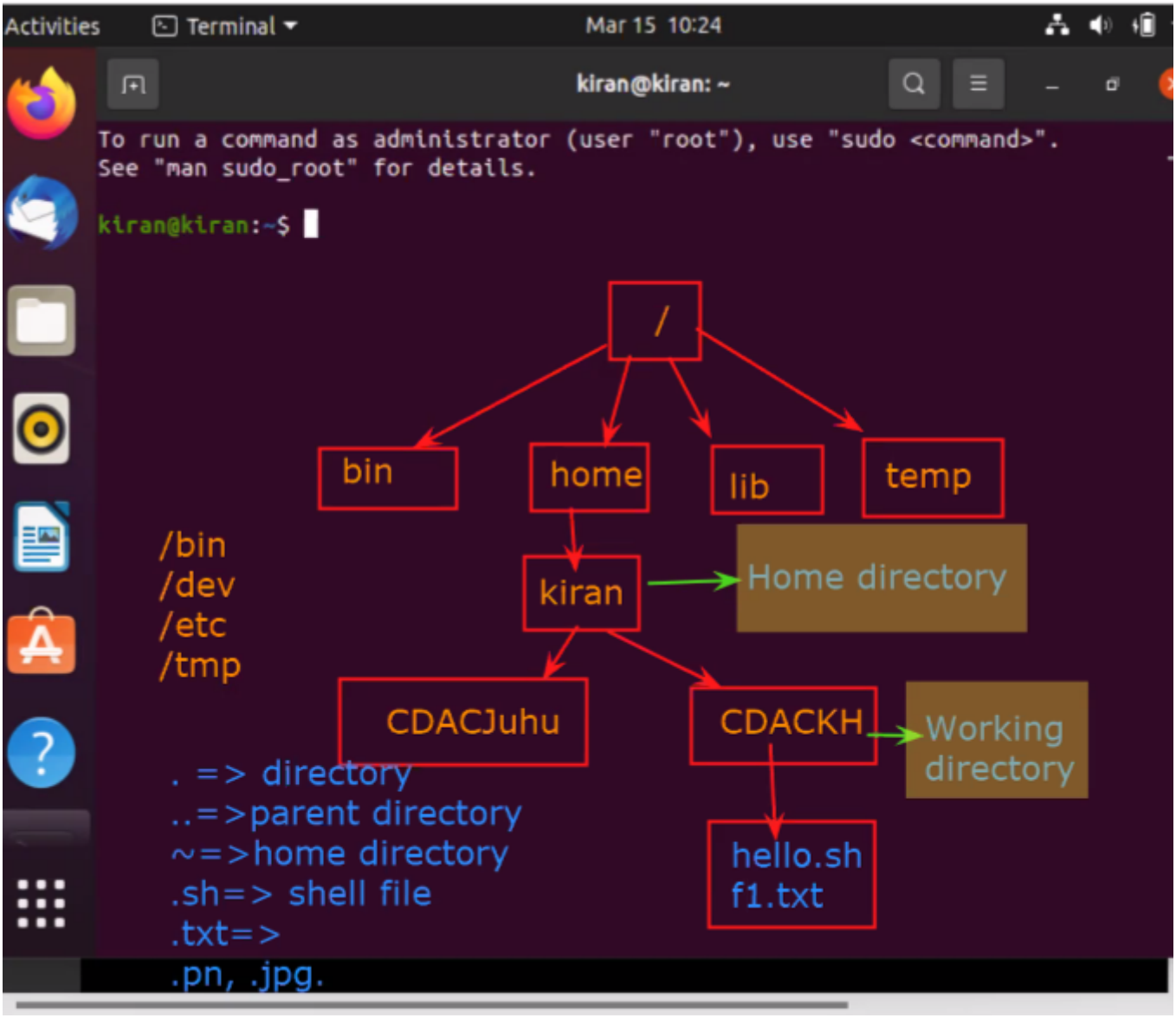
wc file_name.ext
wc -l file_name.ext
wc -c file_name.ext

touch file_name - used to create the files
e.g. touch f1 f2 f3 f4
ls f* - used to list the file starting with the f

. - directory
.. parent directory
~ - home directory
.sh - shell file
.txt -
.pn -
.jpg -

access directories with commadn - cd /dire_name
/bin
/dev
/etc
/tmp

default directory is : Home directory
Working directory



Shell scripting:
Vi Editors: VIM Editor

1.vim file_name.sh
2.press escape and press i button
3.add code in the file
4.Esc + :wq
w- save the file and q is to exit from here
5.chmod +x text.sh - to grant the permissio to access the file for run
6. = > ./text.c - to execute the file OR bash text.sh
7.

now it will give the o/p

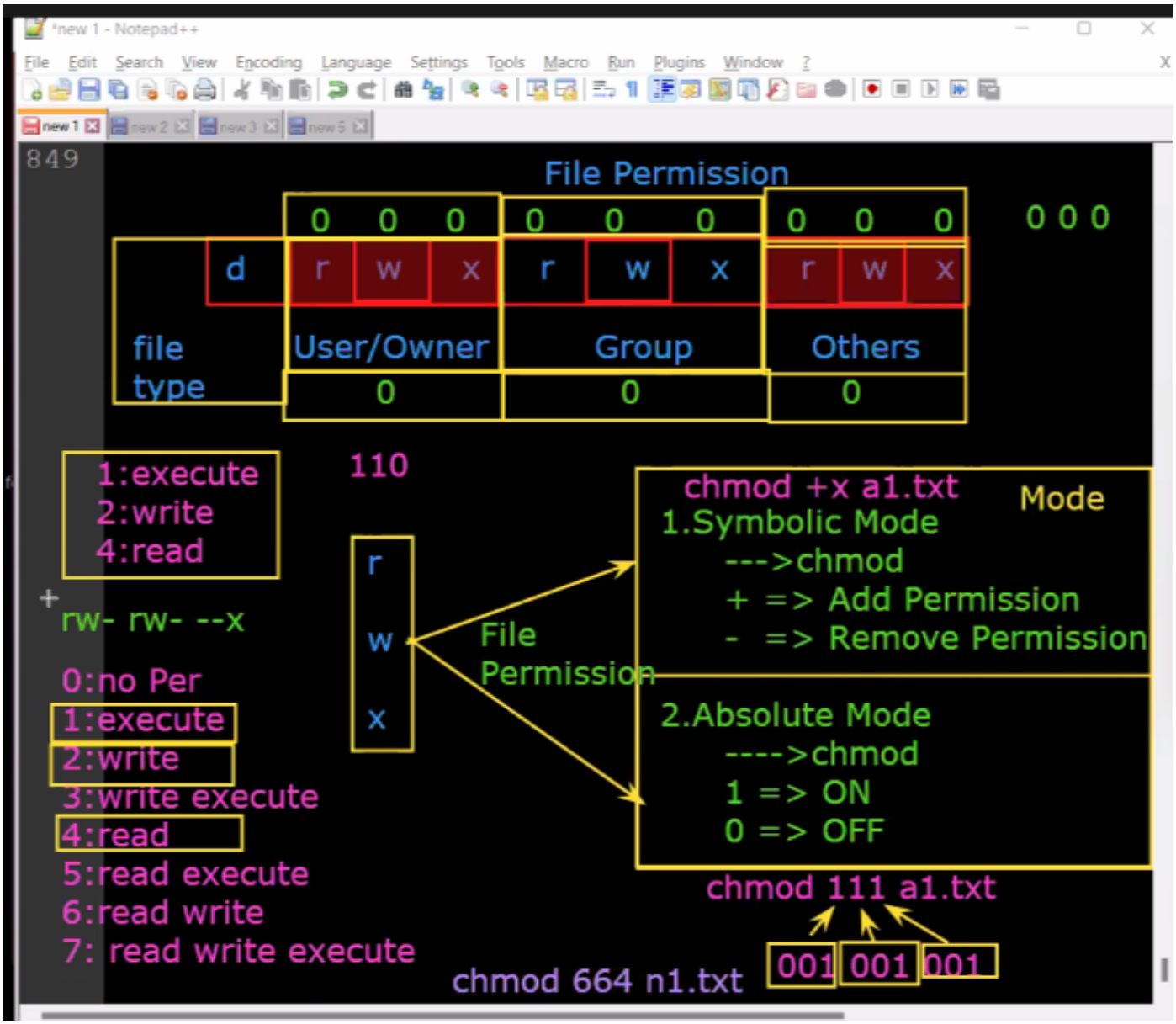
File access rights:
3 types of access rights:

- Read
- Write
- Execute

3 types of access levels:

- User
- Group
- Others

drwerwerwe
here if the first letter is empty then it's file or else it would be the directory
after first block another next 3 blocks give the info about the permission of user and next three about the group and the last three about the others

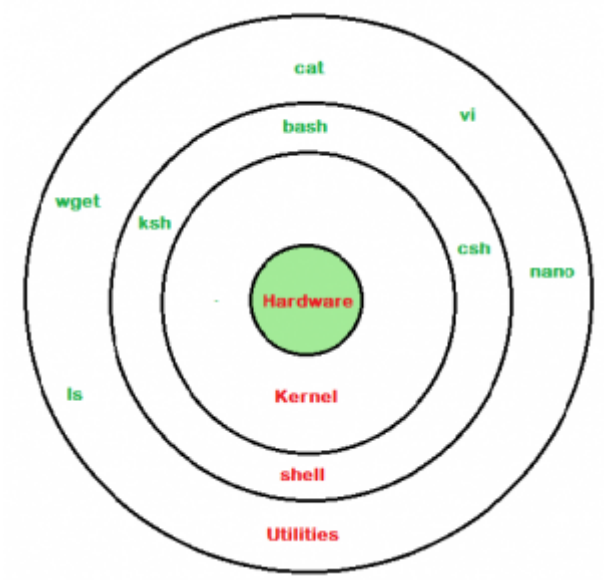


chmod u-x g+r o-e text.sh

16/3/2022

Shell Programming

- interface between the user and the kernel
- why we do it? Reusability,



- SHELL
  - A shell is special user program which provide an interface to user to use operating system services.
  - Shell scripts are interpreted and not compiled!
  - case sensitive lang

echo $SHELL - used to check the shell that we are using

echo "statement" - used to print

#!/bin/sh - sharp hashbang / shilabang helps to execute the program in bash shell

chmod +x test.sh - command used to give permission for the execution of the program to all groups

$1-$9 - positional parameters
${10} - ${n} - positional parameters
$0 - name of the script

eg ./hello.sh mark tom bam
$0 hello.sh
$1 mark

args =("$@")

echo ${args[0]} // this will print mark which is the 0th item in array now and nopt the name of the script/file

echo $@ - to print all the arguments

echo $# - size of array

That is the only diff between $0-9 and ("@")

$* - All the arguments as a string
$@ : same as above only diff is need to focus on "" which is must part here

$# Total number of arguments passed on command promt

$$ Process ID of the script

$? LAST RETURN CODE

"" - it will print the statement as it is
'' - same as above
`` - same as above

#comments in shell programming

HW : accepts the values from the user and display the calculation
1.Area of circle
2.Perimeter of rectangle
3.are of cube

```
if[true_condition]
then
    statements;
fi
```

```
if[true_condition]
then
    statements;
else
    statements;
fi
```

```
if[Condition(true)]
then
    statenments;
elif[Condition(true)]
```

```
elif[Condition(true)]
then
    statements;
elif[Condition(true)]
then
    statements;
else
    statements;
fi
```

Operators:

- Arithmatic
  - +,-,/,*,%

```
sum= `expr $a + $b`
res= `expr $a * $b`
res= `expr $a * $b`
res= `expr $a * $b`
```

- Relational
  - == or -eq
- [$a == $b]
- [$a -eq $b]
  - != or -ne
- `expr [ $a -ne $b ]`
-
  - > or -gt
  - < or -lt
- Boolean
  - And (&& or -a)
  - or (|| or -o)
  - Not equal (!)
- Bitwise
  - AND (&)
  - OR (|)
  - XOR (^)
  - Complement (~)
- File test
  - -b,-c,-d ....

expression command: `expr : 1+5`

```
sum= `expr $a + $b`
```

Switch case:

```
case "$Key" in
pattern1)

        statement;
        statement;
        statement;
        ;; # used for break();
pattern1)
        statement;
        statement;
        ;;
esac
```

LOOPS in Shell Scripting :

```
for i in 1 2 3 4 5
do
        statements;
done
```

OR

```
for ((i=o; i<=5; i++ ))
do
        statements;
done
```

```
while [ condition true ]
do
        statement;
done
```

```
until [ condition_false ]
do
        statements;
done
```

Shell Scripting: by Edureka Unasana

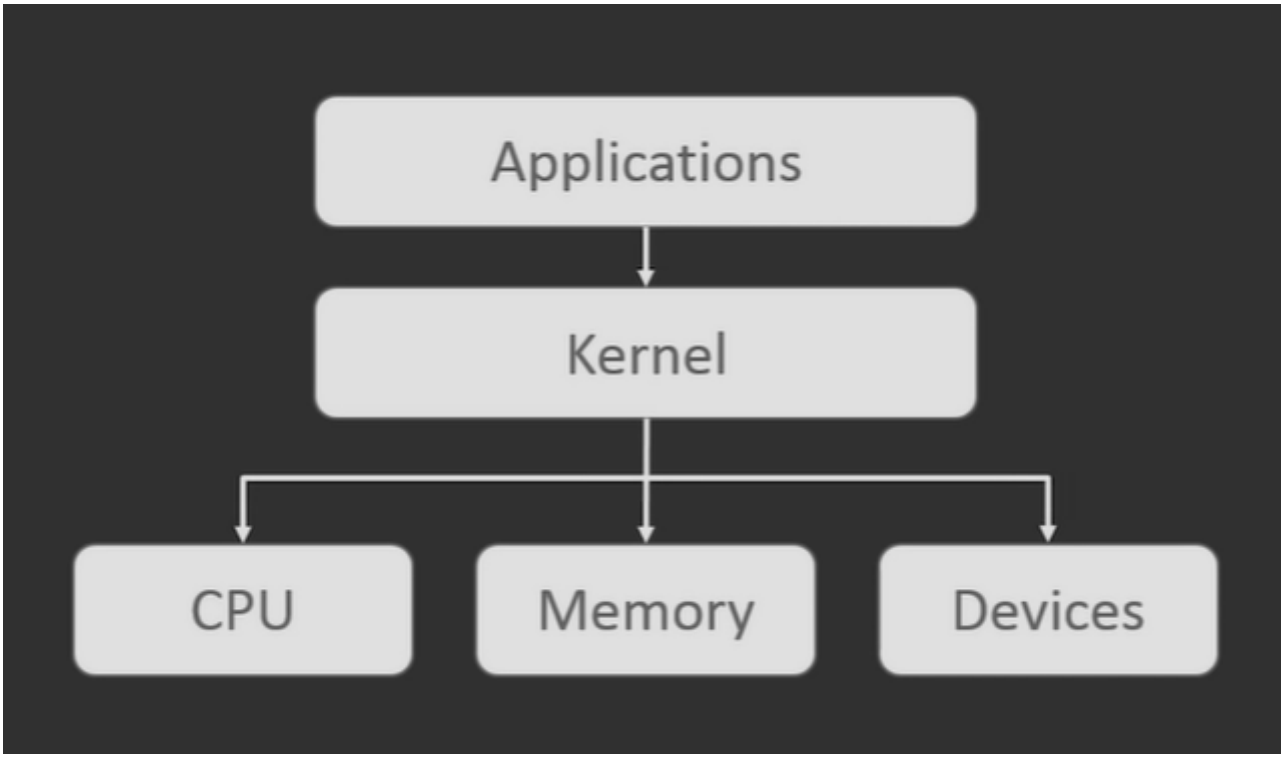
Shell Scripting: by Edureka Upasana

Agenda :
Getting started with the Linux:
Commadn Line Arguments:
Shell Scripting Basics:
Using Variables:
Basic operators:
Shell Loops:
Shell Functions:

Why Linux :
Open source OS - SOURCE CODE IS AVILABLE TO EVRYONE And it can be customised as per the needs
access to source code :
highly secured :
runs faster : if something is deleted it gets deleted from the entire computer systrem unlike windows system thats how it is fast

LINUX :
Community develkoped OS which is capable of handeling activitied from multiple users at the same time.

Kernel :
The comp program that allocate the reources and co ordinates all the details of the computer's internal is called as OS or the kernal. User communicate with the OS thru the programs called as shell!



Path where the the files stored for llnux on windows: C:\Users\Kiran\AppData\Local\Packages

/mnt/c/cdac/shell/Edureka$

to use nano editors:
kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/Edureka$ nano cliessentials.txt

cliessentials.txt - is the file name
#then type inside the file whatever our program is and then ctrl + O to save it and Ctrl + X to exit to the terminal.
kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/Edureka$

Read here about the root directory -- https://www.howtogeek.com/117435/htg-explains-the-linux-directory-structure-explained/

To copy file from one dir to another:
cp file_name.ext path(mnt/f/programs)
in above eg the files will be copied to the programs folder in f drive

In the same way mv command can be used

cat command will display the file content = > cat file_name.ext

cat file1.txt > file2.xyz -- cat is used in this way to copy the content from one file to another

less file_name.txt => will open the content in the new window in much better way and to exit from the new window one has to press q

mv --help \ grep verbose

head -number // number is the line numberthat wants as an output

tail -number // number is the no of line which we want to remove from the op

`kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/edureka$ mv --help | grep verbose
-v, --verbose explain what is being done
kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/edureka$`
Basically this command will help to check the status of the movement in case of big file

TOUCH is used to create the multiple files in the same directory!
kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/edureka$ touch file.txt file1.txt file2.txt
kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/edureka$ ls
cliessentials.txt file.txt file1.txt file2.txt

CLI : Command line interface is basically used for the interact with the softwares and OS BY TYPING commands into the interface and receives a response in a same way

Advantages :

- SAVES A MEMORY
- better control
- can be scalled up

Shell:

- Command line interpreter, it converts the commands entered by the use and converts them into a language that is understood byu the kernel.

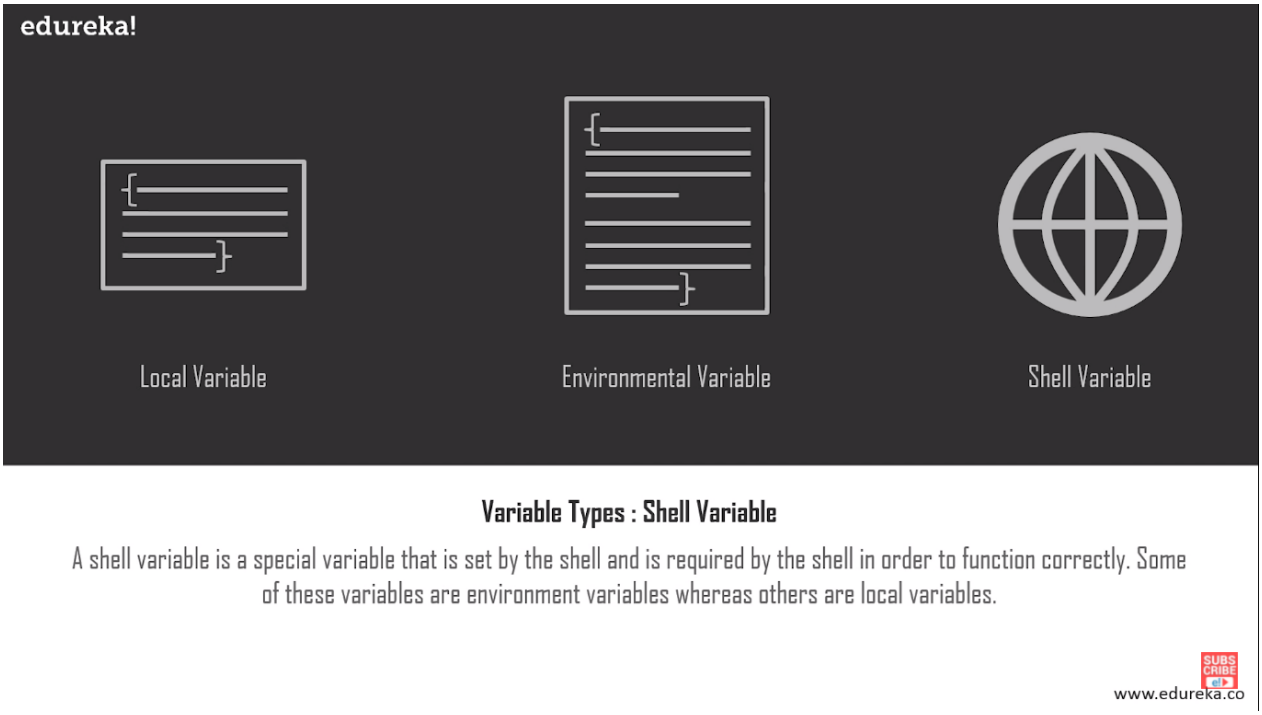
Bourne Shell Type:

- Bourne shell
- Korn shell
- Bourne again shell
- POSIX shell

C Shell Types:

- C shell
- TENEX/TOPS C Shell
- Z Shel

Variables:



Define Variables:

Special Variables :

CLA

Special parameters
Exit

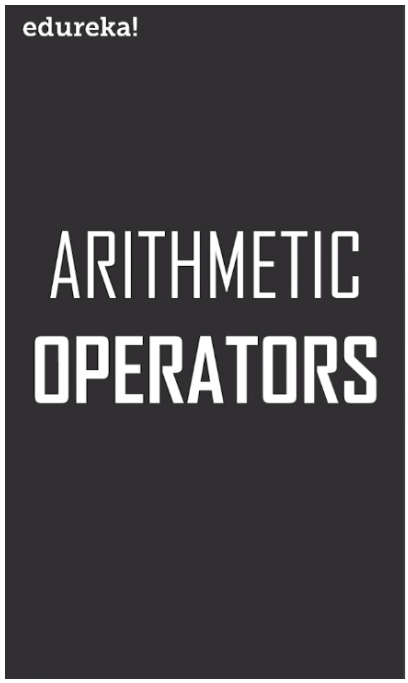
Define Variables:
Scaler Variables : holds only one value at a time
Name= "Kiran"
echo "$Name"

use of readonly
Name= "Kiran"
readonly Name // this variable is same like a const
Name= "Rahul"
echo "$Name"

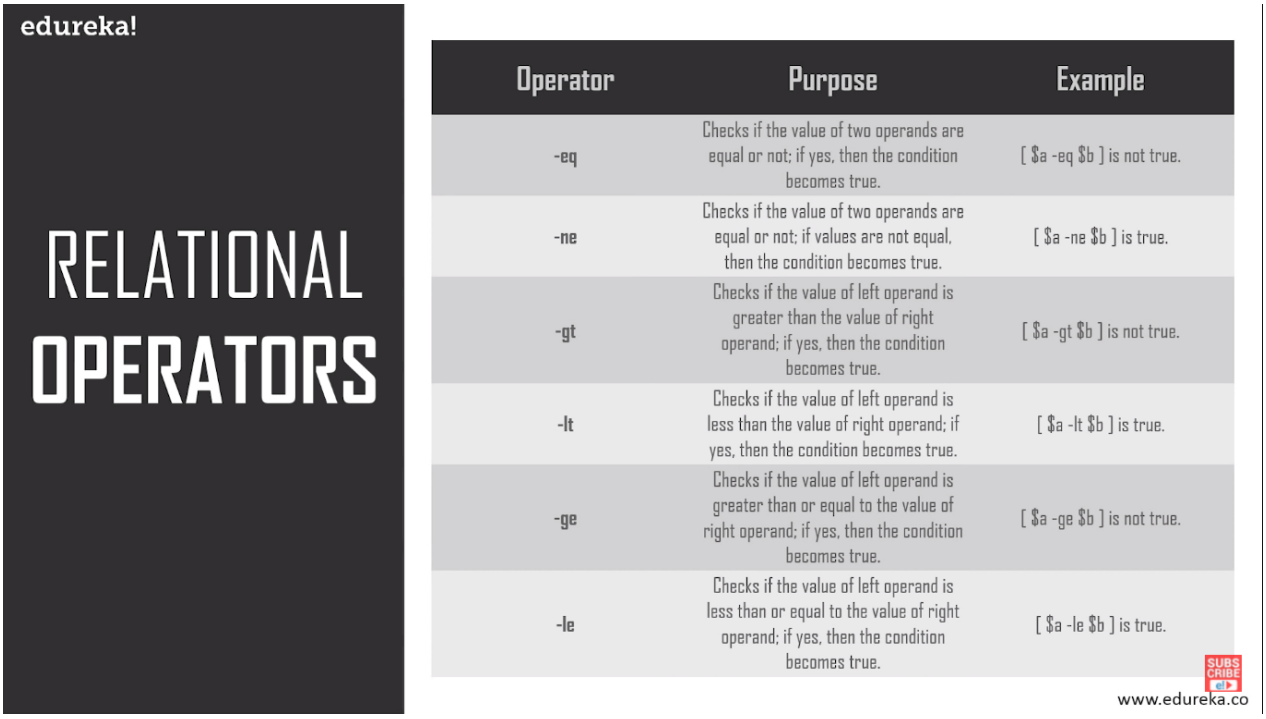
Name= "Kiran"
unset Name // this will remove the value of Name varibale
echo "$Name"

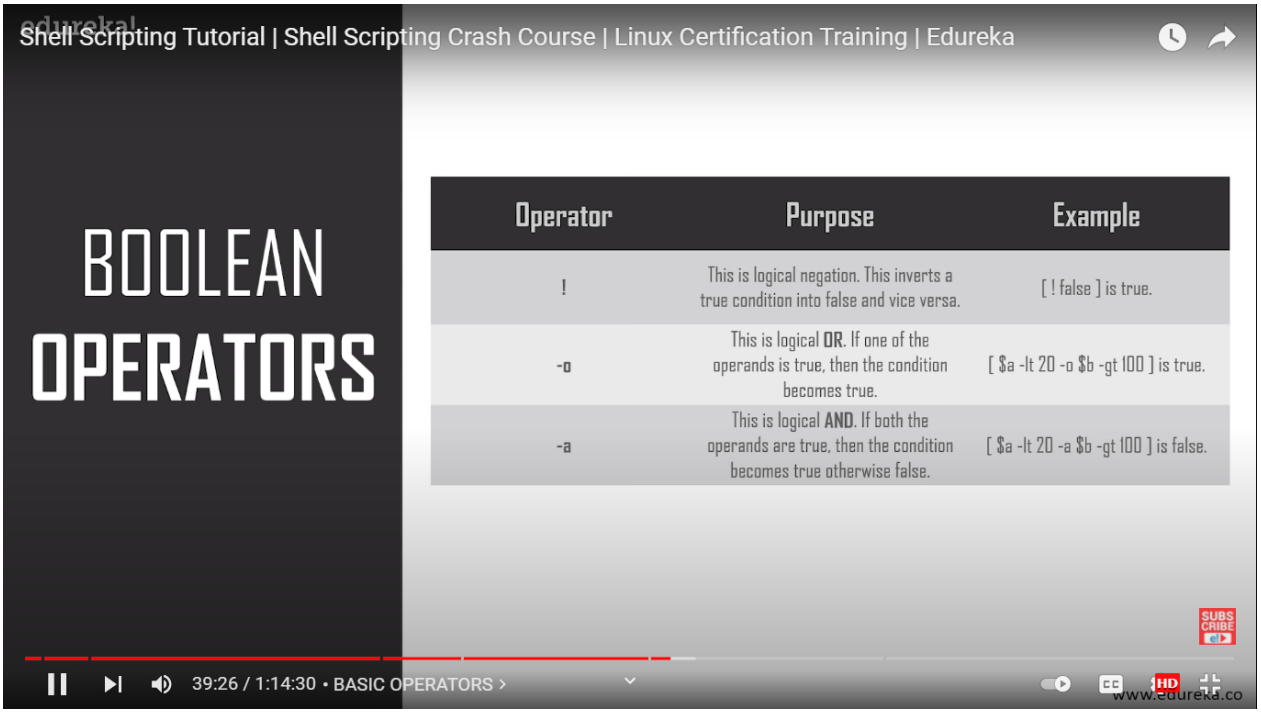
Special Variables :
$0 = file_name of the script
$1....9 = any n nuber which is positive decimal number //
$# / no of arguments applied in the script
$* // all the arguments which are double quoted
$@ // all the arguments which are indivdually double quoted
$? exit status of the last executed command // numerical value returend for the last executed command if its successful thenit return 0 or else it will return 1
$$ process ID under which the command is excuted

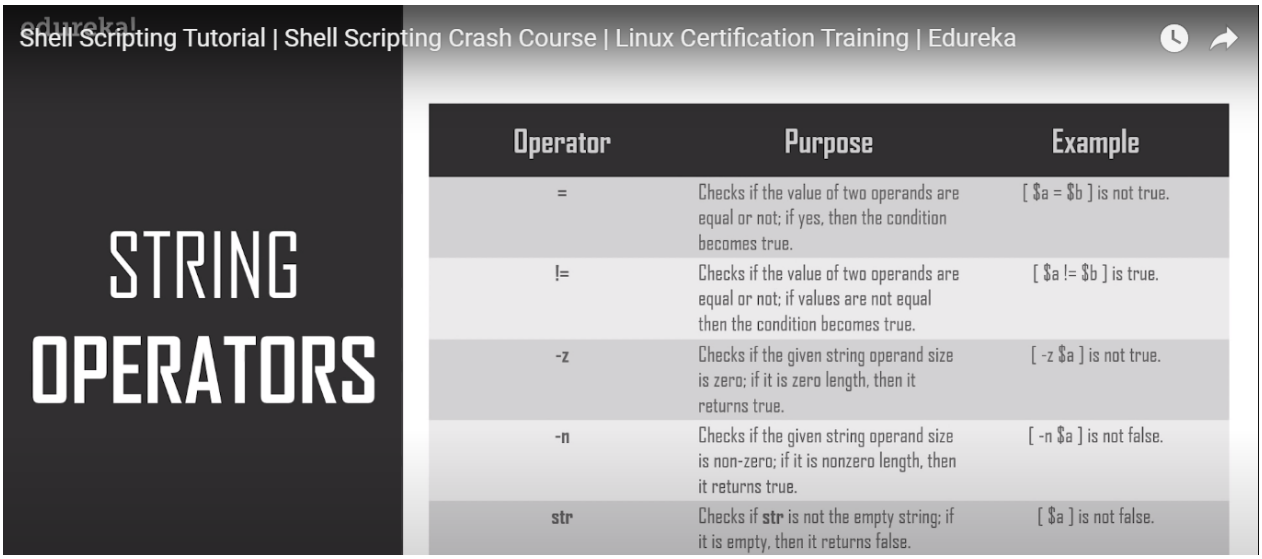
Operators:

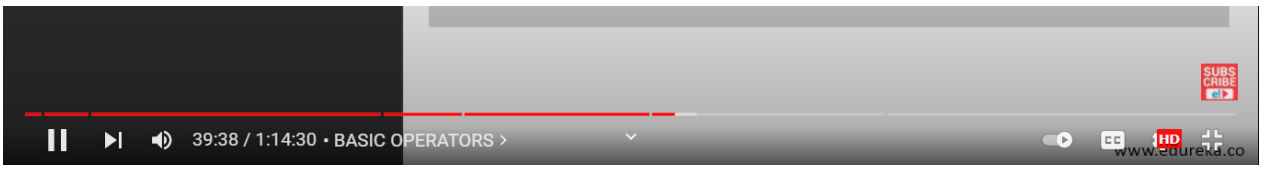


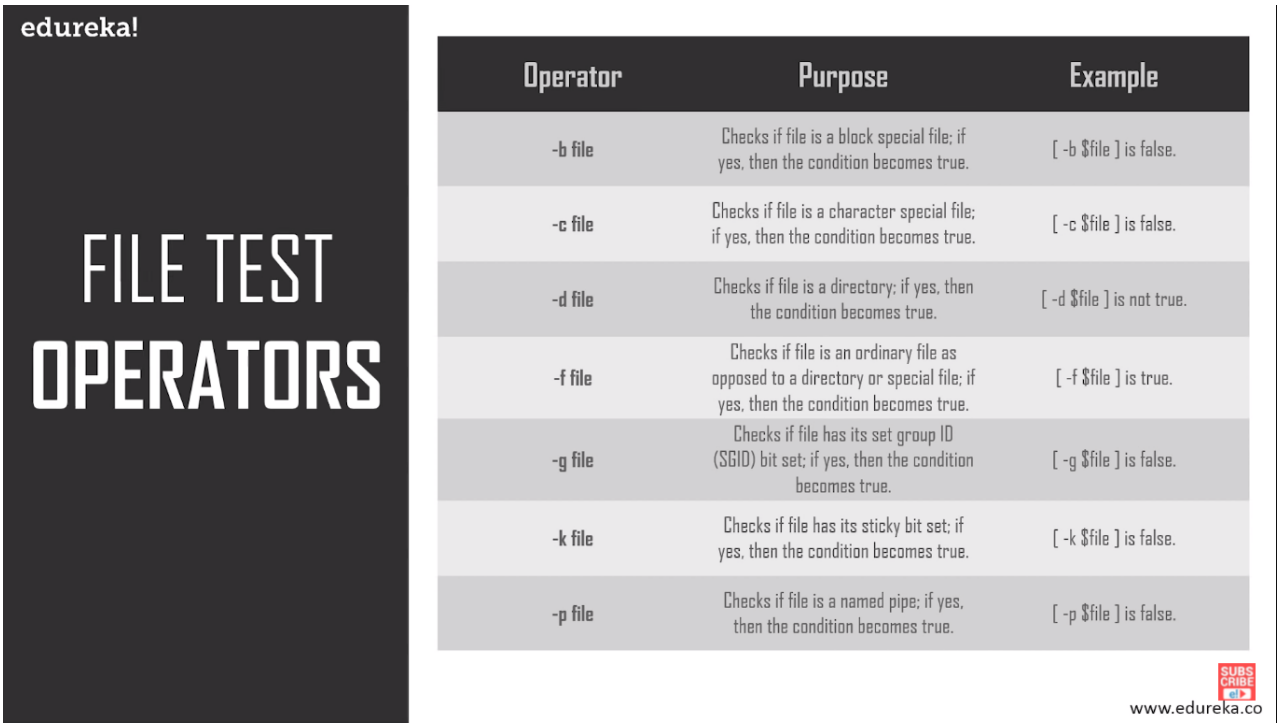
Operator	Purpose	Example
+ (Addition)	Adds values on either side of the operator	`expr $a + $b` will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/ (Division)	Divides left hand operand by right hand operand	`expr $b / $a` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a
== (Equality)	Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!= (Not Equality)	Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

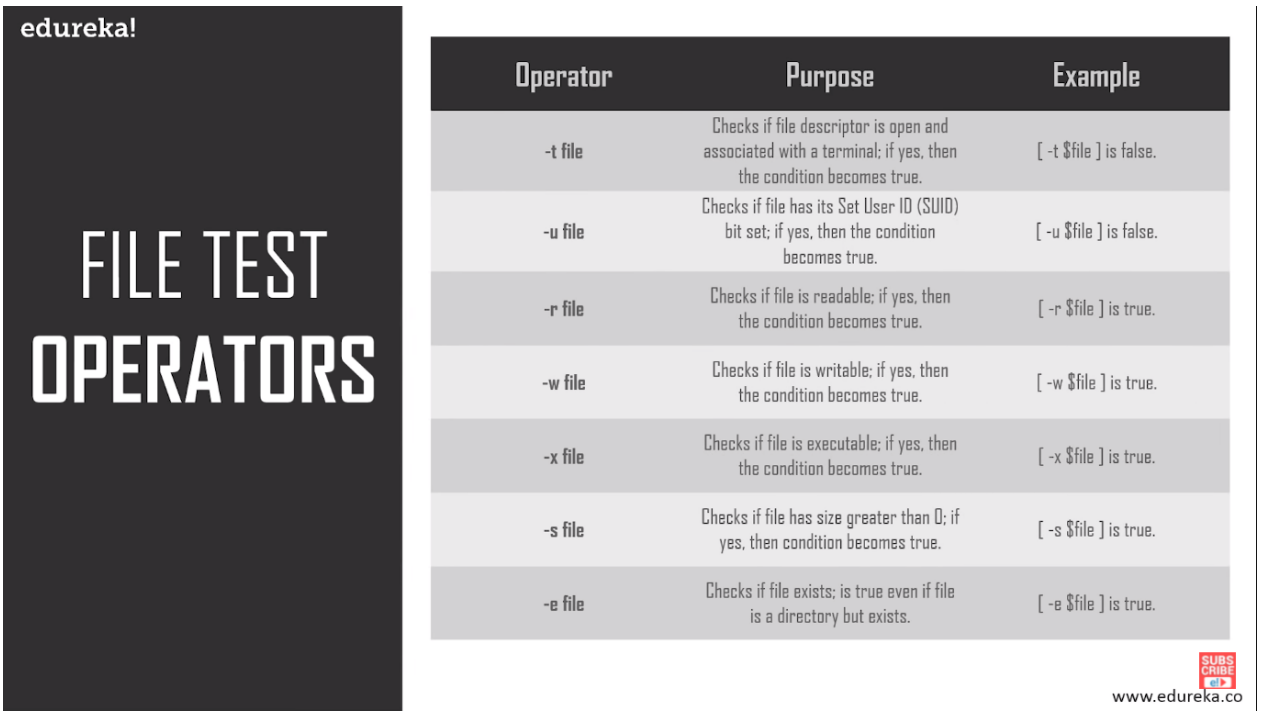

edureka!
RELATIONAL
OPERATORS
Operator
Purpose
Example
-eq
Checks if the value of two operands are equal or not; if yes, then the condition becomes true.
[$a -eq $b] is not true.
-ne
Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.
[$a -ne $b] is true.
-gt
Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.
[$a -gt $b] is not true.
-lt
Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.
[$a -lt $b] is true.
-ge
Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.
[$a -ge $b] is not true.
-le
Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.
[$a -le $b] is true.
www.edureka.co


Shell Scripting Tutorial | Shell Scripting Crash Course | Linux Certification Training | Edureka
BOOLEAN
OPERATORS
Operator
Purpose
Example
!
This is logical negation. This inverts a true condition into false and vice versa.
[! false] is true.
-o
This is logical OR. If one of the operands is true, then the condition becomes true.
[$a -lt 20 -o $b -gt 100] is true.
-a
This is logical AND. If both the operands are true, then the condition becomes true otherwise false.
[$a -lt 20 -a $b -gt 100] is false.
39:26 / 1:14:30 • BASIC OPERATORS ›
www.edureka.co


Shell Scripting Tutorial | Shell Scripting Crash Course | Linux Certification Training | Edureka
STRING
OPERATORS
Operator
Purpose
Example
=
Checks if the value of two operands are equal or not; if yes, then the condition becomes true.
[$a = $b] is not true.
!=
Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true.
[$a != $b] is true.
-z
Checks if the given string operand size is zero; if it is zero length, then it returns true.
[-z $a] is not true.
-n
Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true.
[-n $a] is not false.
str
Checks if str is not the empty string; if it is empty, then it returns false.
[$a] is not false.







Shell Loops:

While Loop

```
while command
do
    Statement;
done
```

For loop

```
for var in w1 w2 w3 ... wN
do
      Satatement;
done
```

Until Loop

Nested Loop

Loop Control

CLA
Special parameters
Exit

SS by PL

Shell: Interpreter
Shell commands are not compiled but interprestyed by the shell

To check the shells supported by our system : cat /etc/shells

kiran_512@LAPTOP-TG43J350:~$ cat /etc/shells (Note the space between cat {sapce} /)
/etc/shells: valid login shells
/bin/sh
/bin/bash
/bin/rbash
/bin/dash
/usr/bin/tmux
/usr/bin/screen

sh - first and original shell!

bash is is the imporved version of sh

bash - born again shell! standard GNU shell!

to check the shell location use = which bash
kiran_512@LAPTOP-TG43J350:~$ which bash

/bin/bash

file name - hello.sh
here sh is given when open file in editor so editor will understamd that its shell script file

kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ touch hello.sh
kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ code .
Installing VS Code Server for x64 (c722ca6c7eed3d7987c0d5c3df5c45f6b15e77d1)
Downloading: 100%
Unpacking: 100%
Unpacked 1619 files and folders to /home/kiran_512/.vscode-server/bin/c722ca6c7eed3d7987c0d5c3df5c45f6b15e77d1.
kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$

```
#! /bin/bash
# this is a comment
echo "Hello Word" # comments
# variables that stores the data
# System Variables :
# User defined variable :
echo Our shell name is $BASH
echo shell version is $BASH_VERSION
echo current working dir is $PWD
echo home directory is $HOME
name=kiran #sapce must not be there between variable and values
echo "My name is $name"
#kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./hello.sh
#Hello Word
#Our shell name is /bin/bash
#shell version is 4.4.20(1)-release
#current working dir is /mnt/c/cdac/shell/PL
#home directory is /home/kiran_512
#My name is kiran
```

read command use:

```
#! /bin/bash
echo enter names
read name1 name2 name3
echo Names: $name1 $name2 $name3
read name
echo Names: $name
echo $name3
#kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
#enter names
#KIRAN AKSHAY PRANIT
```

```
#Names: KIRAN AKSHAY PRANIT
#KIRAN AKSHAY PRANIT
#Names: KIRAN AKSHAY PRANIT
#PRANIT
```

Default read variable is REPLY

```
#! /bin/bash
echo Enter your name
read
echo Your name is $REPLY
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# Enter your name
# kIRAN yADAV
# Your name is kIRAN yADAV
```

print the o/p on the same line: and to make the password silent i.e. invisibile use flag -sp

```
#! /bin/bash

read -p 'username :' username
echo $username
read -sp 'Password : ' Password
echo $Password

kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
username :kiran
kiran
Password : welcome
kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$
```

multiple input save inside an array " flag used is -a

```
#! /bin/bash
echo Enter the names
read -a names
echo Array_elements ${names[0]} ${names[2]

# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# Enter the names
# Reading is good habit
# Array_elements Reading good
```

Passing arguments on command line : $0 prints the fiole name

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

```
echo $0 $1 $2 $3

# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh arg1 arg2 arg3
# ./basics.sh arg1 arg2 arg3
```

Arguments array with the variale

syntax : variablename=("$@")

```
#! /bin/bash
args=("$@")
echo ${args[0]} ${args[2]} ${args[3]}

# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh arg1 arg2 arg3
# arg1 arg3
```

NOTE: So the only difference in above two arrays method is in first method 0th index is assigned to the first arguments i.e. the file name and in case of second method the 0th index is assigned to the first argument excluding the file name

No of args checked with the $# and above second array can also be printed with the command echo $@

Another method to display array is

Array=($(cat))

```
#! /bin/bash
args=("$@")
echo $#
echo $@
```

If esle in bash script:

if [condition]
then
    statement
else
    statement
fi

Relational Operators :
-eq

-ne
-gt
-ge
-lt
-le
To use below operator expression has to be insdie the double poaranthesis (($a > $b))
<
>
<=
>=
String Operators:
= is euqal to - if ["$a " = "$b"]
== is euqal to - if ["$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 it has zero lenght

EXAMPLES :

```
#! /bin/bash
echo  Enter string
read name
if [ $name = "Kiran" ]
then
     echo "Welcome home Mr. $name"
fi
echo Enter palce
read loc
if [[ "$loc" > "K" ]]
then
echo "Whoa! $loc"
fi
```

CAT EXIT : Ctrl + D

File Test Operators :
-e flag used to check whether file exist or not in if conditon as if [-e $filename]

-f flag used to check if file exits and if its regular file or not if [-f $filename]

-d flag used to check if dir exits or not if [-d $direname]

-d flag used to check if dir exits or not if [-d $direname]

two types of files blog special file and character special file so accordngly tow flags are used -b AND -c

-s : flag used to check if the files is empty or not

cat > file_name enter
type the content in file
prees cntrl + D o get out from the command

read, write and execute permision check on command promt: {flags used are -r -w -x}

```
#! /bin/bash
echo -e "Enter the file name : \c"
read file
if [ -r $file ]
then
    echo "read perission"
else
    echo "no read permission"
fi
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# Enter the file name : test
# read perission
```

cat command use :

cat file.ext - this will display the content
cat > file.ext - this will allow to add the content and existing content will be overridden
cat >> file.ext - thsi will allow to append the content

In above code just change r to w and e : to perform write and exaecute

```
# echo -e "Enter the file name : \c"
# read file
# if [ -s $file ]
# then
#     echo "not empty"
# else
#     echo "file is empty"
# fi
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# Enter the file name : test
# file is empty
# echo -e "Enter the file name : \c"
# read file
# if [ -d $file ]
# then
#     echo "dir found"
```

```
# else
#     echo "Not found"
# fi
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# Enter the file name : dir
# dir found
# echo -e "Enter the file name : \c"
# read file
# if [ -f $file ]
# then
#     echo "file found"
# else
#     echo "Not found"
# fi
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# Enter the file name : file
# file found
# echo -e "Enter the file name : \c"
# read file
# if [ -e $file ]
# then
#     echo "file found"
# else
#     echo "Not found"
# fi
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# Enter the file name : file
# file found
```

Code to appends the O/P

Note : `cat >> file_name` command used to append the data

```
#! /bin/bash
echo -e "Enter the file name : \c"
read file
if [ -f $file ]
then
    if [ -w $file ]
    then
        echo "Write your code and press Ctrl + D to exit"
        cat >> $file
    else
        echo "File do not have write permission"
    fi
else
```

```
    echo "$file : file not exist"
fi
```

Logical AND operators: ====> && Vs -a flag Three ways given below

Below are the three ways AND operator can be used :

```
#! /bin/bash
echo "Enter your age"
read age
if [ $age -gt 18 ] && [ $age -lt 30 ]
then
    echo "Valid age"
else
    echo "Invalid"
fi
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# Enter your age
# 35
# Invalid
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# Enter your age
# 20
# Valid age
```

```
#! /bin/bash
echo "Enter your age"
read age
if [ $age -gt 18 -a $age -lt 30 ]
then
    echo "Valid age"
else
    echo "Invalid"
fi
```

```
#! /bin/bash
echo "Enter your age"
read age
if [[ $age -gt 18 && $age -lt 30 ]]
then
    echo "Valid age"
else
    echo "Invalid"
fi
```

Logical OR Operator : ==> || or -o

```
#! /bin/bash
echo "Enter your age"
read age
if [[ $age -gt 18 || $age -gt 30 ]]
then
    echo "Valid age"
else
    echo "Invalid"
fi
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# Enter your age
# 35
# Valid age
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# Enter your age
# 17
# Invalid
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$
```

```
#! /bin/bash
echo "Enter your age"
read age
if [ $age -gt 18 -o $age -gt 30 ]
then
    echo "Valid age"
else
    echo "Invalid"
fi
```

```
#! /bin/bash
echo "Enter your age"
read age
if [ $age -gt 18 ] || [ $age -gt 30 ]
then
    echo "Valid age"
else
    echo "Invalid"
fi
```

Arithmatic Operator :

```
#! /bin/bash
a=10
b=5
echo $(( a + b ))
echo $(( a - b ))
echo $(( a / b ))
echo $(( a * b ))
echo $(( a % b ))
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# 15
# 5
# 2
# 50
# 0
```

```
#! /bin/bash
a=10
b=5
echo $( expr  $a + $b )
echo $( expr  $a - $b )
echo $( expr $a / $b )
echo $( expr  $a * $b )
echo $( expr  $a % $b )
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# 15
# 5
# 2
# expr: syntax error
# 0
```

Note : Here with expr word we need to use * for multiplication as $(expr $a * $b)

Decimal Point Calculations :

```
#! /bin/bash
a=10.4
b=5
echo $( expr  $a + $b )
echo $( expr  $a - $b )
echo $( expr $a / $b )
echo $( expr  $a * $b )
echo $( expr  $a % $b )
echo $(( a + b ))
echo $(( a - b ))
```

```
echo $(( a / b ))
echo $(( a * b ))
echo $(( a % b ))

ERROR:
# expr: non-integer argument
# expr: non-integer argument
# expr: non-integer argument
# expr: syntax error
# expr: non-integer argument
# ./basics.sh: line 14: 10.4: syntax error: invalid arithmetic operator (error token
is ".4")
# ./basics.sh: line 15: 10.4: syntax error: invalid arithmetic operator (error token
is ".4")
# ./basics.sh: line 16: 10.4: syntax error: invalid arithmetic operator (error token
is ".4")
# ./basics.sh: line 17: 10.4: syntax error: invalid arithmetic operator (error token
is ".4")
# ./basics.sh: line 18: 10.4: syntax error: invalid arithmetic operator (error token
is ".4")
```

Sqaure root and decimla point calculation in arithmatic operators: man bc

```
#! /bin/bash
a=10.4
b=5
echo "$a + $b" | bc
echo "$a - $b" | bc
echo "$a * $b" | bc
echo "scale=5; $a / $b" | bc
echo "$a % $b" | bc
echo "scale=5; sqrt($a)" | bc -l

# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# 15.4
# 5.4
# 52.0
# 2.08000
# .4
# 3.22490
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$
```

Note : -l (pipe) is ued to call math libarary to perform sqrt ops

#!/bin/bash
read word
value=` echo "$word " | bc -l

value=" echo $word " | bc -l
echo $(printf %.3f $value)

for power calculations

```
echo "scale=5; $b^2" | bc -l
```

//sqrt
kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell$ echo "scale = 2; sqrt(27)" | bc -l

5.19

Alternate way for arithmatic calcualtions
Also we can use

echo ` expr 1 + 2 `

-- op is 3

Switch CASE:

```
case "$Key" in
pattern1)
    statement;
    statement;
    statement;
    ;; # used for break();
pattern1)
    statement;
    statement;
    ;;
esac
```

```
#! /bin/bash

veh=$1
case $veh in
"car" )
    echo "Rent of the $veh is 100 dollar";;
"van" )
    echo "Rent of the $veh is 80 dollar";;
"bicycle" )
    echo "Rent of the $veh is 10 dollar";;
"truck" )
```

```
    echo "Rent of the $veh is 150 dollar";;
* )
echo "Please enter car, van, byclce or truck to know its rent cost";;
esac

# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# Please enter car, van, byclce or truck to know its rent cost
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh van
# Rent of the van is 80 dollar
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$
```

```
#! /bin/bash
echo -e "Enter some character : \c"
read value
case $value in
    [a-z] )
        echo "Enter value is in small case";;
    [A-Z] )
        echo "Enter value is in Upper case";;
    [0-9] )
        echo "Enter value is in the ange of 0-9";;
    ?)
        echo "Enter value is special character";;
    *)
        echo "Unknown letter";;
esac
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh van
# Enter some character : A
# Enter value is in Upper case
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh van
# Enter some character : 5
# Enter value is in the ange of 0-9
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh van
# Enter some character : $
# Enter value is special character
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh van
# Enter some character : a
# Enter value is in small case
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$
```

if even after entring the capital letter lets say K then if the result if small case then user will have to set the LANG=C

Arrays in Scripting:

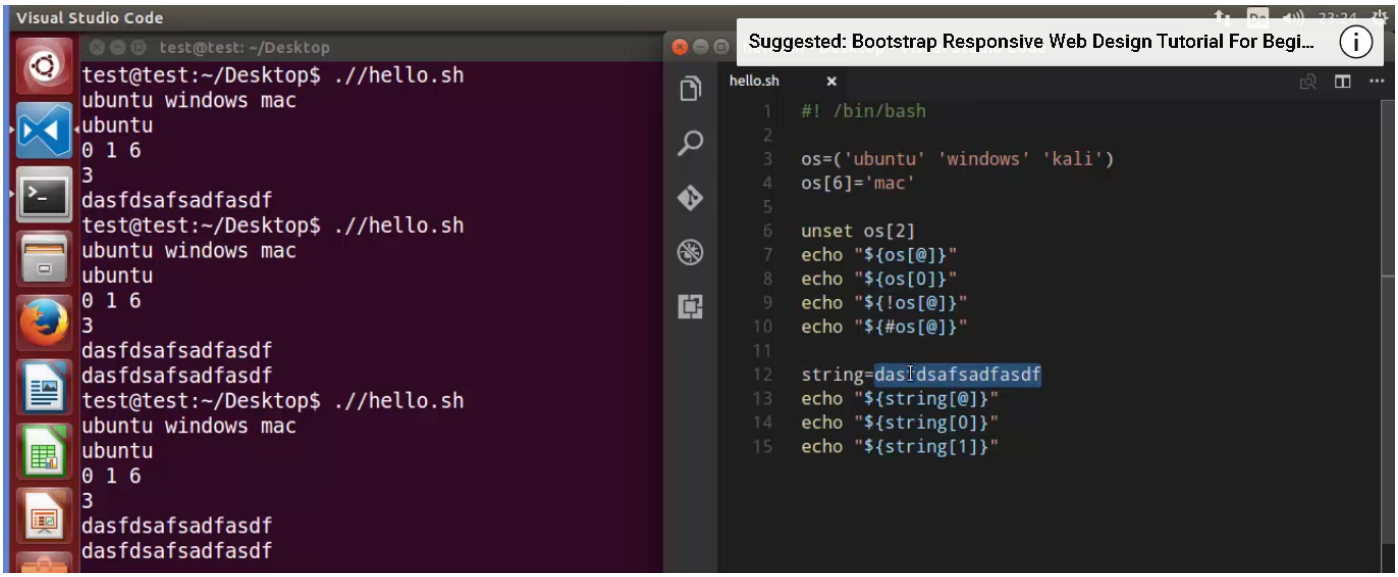
```
#! /bin/bash
os=('Ubantu' 'Windows' 'Kali')
os[3]='mac'  #ADD
os[0]='lINUX' #UPDATE
unset os[2]
os[100] = 'newos' #we can add element at any index in bash, size wont get affected
with this.. size is calculated based on only the no of actual elements
echo ${os[@]} # prints entire array
echo ${os[1]}  # prints element at index 1
echo ${!os[@]}  # prints index of allelements in array
echo ${#os[@]}  #prints size of an array
echo $@


# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh command line
arguiments in array for with index for first word starting from 1
# Ubantu Windows Kali mac
# Windows
# 0 1 2 3
# 4
# command line arguiments in array for with index for first word starting from 1
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$
```

NOTE : TO Print the command line args we use echo $@ and to print the array we use echo ${arrayname[@]}

Command used to remove array element

```
unset os[2]
```

NOTE : For any normal variable even if its not declared as an array it will act as an array and if we apply the array command it will give the resutls and the if var=1 then var[0] value is 1 and var[1] is empty and so on...



While loop:

```
#! /bin/bash
n=1
while (( $n < 11 ))
do
    echo  -e " $n  \c "
    ((n++))
done
```

while loop using sleep and open terminal :

```
#! /bin/bash
n=1
while (( $n < 11 ))
do
    echo  -e " $n  \c "
    $((n+1))
    sleep 1 #1 in seconds so in op each no will be printed with the gap of 1 seconds
done
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
#  1   2   3   4   5   6   7   8   9   10  kiran_512@LAPTOP-
TG43J350:/mnt/c/cdac/shell/PL$
```

here sleep will pause the execution for n sec in above eg its for 1 sec

```
#! /bin/bash
n=1
while (( $n < 11 ))
do
    echo  -e " $n  \c "
    $((n+1))
    gnome-terminal & #OR xterm & => WILL OPEN THREE TERMINALS
done
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
#  1   2   3   4   5   6   7   8   9   10  kiran_512@LAPTOP-
TG43J350:/mnt/c/cdac/shell/PL$
```

Read file content in bash:

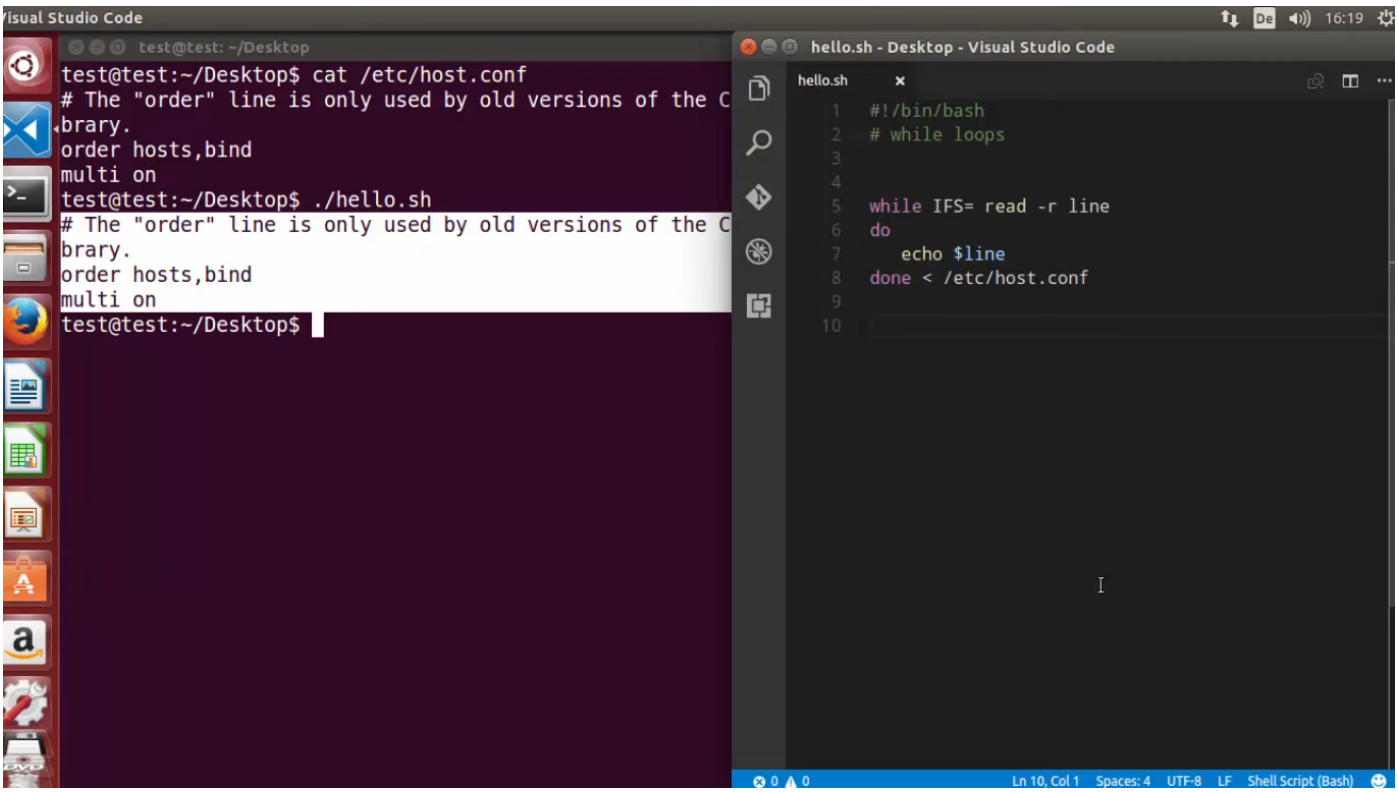
```
#! /bin/bash
while read p
do
    echo $p
done < hello.sh
```

file content inside the file is redirected to the while loop and directipon is set using < symbol

```
#! /bin/bash
cat hello.sh | while read p
do
    echo $p
done
```

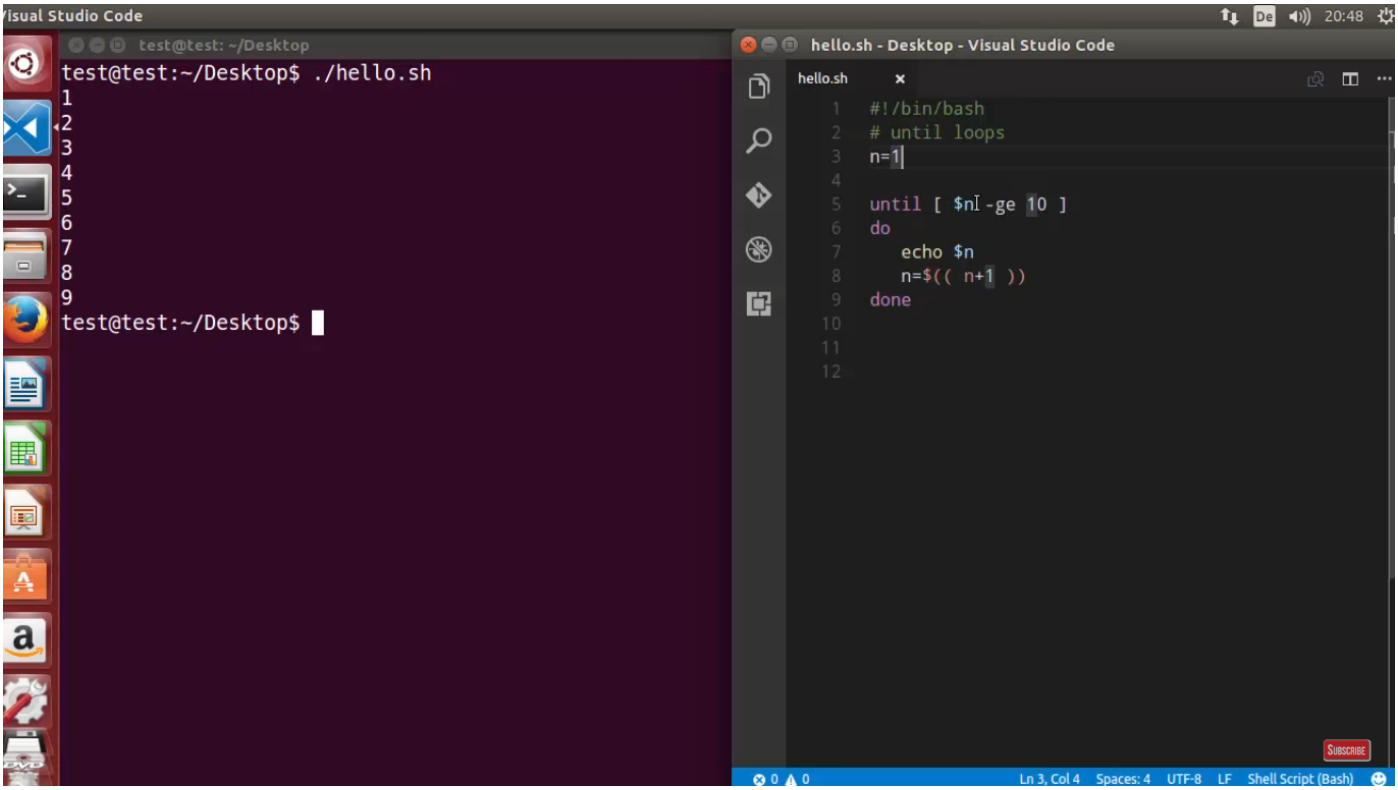
IFS - Internal Fiedl seperator

```
#! /bin/bash
while IFS =' ' read -r line
do
    echo $line
done < hello.sh
```

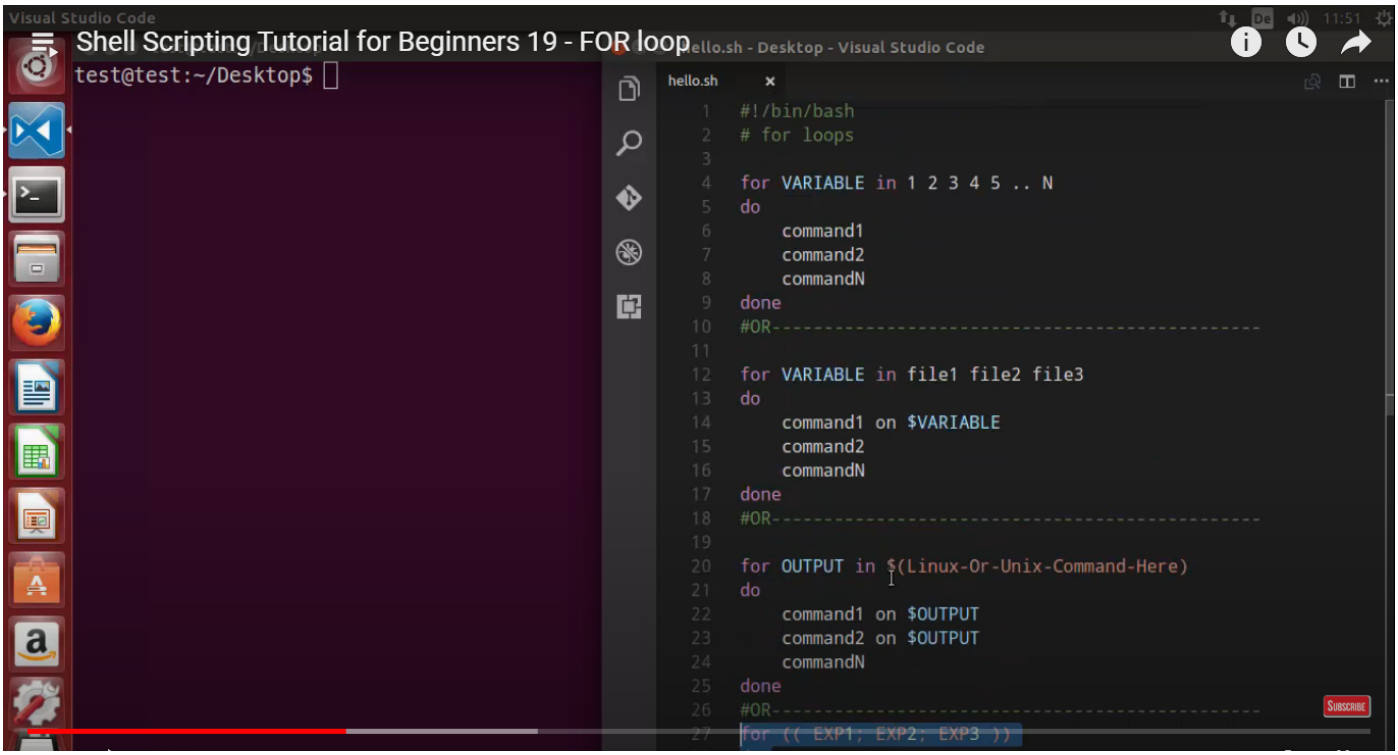


UNTIL Loop:

```
until [ false condition ]
do
      statement
done
```



for loops:



Q. prints 1 to 10 numbers with the increment of 2 in each printed number

```
#! /bin/bash
for i in {1..10..2} #start..end..increment
do
    echo $i
done
```

Q. prints 1 to 10 numbers

```
#! /bin/bash
for i in {1..10} #START..END
do
    echo $i
done
```

```
#! /bin/bash
for i in 1 2 3 4 5
do
    echo $i
done
```

#BASH_VERSION

For loop with the various commands:

```
#! /bin/bash
for command in ls pwd date cal
do
    echo "----------------$command----------------"
    $command
done


# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# ----------------ls----------------
# basics.sh  dir  file  hello.sh  test  test.sh
# ----------------pwd----------------
# /mnt/c/cdac/shell/PL
# ----------------date----------------
# Mon Mar 21 16:00:54 IST 2022
# ----------------cal----------------
```

```
# ----------------cal-----------------
#      March 2022
# Su Mo Tu We Th Fr Sa
#        1  2  3  4  5
#  6  7  8  9 10 11 12
# 13 14 15 16 17 18 19
# 20 21 22 23 24 25 26
# 27 28 29 30 31
```

for loop with the * (asterisk): ITERATE OVER ALL THE FILES IN PRESENT DIRECTORY

```
#! /bin/bash
for item in *
do
    if [ -d $item ]
    then
        echo "$item is direcotry"
    else
        echo "$item is file"
    fi
done
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# basics.sh is file
# dir is direcotry
# file is file
# hello.sh is file
# test is file
# test.sh is file
```

Select Loop :

```
#! /bin/bash
select name in mark tom john ben
do
    case $name in
    mark )
        echo $name is selected;;
    tom )
        echo $name is selected;;
    john )
        echo $name is selected;;
    ben )
        echo $name is selected;;
    *)
        echo Please provide the no between 1 to 4;;
    esac
```

```
done
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# 1) mark
# 2) tom
# 3) john
# 4) ben
# #? mark
# Please provide the no between 1 to 4
# #? 1
# mark is selected
# #?
```

break and continue isused in shell scripting in the similar fashion as in other codes:
NOTE :

;

semicolon in b elow code

```
#! /bin/bash
for ((i = 1; i <= 10; i++)); do
    if [ $i -eq 3 -o $i -eq 6 ];
    then
        continue
    fi
    echo "$i"
done

#! /bin/bash
for ((i = 1; i <= 10; i++)); do
    if [ $i -eq 3 -o $i -eq 6 ];
    then
        continue
    fi
    echo "$i"
done


# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# 1
# 2
# 4
# 5
# 7
# 8
# 9
# 10
```

```
# 10
```

Functions in Shell Scrpting :

```
#! /bin/bash
function Hello(){
    echo "Hello"
}
quit(){
    exit
}
Hello
echo "quit function will exit the code"
quit
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# Hello
# quit function will exit the code
```

```
#! /bin/bash
function print(){
    echo $1 $2 $3
}
quit(){
    exit
}
print kiran akshay pranit
echo "quit function will exit the code"
quit
```

```
#! /bin/bash
function print(){
    echo enter your city
    read city
    name=$1
    echo "the name is $name and city is $city"
}
name=kiran
echo the name is $name
print max
echo "foo"
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# the name is kiran
# enter your city
# kalamboli
# the name is max and city is kalamboli
```

```
# foo
# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$
```

function examples:
Local Variables : SOMETHING DIFEREEEEEEEEEENT than usual in case of Local Variable in Shell scripting

```
#! /bin/bash
function print(){
    name=$1
    echo The name is $name :in function
}
quit(){
    exit
}
name="tom"
echo "The name is $name :before"
print kiran
echo "The name is $name :after"
quit
---
OP is :
The name is Tom :before
The name is karan :in function
The name is karan : after
```

To make it local only see below code : USE OF LOCAL keyword

```
#! /bin/bash
function print(){
    local name=$1
    echo The name is $name :in function
}
quit(){
    exit
}
name="tom"
echo "The name is $name :before"
print kiran
echo "The name is $name :after"
quit
---
OP is :
The name is Tom :before
The name is karan :in function
The name is Tom : after
```

= = > Ternary Operator in shell scripting :

[[10 > 5]] && return 1 || return 0

-- if the condition is true then return 1 or else return 0

```
#! /bin/bash

usage(){


echo "Please enter the file name"
echo "usage $0 file_name"
}




isPresent(){



local file=$1
[[ -f "$file" ]] && return 0 || return 1


}



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



if ( isPresent "$1" )
then
        echo "file found"
else
        echo "file Not found"
fi
```

Readyonly variable and function :

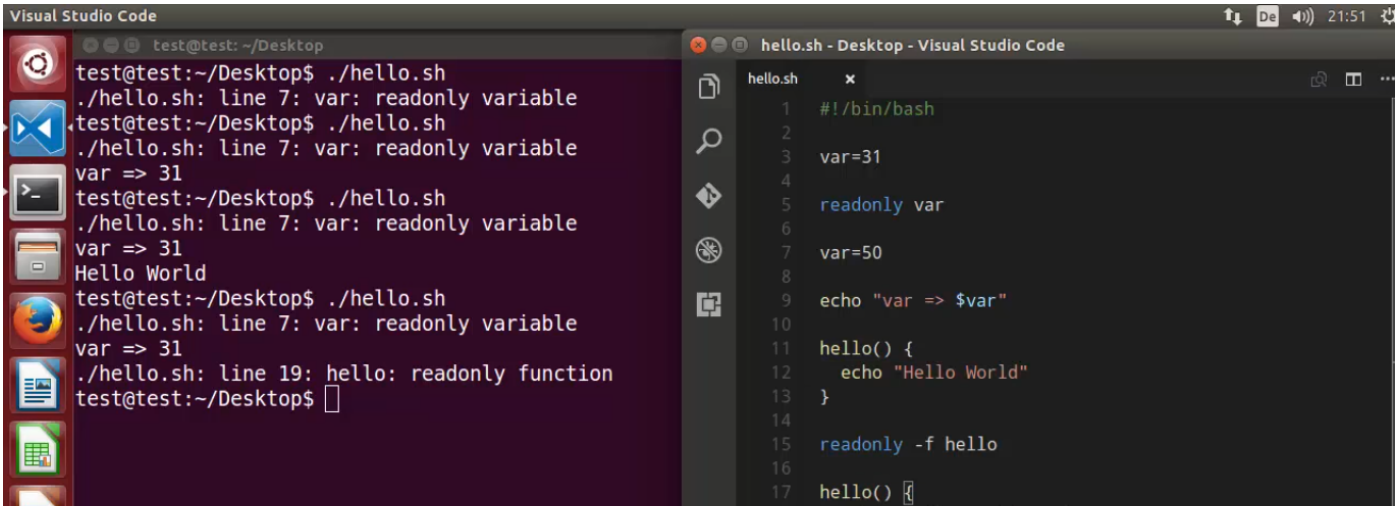
readonly variables :

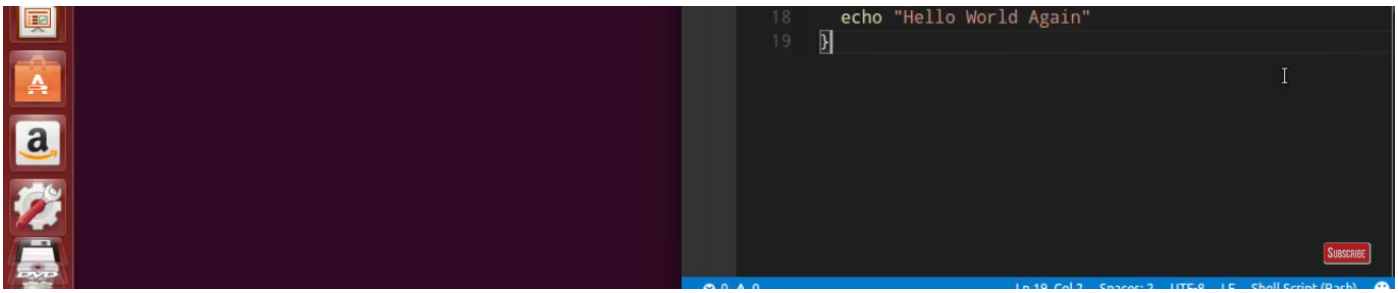
```
#! /bin/bash
var=30
readonly var
var=50
echo $var

#./basics.sh: line 5: var: readonly variable
#30
```

readonly functions :

```
#! /bin/bash
hello(){
    echo "Hello"
}
readonly -f hello
readonly -f


# kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/PL$ ./basics.sh
# hello ()
# {
#     echo "Hello"
# }
# declare -fr hello
```

We can not override readonly functions





---

to check readonly built in var and functions

Use below command :

readonly

readonly -p

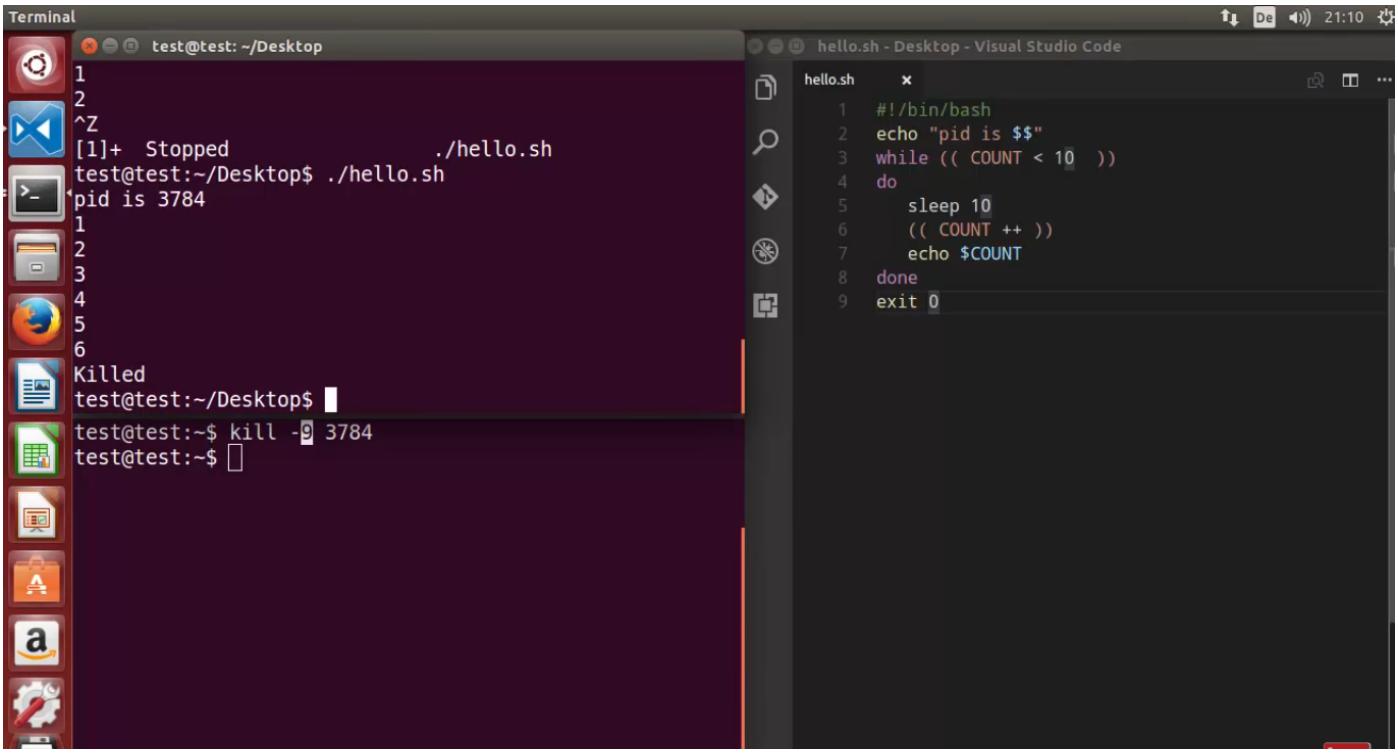
readonly - f

Signals and trap

pid is $$

cntrl + Z is to stop the process
sig kill command : kill -9 PID

Cntrl + C on command promt during code execution is interuupt signal => siginit command



MAN 7 SIGNAL COMMAND

Signal Value Action Comment

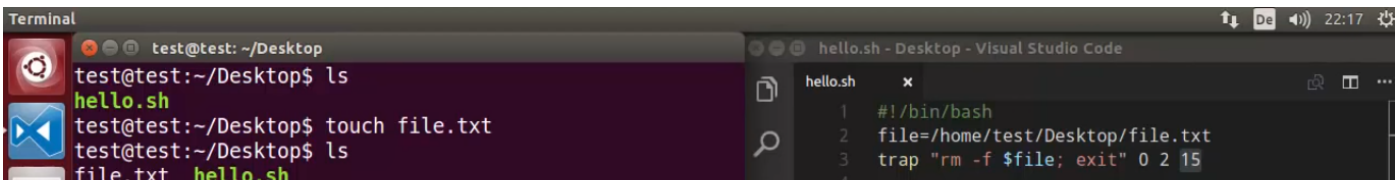
SIGHUP 1 Term Hangup detected on controlling terminal
or death of controlling process
SIGINT 2 Term Interrupt from keyboard
SIGQUIT 3 Core Quit from keyboard
SIGILL 4 Core Illegal Instruction
SIGABRT 6 Core Abort signal from abort(3)
SIGFPE 8 Core Floating-point exception
SIGKILL 9 Term Kill signal
SIGSEGV 11 Core Invalid memory reference
SIGPIPE 13 Term Broken pipe: write to pipe with no
readers; see pipe(7)
SIGALRM 14 Term Timer signal from alarm(2)
SIGTERM 15 Term Termination signal
SIGUSR1 30,10,16 Term User-defined signal 1
SIGUSR2 31,12,17 Term User-defined signal 2
SIGCHLD 20,17,18 Ign Child stopped or terminated
SIGCONT 19,18,25 Cont Continue if stopped
SIGSTOP 17,19,23 Stop Stop process
SIGTSTP 18,20,24 Stop Stop typed at terminal
SIGTTIN 21,21,26 Stop Terminal input for background process
SIGTTOU 22,22,27 Stop Terminal output for background process

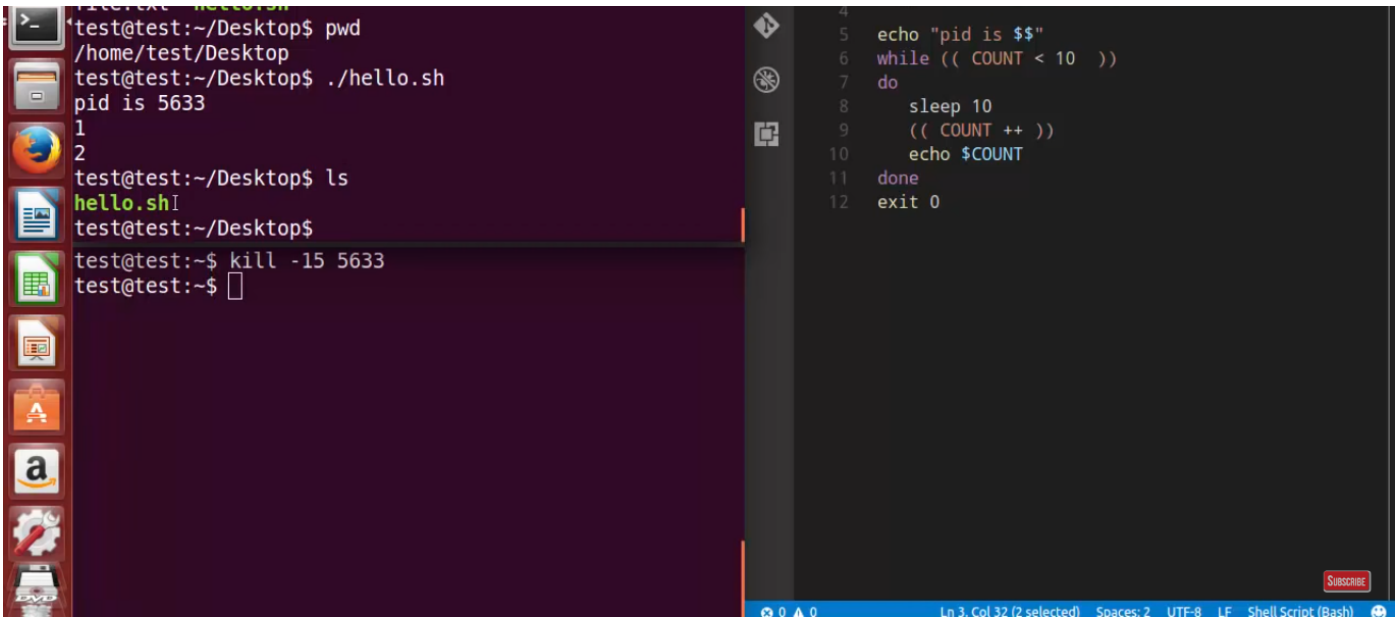
The signals SIGKILL and SIGSTOP cannot be caught, blocked, or ignored.

trap can not catch sigkill and sigstop command

trap "some signal is detected" 0 2

File is removed with the some interruption with the help of trap in below code :





---

bash -x ./filename.sh

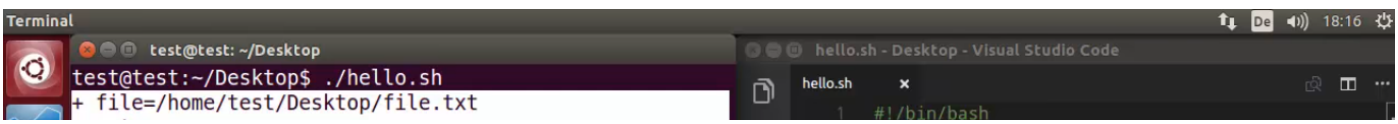
```
kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/Files$ bash -x ./file.sh
+ [[ 0 -eq 0 ]]
+ usage
+ echo 'Please enter the file name'
Please enter the file name
+ echo 'usage ./file.sh file_name'
usage ./file.sh file_name
+ isPresent ''
+ local file=
+ [[ -f '' ]]
+ return 1
+ echo 'file Not found'
file Not found
kiran_512@LAPTOP-TG43J350:/mnt/c/cdac/shell/Files$
```

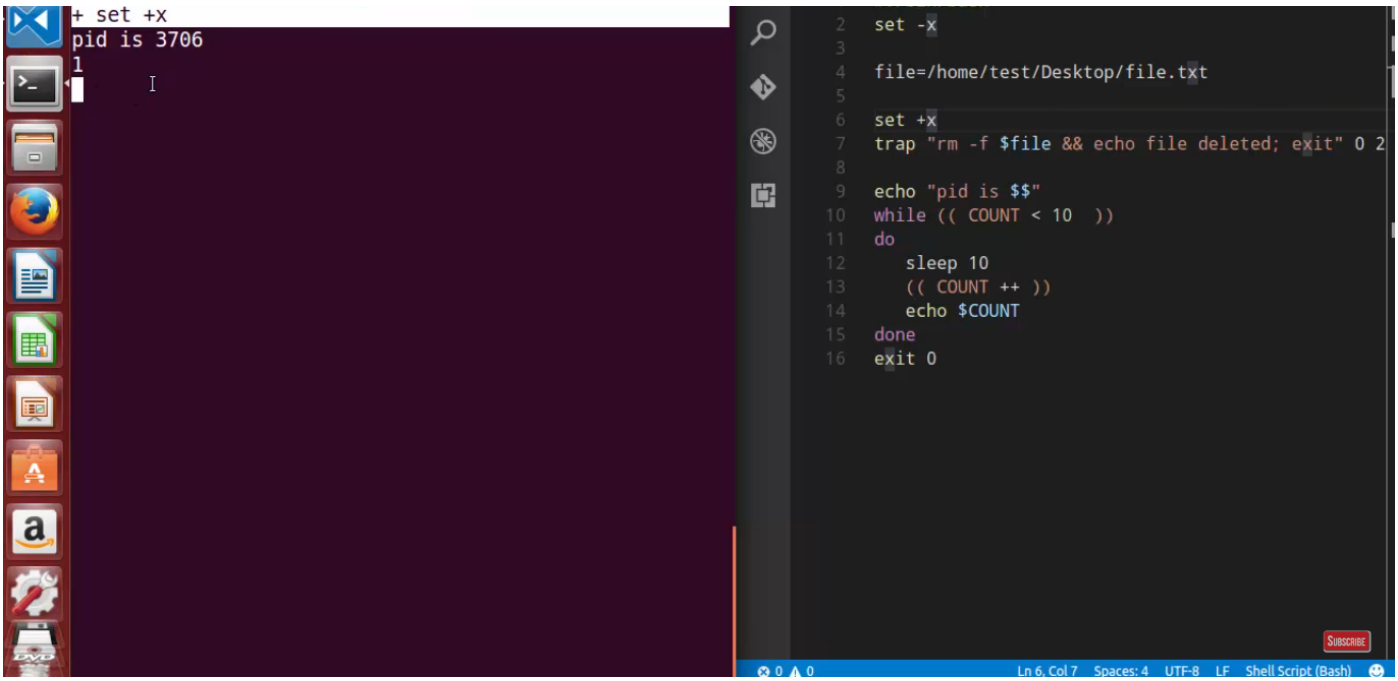
---

also we can use -x in the first line

#! /bin/bash -x

Third o[ptions :





#! /bin/bash
read
arr=($(cat))
echo "${arr[@]}" | tr ' ' '\n' |sort | uniq -u | tr '\n' ' '

SSH :

- Secure Socket Shell
- TO ACCESS SERVER FROM CLIENT OR VICE A VERSA WE CAN SET UP ssh FOR Secure connection!
- systemctl status sshd