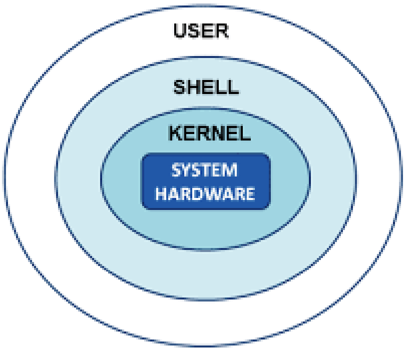
UNIVERSITY OF ENGINEERING & MANAGEMENT, KOLKATA

Course Name : Operating System Laboratory

Shell Programming

What is Shell?

- **A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output.**
- **It is the interface through which a user works on the programs, commands, and scripts.**
- **A shell is accessed by a terminal which runs it.**



Types of Shell

1. The Bourne Shell: The prompt for this shell is $ and its derivatives are listed below:

- POSIX shell also is known as sh
- Korn Shell also knew as ksh
- **B**ourne **A**gain **Sh**ell also knew as bash (most popular)

2.The C shell: The prompt for this shell is %, and its subcategories are:

- C shell also is known as csh
- Tops C shell also is known as tcsh

Steps in creating a Shell Script

- **Create a file using** a **vi** editor(or any other editor). Name script file with **extension .sh**
- **Start** the script with **#! /bin/sh**
- Write some code.
- Save the script file as filename.sh
- For **executing** the script type **bash filename.sh**

"#!" is an operator called shebang which directs the script to the interpreter location. So, if we use"#! /bin/sh" the script gets directed to the bourne-shell.
#!/bin/sh This tells the system that the commands that follow are to be executed by the Bourne shell. It's called a shebang because the
# symbol is called a hash, and the
! symbol is called a bang

Let's create a small script -

```
#!/bin/sh
pwd
ls
```

Adding shell comments

- Commenting is important in any program. In Shell programming, the syntax to add a comment is

```
# Comment
```

```
#!/bin/bash
# Script follows here:
pwd
ls
```

Save the above content and make the script executable –

```
$chmod u+x test.sh
```

The shell script is now ready to be executed –

```
$./test.sh
```

read command

- The following script uses the read command which takes the input from the keyboard and assigns it as the value of the variable PERSON and finally prints it on STDOUT.

```
echo "What is your name?"
read PERSON
echo "Hello, $PERSON"
```

Here is a sample run of the script –

```
$./test.sh
What is your name?
Robert
Hello, Robert
$
```

Variables in Shell

- In Linux (Shell), there are two types of variable:
- **System variables -** Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
- **User defined variables (UDV) -** Created and maintained by user. This type of variable defined in lower letters.

User defined variables (UDV)

- To define UDV use following syntax:

```
variable name=value
$ no=10
```

Rules for Naming variable name

- Variable name must begin with Alphanumeric character or underscore character (_), followed by one or more Alphanumeric character.
- Don't put spaces on either side of the equal sign when assigning value to variable.
- Variables are case-sensitive.
- You can define NULL variable
- Do not use ?,* etc, to name your variable names.

The following examples are valid variable names –

```
_ALI
TOKEN_A
VAR_1
VAR_2
```

The following examples are valid variable names –

```
2_VAR
-VARIABLE
VAR1-VAR2
VAR_A!
```

Print or access valueof UDV

To print or access UDV use following syntax :

$variablename.

Examples:

```
$vech=Bus
$ n=10

$ echo $vech
$ echo $n
```

The expr command is used to evaluate a given expression and display its standard output. Each separated expression is considered as an argument. for integer: addition, subtraction, multiplication, division, and modulus.

For strings: regular expression, set of characters in a string.

Addition: Add two numbers, 15 and 12. `expr 15 + 12`

Multiplication: Multiply 10 and 5. `expr 10 \* 5`

```
a=50
b=70
```

Check if a is equal to b:

```
c=`expr $a = $b`

$c
```

Check if a is less than b:

```
c=`expr $a \< $b`

$c
```

Check if a is not equal to b:

```
c=`expr $a \!= $b`
$c
```

Shell Basic Operators

- **Arithmetic Operators**
- Syntax: expr op1 math-operator op2
- Examples:

```
$ expr 1 + 3
$ expr 2 – 1
$ expr 10 / 2
$ expr 20 % 3
$ expr 10 \* 3
$ echo `expr 6 + 3`
```

- The following example shows how to add two numbers

```
#!/bin/sh
val=`expr 2 + 2`
echo "Total value : $val"
```

There must be spaces between operators and expressions. For example, 2+2 is not correct; it should be written as 2 + 2
The complete expression should be enclosed between ``, called the backtick.

Shell Basic Operators

- **Boolean Operators**
- Assume variable **a** holds 10 and variable **b** holds 20 then -

Operator	Description	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.

Shell Basic Operators

- **Relational Operators**
- Assume variable **a** holds 10 and variable **b** holds 20 then -

Operator	Description	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.

Shell Decision Making

- **Unix Shell -The if…fi**
- **Syntax :**

```
if [ expression ]
then
 Statement(s) to be executed if expression is true
fi
```

- **Unix Shell -The if…else…fi statement**
- **Syntax :**

```
if [ expression ]
then
 Statement(s) to be executed if expression is true
else
 Statement(s) to be executed if expression is not true
fi
```

Shell Decision Making

```
a=10
b=20
if [ $a == $b ]
then
 echo "a is equal to b"
else
 echo "a is not equal to b"
fi
```

- **Unix Shell -The if...elif...fi statement**
- **Syntax :**

```
if [ expression 1 ]
then
 Statement(s) to be executed if expression 1 is true
elif [ expression 2 ]
then
 Statement(s) to be executed if expression 2 is true
else
 Statement(s) to be executed if no expression is true
fi
```

Shell Decision Making

- **The case...esac Statement**
- **Syntax :**

```
case word in
 pattern1)
 Statement(s) to be executed if pattern1 matches
 ;;
 pattern2)
 Statement(s) to be executed if pattern2 matches
 ;;
 pattern3)
 Statement(s) to be executed if pattern3 matches
 ;;
esac
```

Shell Decision Making

- **The case...esac Statement**
- **Example:**

```
FRUIT="kiwi"
case "$FRUIT" in
 "apple") echo "Apple pie is quite tasty."
 ;;
 "banana") echo "I like banana nut bread."
 ;;
 "kiwi") echo "New Zealand is famous for kiwi."
 ;;
esac
```

Command-Line Arguments

- The command-line arguments $1, $2, $3, …$9 are positional parameters, with $0 pointing to the actual command, program, shell script, or function and $1, $2, $3, …$9 as the arguments to the command.
- Following script uses various special variables related to the command line −

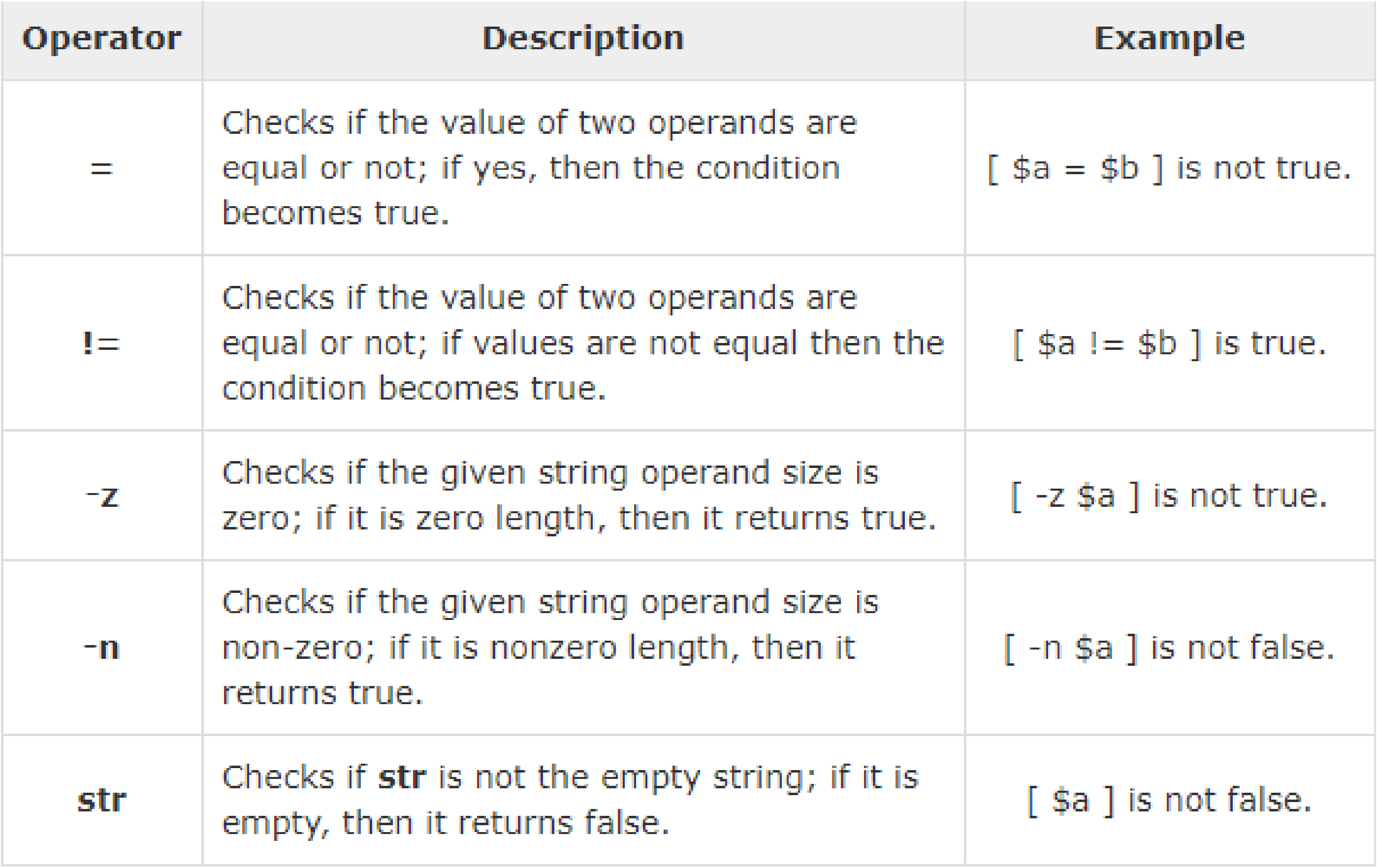
```
#!/bin/sh

echo "File Name: $0"
echo "First Parameter : $1"
echo "Second Parameter : $2"
echo "Quoted Values: $@"
echo "Quoted Values: $*"
echo "Total Number of Parameters : $#"
```

Special Variables

- The following table shows a number of special variables that you can use in your shell scripts –

Variable & Description
$0 The filename of the current script.
$n These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on).
$# The number of arguments supplied to a script.
$* All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2.
$@ All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.
$? The exit status of the last command executed.
$$ The process number of the current shell. For shell scripts, this is the process ID under which they are executing.
$! The process number of the last background command.

String Handling Operators



Operator	Description	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.

String Handling

```
#!/bin/sh
a="abc"
b="efg"
if [ $a = $b ]
then
        echo "$a = $b : a is equal to b"
else
         echo "$a = $b: a is not equal to b“
fi
if [ $a != $b ]
then
        echo "$a != $b : a is not equal to b“
else
         echo "$a != $b: a is equal to b“
fi
if [ -z $a ]
then
         echo "-z $a : string length is zero“
else
        echo "-z $a : string length is not zero“
fi
```

Assignment

1) Write a shell script to calculate addition of two numbers.
2) Write a shell script to show the maximum of three numbers.
3) Write a shell script which displays the result of division of one integer by another integer and informs if the user tries to divide an integer by 0.
4) Write a shell script to find out whether any year input through the keyboard is a leap year or not. If no argument is supplied the current year should be assumed

Assignments

- **Rajesh's basic salary (BASIC) is input through the keyboard. His dearness allowance (DA) is 52% of BASIC. House rent allowance (HRA) is 15% of BASIC. Contributory provident fund is 12% of (BASIC + DA). Write a shell script to calculate his gross salary and take home salary using the following formula:**

  Gross salary = BASIC + DA + HRA   Take home salary = Gross salary - (BASIC + DA) * 0.12

- **Write a shell script that produces a shell calculator to perform the following operations:**
  - **Addition**
  - **Subtraction**
  - **Multiplication**
  - **Division**

Shell Loop Types

- The while loop
- The for loop
- The until loop
- The select loop
- **Unix Shell -The while Loop**
- **Syntax :**

```
while command
do
 Statement(s) to be executed if command is true
done
```

- **Example :**

```
while [ $a -lt 10 ]
do
 echo $a
 a=`expr $a + 1`
done
```

Shell Loop Types

- **Unix Shell -The for Loop**
- **Syntax :**

```
for var in word1 word2 ... word N
do
 Statement(s) to be executed for every word.
done
```

- **Example :**

```
for var in 0 1 2 3 4 5 6 7 8 9
do
 echo $var
done
```

Shell Loop Types

- **Syntax of for like C programming language :**

```
for (( expr1; expr2; expr3 ))
do
 command1
 command2
 ..
done
```

- **Example :**

```
for ((i=1, j=10; i <= 5 ; i++, j=j+5))
do
 echo "Number $i: $j"
done
```

Shell Loop Types

- **Syntax of Unix Shell -The until Loop:**

```
until command
do
 Statement(s) to be executed until command is true
done
```

- **Example :**

```
#!/bin/sh
a=0
until [ ! $a -lt 10 ]
do
 echo $a
 a = 'expr $a + 1'
done
```

Shell Loop Types

- **Syntax of Unix Shell -The select Loop**

```
select var in word1 word2 ... wordN
do
 Statement(s) to be executed for every word.
done
```

Shell Loop Types

- **Syntax of Unix Shell -The select Loop**
- **Example :**

```
#!/bin/ksh
select DRINK in tea cofee water juice appe all none
do
 case $DRINK in
 tea|cofee|water|all)
 echo "Go to canteen"
 ;;
 juice|appe)
 echo "Available at home"
 ;;
 none)
 break
 ;;
 *) echo "ERROR: Invalid selection"
 ;;
 esac
done
```

Shell Loop Control

- The break statement
- The continue statement
- **The break Statement**

  Here is a simple example which shows that loop terminates as soon as a becomes 5:

```
#!/bin/sh
a=0
while [ $a -lt 10 ]
do
 echo $a
 if [ $a -eq 5 ]
 then
 break
 fi
 a=`expr $a + 1`
done
```

Shell Loop Control

- The break statement
- The continue statement
- **The continue statement**

The following loop makes use of the continue statement which returns from the continue statement and starts processing the next statement −

```
NUMS="1 2 3 4 5 6 7"
for NUM in $NUMS
do
 Q='expr $NUM % 2'
 if [ $Q -eq 0 ]
 then
 echo "Number is an even number!!"
 continue
 fi
 echo "Found odd number"
done
```

Arrays

- Arrays provide a method of grouping a set of variables
- Syntax and Example

```
array_name[index]=value  //Array declaration
${array_name[index]} //Accessing an Array
NAME[0]="Zara "
 NAME[1]="Qadir"
NAME[2]="Mahnaz "
 NAME[3]="Ayan"
 NAME[4]="Daisy"
echo "First Index: ${NAME[0]} "
 echo "Second Index: ${NAME[1]}"
```

Assignment

1) **Write a shell script to calculate the sum of digits of any number entered through the keyboard.**
2) **Write a shell script that takes a number from user and prints the reverse of the number.**
3) **Write a shell script to check whether a given number is prime or not.**
4) **Write a shell script to list the names of files under the current directory started with vowels.**

Command Substitution

- Command substitution is the mechanism by which the shell performs a given set of commands and then substitutes their output in the place of the commands

```
DATE=`date`
 echo "Date is $DATE "
 USERS=`who | wc -l`
 echo "Logged in user are $USERS" UP=`date ; uptime` echo "Uptime is $UP "
```

Output:

```
Date is Thu Jul 2 03:59:57 MST 2009
Logged in user are 1
 Uptime is Thu Jul 2 03:59:57 MST 2009
 03:59:57 up 20 days, 14:03, 1 user, load avg: 0.13, 0.07, 0.15
```

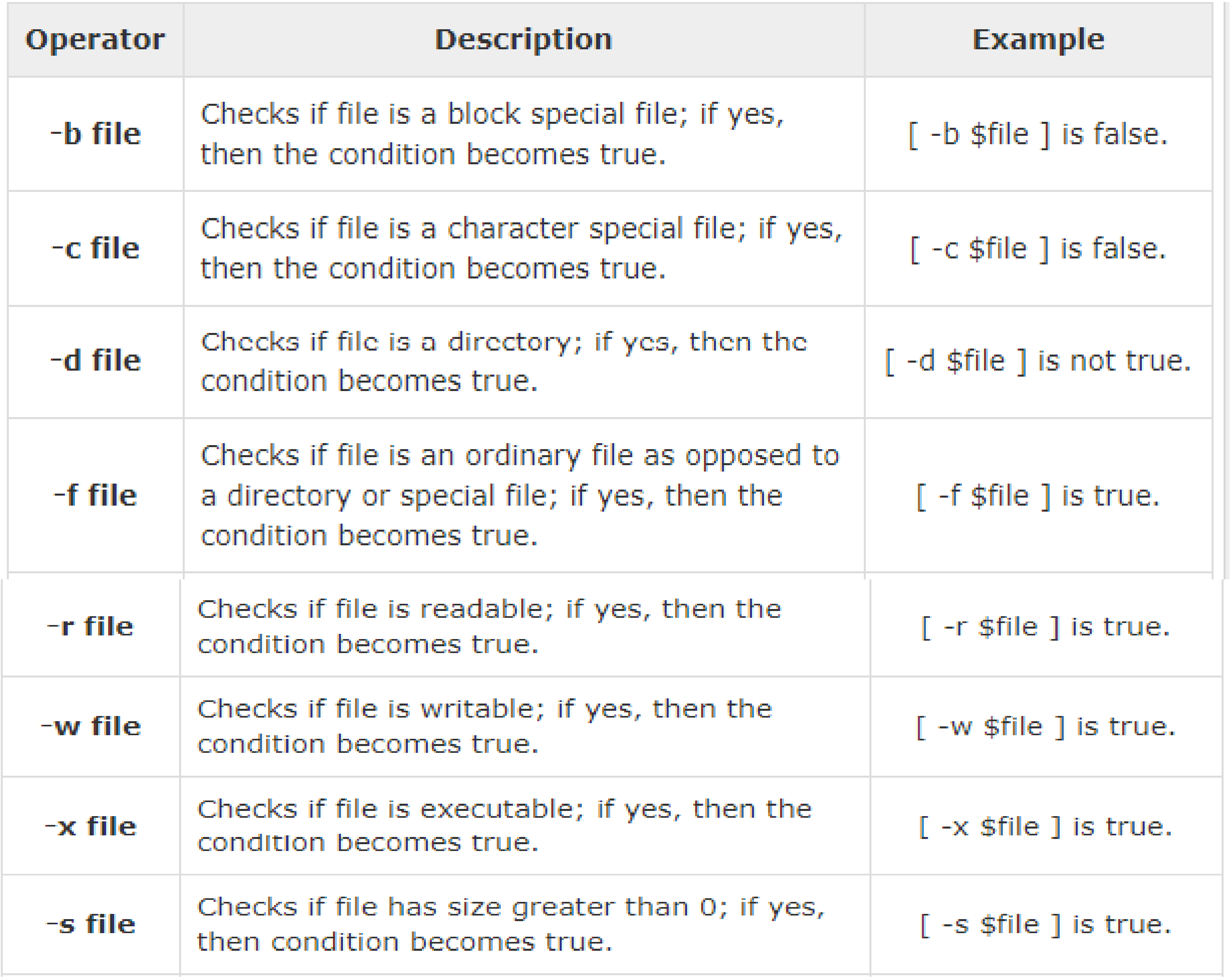
Functions

- function_name ()

{ list of commands }

```
#!/bin/sh
# Define your function here
 Hello () { echo "Hello World" }
 # Invoke your function
 Hello
```

File Tests



Operator	Description	Example
-b file	Checks if file is a block special file; if yes, then the condition becomes true.	[-b $file] is false.
-c file	Checks if file is a character special file; if yes, then the condition becomes true.	[-c $file] is false.
-d file	Checks if file is a directory; if yes, then the condition becomes true.	[-d $file] is not true.
-f file	Checks if file is an ordinary file as opposed to a directory or special file; if yes, then the condition becomes true.	[-f $file] is true.
-r file	Checks if file is readable; if yes, then the condition becomes true.	[-r $file] is true.
-w file	Checks if file is writable; if yes, then the condition becomes true.	[-w $file] is true.
-x file	Checks if file is executable; if yes, then the condition becomes true.	[-x $file] is true.
-s file	Checks if file has size greater than 0; if yes, then condition becomes true.	[-s $file] is true.

Assignment

1) **Write a shell script that displays a list of all files in the current directory to which you have read, write and execute permissions.**
2) **Write a shell script which displays the message “welcome” and prints the date when you log in to your system.**
3) **Write a shell script that lists files by modification time when called with lm and by access time when called with la. By default, the script should show the listing of all files in the current directory.**
4) **Write a shell script to display the files created or updated within fourteen days from the current date.**
5) **Develop a shell script that displays all files with all attributes those have been created or modified in the month of November**

Assignment

1) **Write a shell script to check if a given file (filename supplied as command line argument) is a regular file or not and find the total number of words, characters and lines in it.**
2) **Write a shell script, which reads a directory name and compares the current directory with it (which has more files and how many more files).**
3) **Write a shell function size( ) which lists only the total size of the files(filenames supplied as arguments).**
4) **Write a shell script to check whether the given file is a blank file or not. If not found blank then display the contents of the file.**
5) **Write a shell script to concatenate two files and count the number of characters, number of words and number of lines in the resultant file.**

Home Assignment

- **Write a shell script, which gets executed the moment a user logs in. It should display the message "GOOD MORNING" or "GOOD AFTERNOON" or "GOOD EVENING" depending upon the time at which the user logs in.**
- **Write a shell script that accepts two directory names, say btech1 and btech2 as arguments and deletes those files in btech2, which have identical named files in btech1.**
- **Write a shell script to make a password based menu-driven program, which will give a maximum of three chances to enter the password. If the given password is correct then the program will show the**
  - **Number of users currently logged in.**
  - **Calendar of current month.**
  - **Date in the format: dd / mm / yyyy.**
  - **Quit**

  The menu should be placed approximately in the centre of the screen and should be displayed in bold.

Attendance Code for Practical

Examination code is 2022000698 AIML
Examination code is 2022000699 IOT
Examination code is 2022000701 CSE D
Examination code is 2022000702 CSBS