shell scripting

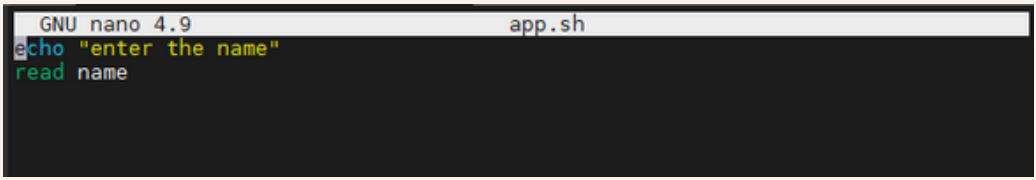
DEVOPS ENGINEER

CHANDA GULSHAN.M.A

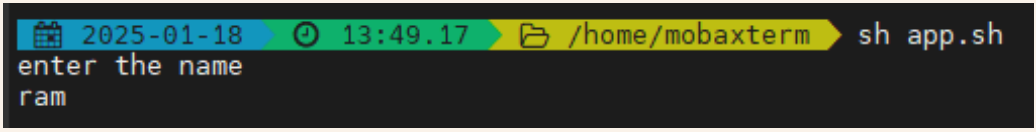
HOW TO CREATE THE SCRIPT FILE

nano <filename>.sh - *The command "nano app.sh" is used to open or create a script file named app.sh using the Nano text editor.*

SCRIPT FILE:



HOW TO EXECUTE OR RUN THE SCRIPT



PROGRAM NO.1

HOW TO ADD TWO NUMBERS

```
echo "enter  first number:"
read num1
echo "enter second number:"
read num2
echo "sum of  the two number is $(($num1 + $num2))"
```

OUTPUT:

```
enter  first number:
4
enter second number:
4
sum of  the two number is 8
```

$COMMANDS(COMMAND LINE ARGUMENTS)

$1-First argument
$2-Second argument
$3-Third argument
$0-It will print filename
$#-It will display number of arguments

$COMMANDS SCRIPT FILE:

```
 GNU nano 4.9                    app.sh
echo "$1"
echo "$2"
echo "$3"
echo "$0"
echo "$#"
```

OUTPUT:

```
2025-01-18  15:12.40  /home/mobaxterm  sh app.sh ram vim tin
ram
vim
tin
app.sh
3
```

HOW TO USE $ COMMANDS WITH VARIABLES:

```
 GNU nano 4.9                                app.sh
c=$(($1 + $2))
echo "add two number $c"
```

OUTPUT:

```
2025-01-18  19:33.01  /home/mobaxterm  sh app.sh 4 5
add two number 9
```

CONDITIONAL STATEMENT:

- IF ELSE STATEMENT
- ELIF STATEMENT

IF STATEMENT:

SYNTAX:

```
if [ expression ]
then
   statement1
else
   statement2
fi
```

NUMERIC COMPARISON OPERATORS:

- -eq → EQUAL TO
- -gt → GREATER THAN
- -lt → LESSER THAN
- -ge → GREATER THAN OR EQUAL TO
- -le → LESSER THAN OR EQUAL TO
- -ne → NOT EQUAL

PROGRAM NO.2:

```
  GNU nano 4.9                                  app.sh
echo "enter ur age"
read age

if [ $age -gt 18 ];
then
     echo "you can vote"
else
     echo "you can't vote"
fi
```

OUTPUT:

```
 2025-01-18  23:26.15  /home/mobaxterm  sh app.sh
enter ur age
30
you can vote
```

```
 2025-01-18  23:30.08  /home/mobaxterm  sh app.sh
enter ur age
15
you can't vote
```

ELIF 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
elif [ expression 3 ]
then
   Statement(s) to be executed if expression 3 is true
else
   Statement(s) to be executed if no expression is true
fi
```

PROGRAM NO.3:

```
 GNU nano 4.9                                app.sh
echo "enter ur age"
read age

if [ $age -gt 18 ];
then
     echo "you can vote"

elif [ $age -eq  18 ];
then
     echo "you can apply for vorter ID"
else
     echo "you can't vote"
fi
```

OUTPUT:

```
2025-01-18  23:46.53  /home/mobaxterm  sh app.sh
enter ur age
18
you can apply for vorter ID
```

```
2025-01-18  23:50.19  /home/mobaxterm  sh app.sh
enter ur age
20
you can vote
```

```
2025-01-18  23:50.24  /home/mobaxterm  sh app.sh
enter ur age
14
you can't vote
```

STRING COMPARISON:

PROGRAM NO.4:

```
 GNU nano 4.9                                  app.sh
echo "enter your name:"
read name

if [ $name = raja ];
then
    echo "hi,$name"

elif [ $name = rani ];
then
    echo "hi,$name"
else
    echo "name is not mentioned"
fi
```

OUTPUT:

```
2025-01-18  23:58.32  /home/mobaxterm  sh app.sh
enter your name:
chanda
name is not mentioned
```

```
2025-01-18  23:58.13  /home/mobaxterm  sh app.sh
enter your name:
rani
hi,rani
```

```
2025-01-18  23:57.25  /home/mobaxterm  sh app.sh
enter your name:
raja
hi,raja
```

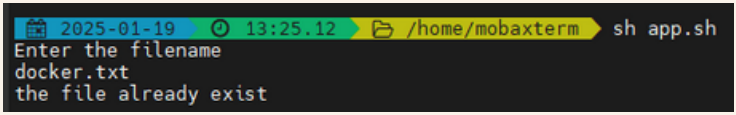
PROGRAM NO.5:

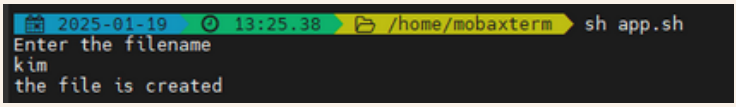
```
 GNU nano 4.9                                  app.sh
echo "Enter the filename"
read filename

if [ -f "$filename" ]
then
     echo "the file already exist"

else
    touch $filename
    echo "the file is created"
fi
```

OUTPUT:

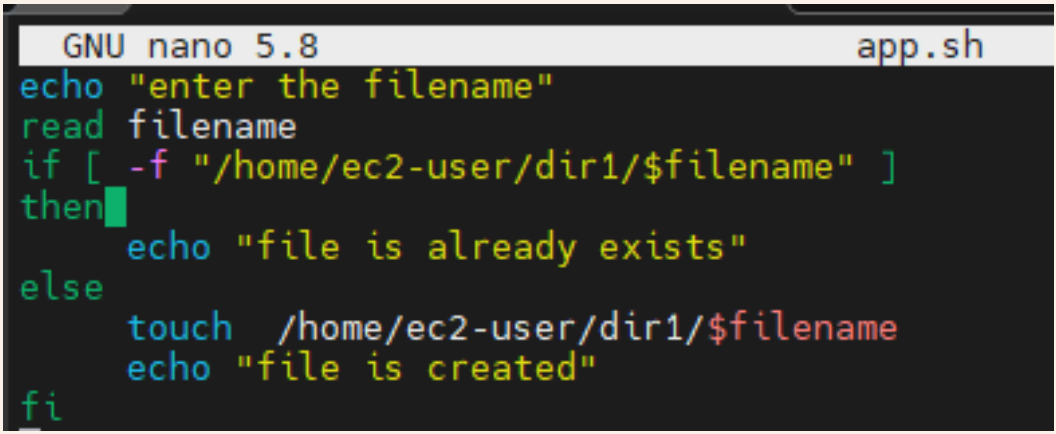




"If the file already exists, display the output 'File already exists.' If the file does not exist, create the file."

PROGRAM NO.6:

"Check if the file exists in the specified directory. If it exists, display 'File already exists.' If not, create the file in that directory."



OUTPUT:

```
[ec2-user@ip-172-31-43-188 ~]$ cd dir1
[ec2-user@ip-172-31-43-188 dir1]$ ls
app.sh  file1  file2  file3
[ec2-user@ip-172-31-43-188 dir1]$ touch docker ansible shell
```

```
[ec2-user@ip-172-31-43-188 ~]$ sh app.sh
enter the filename
docker
file is already exists
```

```
[ec2-user@ip-172-31-43-188 ~]$ sh app.sh
enter the filename
git
file is created
```

LOOPS:

- *FOR LOOP*
- *WHILE LOOP*
- *DO WHILE LOOP*

forLoop:

SYNTAX:

```
for ((initialization ; condition ; increment))
do
    echo "statement"
done
```

PROGRAM NO.7:

"PRINT THE FIRST 10 NUMBERS."

```
  GNU nano 5.8                                script.sh
for ((i=1;i<=10;i++))
do
    echo "$i"
done
```

OUTPUT:

```
[ec2-user@ip-172-31-43-188 ~]$ sh script.sh
1
2
3
4
5
6
7
8
9
10
```

while loop:

SYNTAX:

```
initialization
while [condition]
do
    statement
    ((increment))
done
```

PROGRAM NO.8:

"PRINT THE FIRST 10 NUMBERS."

```
  GNU nano 5.8                                script1.sh
i=1
while [i -le 10]
do
   echo "$i"
   ((i++))
done
```

OUTPUT:

```
[ec2-user@ip-172-31-43-188 ~]$ sh script.sh
1
2
3
4
5
6
7
8
9
10
```

FUNCTIONS:

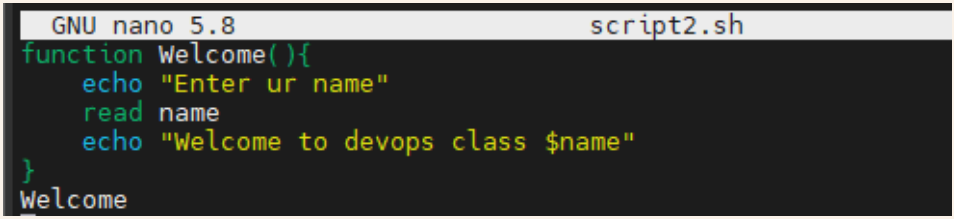
DEFINITION:

"Functions in shell scripts group commands into reusable blocks, accept arguments, perform tasks, and return results or exit status."

SYNTAX:

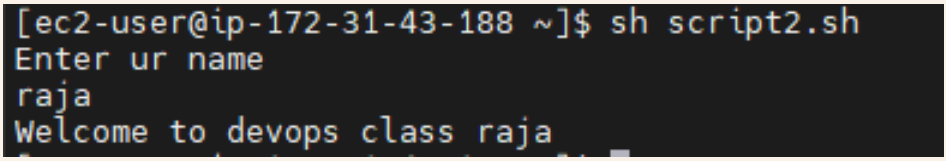
```
Function Function name()
{
    statement
}
Function name
```

PROGRAM NO.9:



```
  GNU nano 5.8                                script2.sh
function Welcome(){
    echo "Enter ur name"
    read name
    echo "Welcome to devops class $name"
}
Welcome
```

OUTPUT:



```
[ec2-user@ip-172-31-43-188 ~]$ sh script2.sh
Enter ur name
raja
Welcome to devops class raja
```

-P PROMPT COMMAND:

"The -p option in read shows a prompt message before asking for user input, making it clearer for the user".

PROGRAM NO.10:

```
  GNU nano 5.8                                  script3.sh
function welcome()
{
    read -p "enter your name:" name
    echo "welcome to devops class $name"
}
welcome
```

OUTPUT:

```
[ec2-user@ip-172-31-43-188 ~]$ sh script3.sh
enter your name:chanda
welcome to devops class chanda
```

PROGRAM NO.11:

```
  GNU nano 5.8                                  script4.sh
function READTHEFILE()
{
    read -p "Enter the filename:" FILENAME
    echo "FILE READING"
    cat $FILENAME
    echo "FILE READING  IS COMPLETED"
}
READTHEFILE
```

```
  GNU nano 5.8                                  index.html
<h1>welcome to devops class</h1>
<h2>hi,hello<h2>
```

OUTPUT:

```
[ec2-user@ip-172-31-43-188 ~]$ sh script4.sh
Enter the filename:index.html
FILE READING
<h1>welcome to devops class</h1>
<h2>hi,hello<h2>
```

PROGRAM NO.12:

```
  GNU nano 5.8                                script5.sh
function READTHEFILE()
{
    read -p "ENTER THE FILENAME:" FILENAME
    if [ -f "$FILENAME" ];
    then
         echo "FILE IS READING"
         cat "$FILENAME"
         echo "FILE READING IS COMPLETED"
    else
         echo "FILE DOES NOT EXIST"
    fi
}
READTHEFILE
```

OUTPUT:

```
[ec2-user@ip-172-31-43-188 ~]$ sh script5.sh
ENTER THE FILENAME:index.html
FILE IS READING
<h1>welcome to devops class</h1>
<h2>hi,hello<h2>
```

```
[ec2-user@ip-172-31-43-188 ~]$ sh script5.sh
ENTER THE FILENAME:style.css
FILE DOES NOT EXIST
```

PROGRAM NO.13:

```
  GNU nano 5.8                          script6.sh
function READTHEFILE() {
  read -p "ENTER THE FILENAME:" FILENAME
  if [ -f "$FILENAME" ]
  then
      echo "FILE IS READING"
      cat "$FILENAME"
  else
      touch "$FILENAME"
      echo "hello world">"$FILENAME"
      echo "file is created and content is also written."
      cat "$FILENAME"
  fi
}
READTHEFILE
```

OUTPUT:

```
[ec2-user@ip-172-31-43-188 ~]$ sh script6.sh
ENTER THE FILENAME:HELLO
file is created and content is also written.
hello world
```

CRONTAB

CRONTAB IS A COMMAND-LINE TOOL IN LINUX USED TO SCHEDULE REPETITIVE TASKS OR SCRIPTS (CRON JOBS) TO RUN AUTOMATICALLY AT SPECIFIED TIMES OR INTERVALS.

SYNTAX OF LINUX CRONTAB:

```
MIN HOUR DOM MON DOW CMD
```

1.*-min(0-59)
2.*- hours(0-23)
3.*- day of month(1-31)
4.*- month(1-12)
5.*- week(mon-sun)

Field	Description	Allowed Value
MIN (Minute)	Specifies the minute when the command will run	It ranges from 0 to 59.
HOUR	Denotes the hour of the day when the command is scheduled to execute.	It spans from 0 to 23.
DOM (Day of Month)	Specifies the day of the month for the task.	It ranges from 1 to 31.
MON (Month)	Indicates the month during which the command will be executed.	It varies from 1 to 12.
DOW (Day of Week)	Specifies the day of the week for the task.	It is represented by numbers from 0 to 7, where both 0 and 7 correspond to Sunday.
CMD (Command)	Represents the actual command or script that will run at the scheduled time.	————

SOME EXAMPLE OF CRONTAB COMMANDS:

1.To execute a script file every 15 minutes.
*/15 * * * *
2.To execute a script file at 10 AM every day.
0 10 * * *
3.To execute a script file at 5 PM every day.
0 17 * * *

PROGRAM NO.14:

STEPS:

1.CREATE SCRIPT FILE

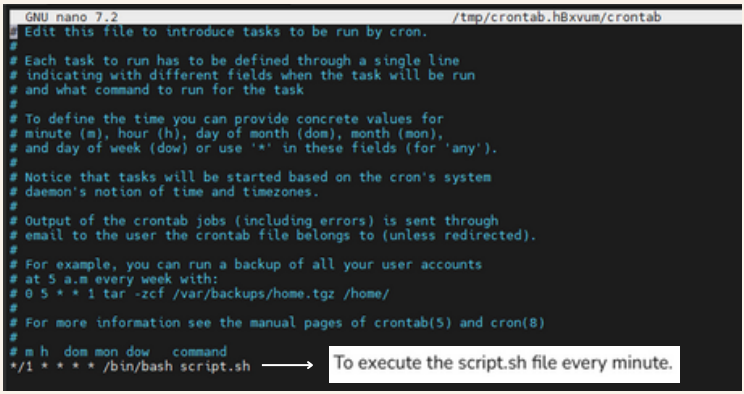
```
ubuntu@ip-172-31-44-148:~$ nano script.sh
```

```
 GNU nano 7.2                                    script.sh *
mkdir directory1
touch file1 file2
```

2.CRONTAB -E

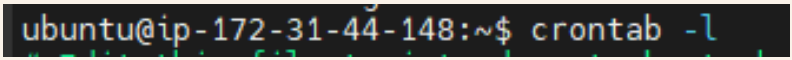
-opens the cron table for editing tasks.

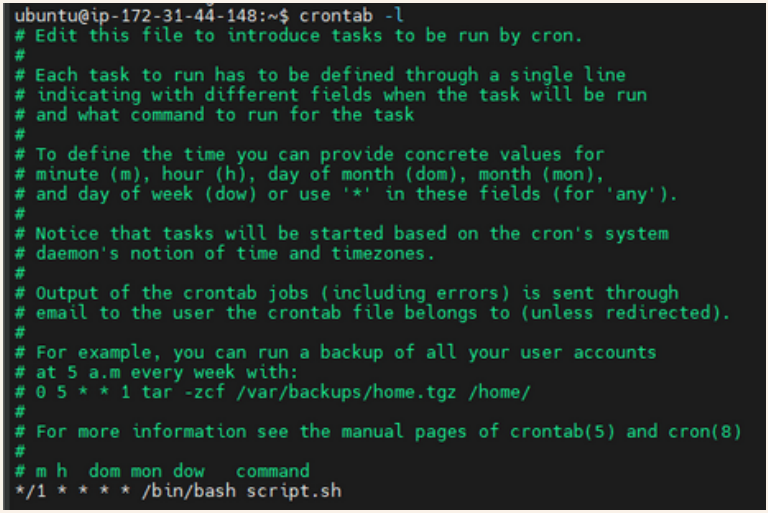
```
ubuntu@ip-172-31-44-148:~$ crontab -e
```



3. CRONTAB -L

- lists all the crontab jobs scheduled for the current user.





OUTPUT: