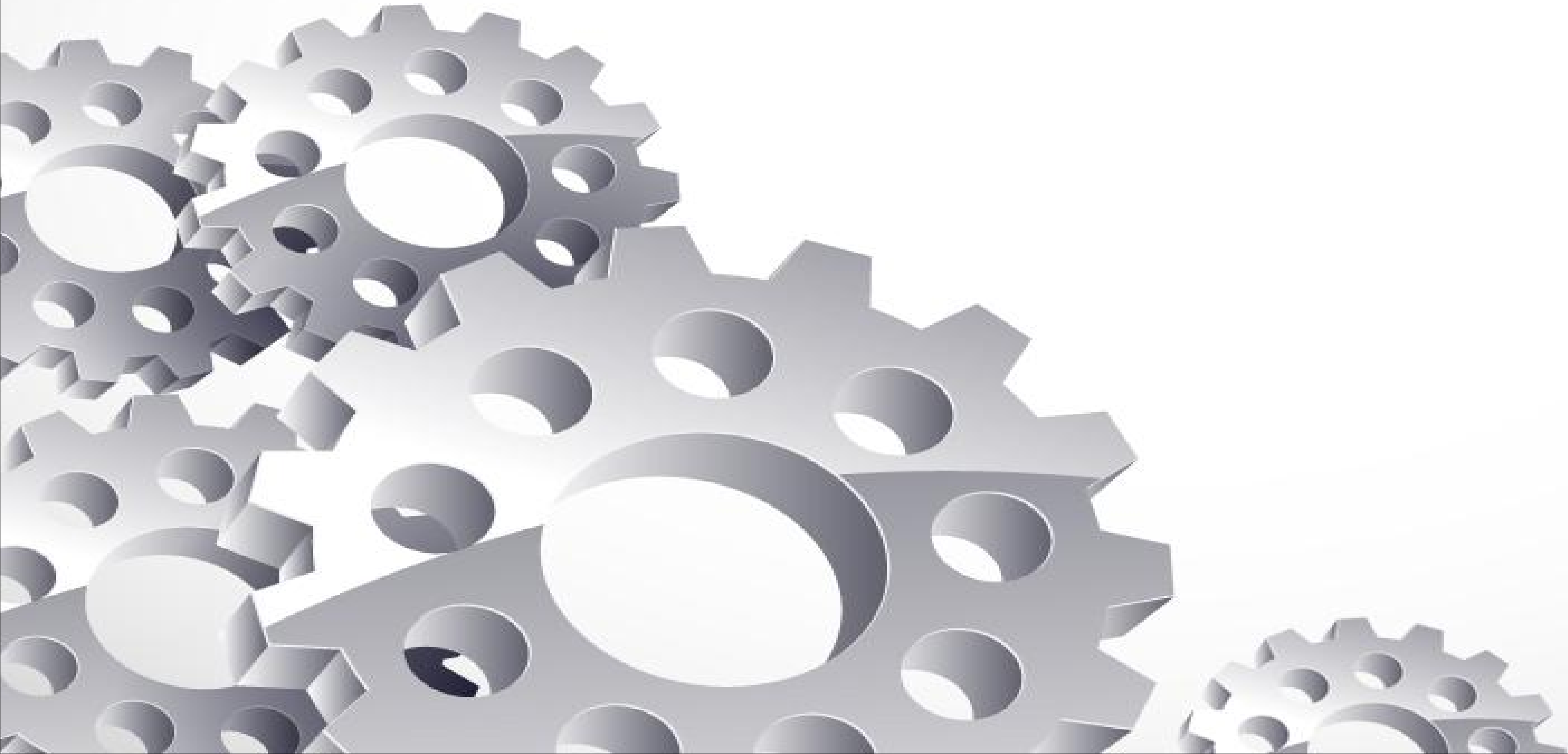
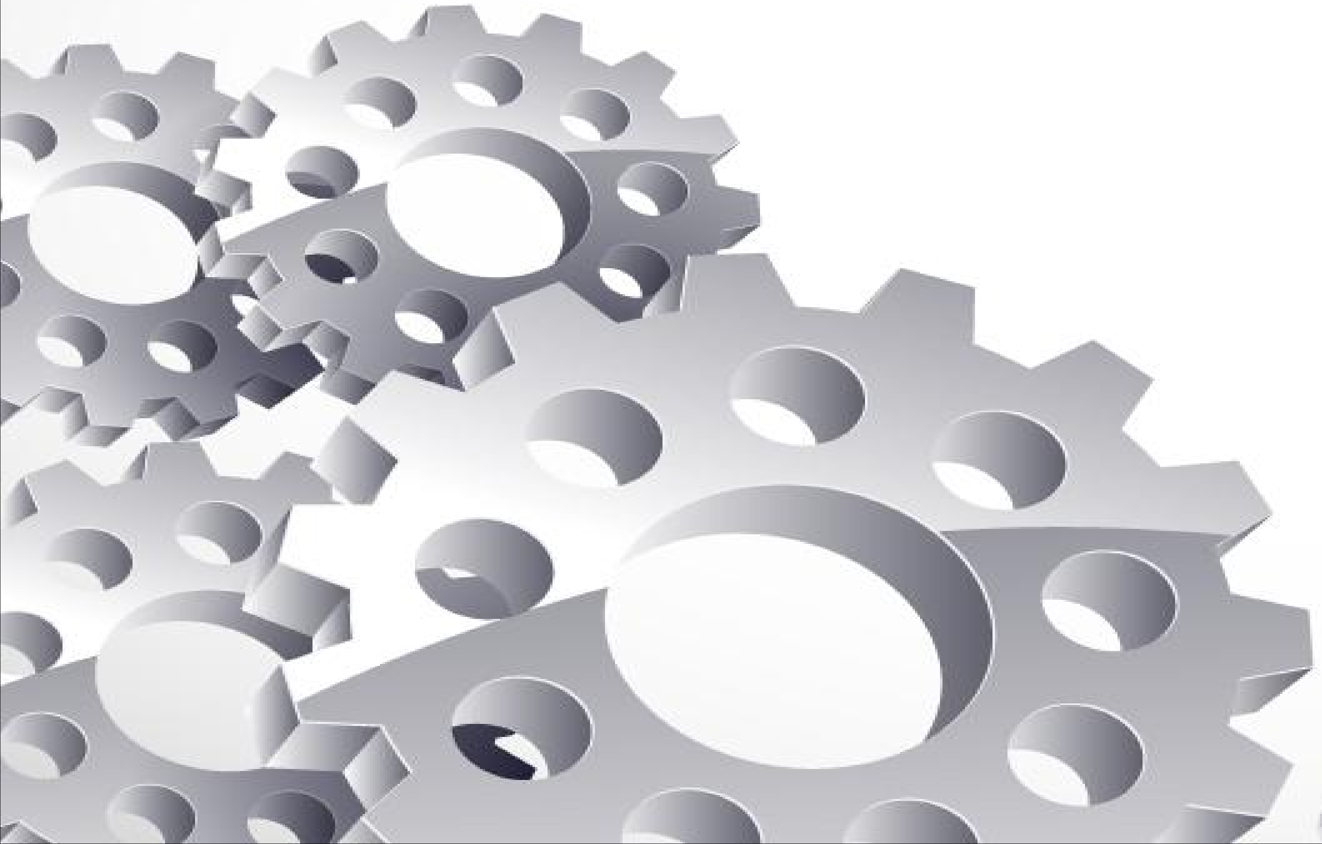
Shell Programming



chmod command



Make Script Executable in Linux | chmod Command

- In Unix operating systems, the **chmod** command is used to change the access mode of a file. The name is an abbreviation of change mode. Which states that every file and directory has a set of permissions that control the permissions like who can read, write or execute the file.
- Three categories: read, write, and execute simultaneously represented by \`r\`, \`w\` and \`x\`.
- These letters combine together to form a specific permission for a group of users.

Syntax of chmod command

- chmod [options] [mode] [File_name]
- Options: Optional flags that modify the behavior of the chmod command.
- Mode: The permissions to be set, represented by a three-digit octal number or symbolic notation (e.g., u=rw,go=rx).
- File_name: The name of the file or directory for which the permissions are to be changed.

Modes in chmod Command in Linux

- The “mode” helps in setting new permissions that have to be applied to files or directories.

- This mode can be specified in several ways, we will discuss two modes: Symbolic and Octal mode.

Symbolic mode

- The following operators can be used with the symbolic mode:

Operators	Definition
`+`	Add permissions
`-`	Remove permissions

Letters	Definition
`r`	Read permission
`w`	Write permission
`x`	Execute permission

- The following Reference that are used:

Reference	Class
u	Owner
g	Group
o	Others
a	All (owner,groups,others)

Examples

- Read, write and execute permissions to the file owner:
- chmod u+rwx [file_name]

- Remove write permission for the group and others:
- chmod go-w [file_name]

- Read and write for Owner, and Read-only for the group and other:
- chmod u+rw,go+r [file_name]

Octal mode

- It is also a method for specifying permissions. In this method we specify permission using three-digit number. Where..

- First digit specify the permission for Owner.
- Second digit specify the permission for Group.
- Third digit specify the permission for Others. The digits

The digits are calculated by adding the values of the individual permissions.

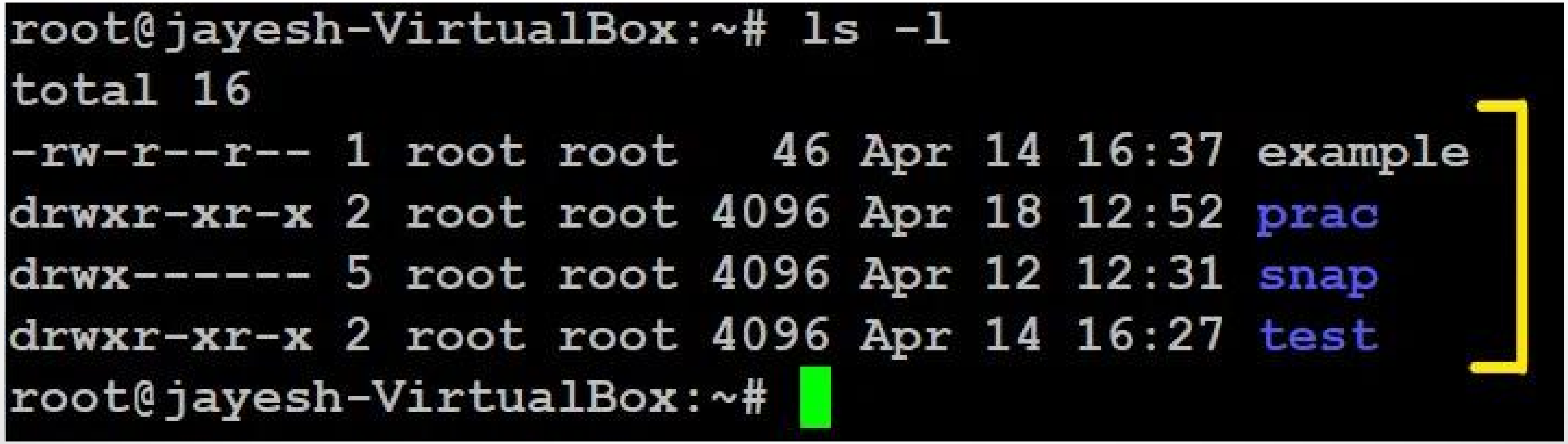
Value	Permission
4	Read Permission
2	Write Permission
1	Execute Permission

Examples

- Give read and write permission to the file Owner.
- Read, write and executable permission to the Group.
- Read-only permission to the Other.
- command would be.
  - chmod 674 [file_name]

Check Current Permissions

- Use the `ls` command with the `-l` option to list the files in the directory along with their permissions. This step helps you identify the current permissions of your script:
- ls -l



```
root@jayesh-VirtualBox:~# ls -l
total 16
-rw-r--r-- 1 root root   46 Apr 14 16:37 example
drwxr-xr-x 2 root root 4096 Apr 18 12:52 prac
drwx------ 5 root root 4096 Apr 12 12:31 snap
drwxr-xr-x 2 root root 4096 Apr 14 16:27 test
root@jayesh-VirtualBox:~#
```

Variables

- Create a variable
  - Variablename=value (no spaces, no $)
  - read variablename (no $)
- Access a variable's value
  - $variablename
- Set a variable
  - Variablename=value (no spaces, no $ before variablename)
- Sample:

```
wget
http://home.adelphi.edu/~pe16132/csc271/note/scripts/playwithvar
```

The **read** Command (continued)

Read from stdin (screen)
Read until new line

Format	Meaning
`read answer`	Reads a line from `stdin` into the variable `answer`
`read first last`	Reads a line from `stdin` up to the whitespace, putting the first word in `first` and the rest of the of line into `last`
`read`	Reads a line from `stdin` and assigns it to `REPLY`
`read –a arrayname`	Reads a list of word into an array called `arrayname`
`read –p prompt`	Prints a prompt, waits for input and stores input in `REPLY`
`read –r line`	Allows the input to contain a backslash.

wget
http://home.adelphi.edu/~pe16132/csc271/note/script
s/nosy

User defined

1. a=50   # predefined value

    echo “$a”

2. read a b c   # user input from terminal
3. read –a name # to take array input

    echo “${name[0]} ${name[1]} ${name[2]}”

4. For real values

```
read rad
area=`echo 3.14 \* $rad \* $rad |bc -l`
echo $area
```

Expression

To add integer numbers

```
Val=`expr $a + $b`
Val1=`expr $a + 100`
```

To multiply integer numbers

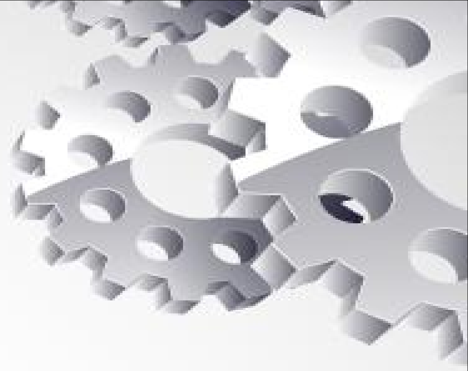
```
Val=`expr $a \* $b`
```

If Statements

```
if  [ <some test> ]
   then
         <commands>
fi
```

Example

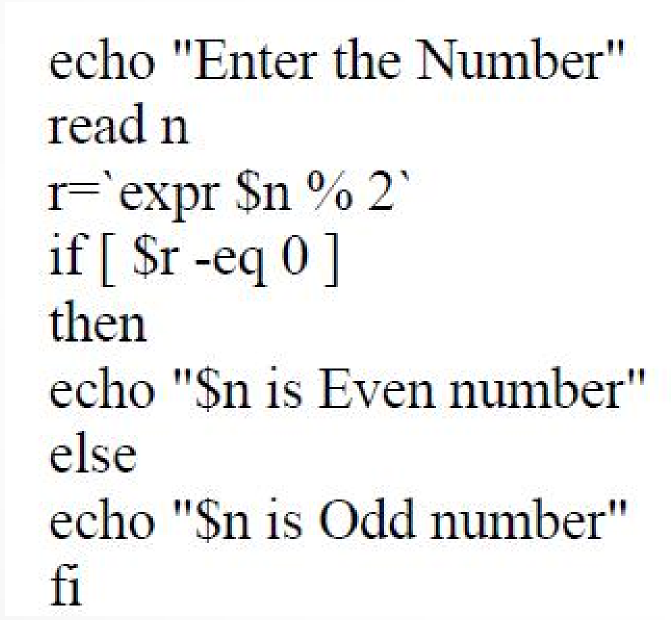
```
if [ $val -gt 100 ]
then
      echo "That\'s a large number"
      pwd    # directory
fi
date          # date
```



AIM: To write simple shell programs by using conditional, branching and looping statements.

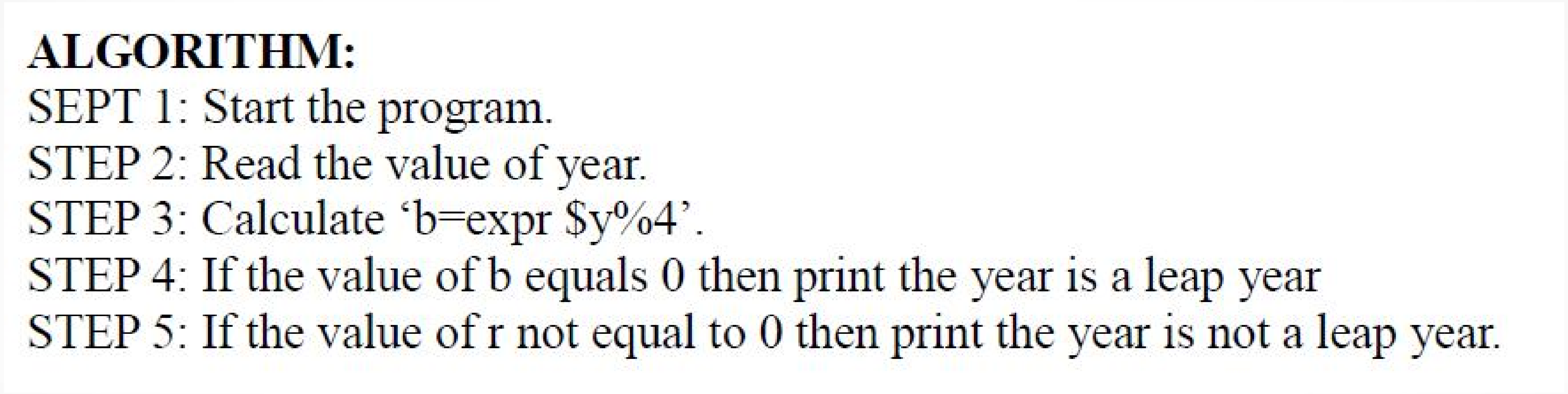
- 1.Write a Shell program to check the given number is even or odd
- ALGORITHM:
- SEPT 1: Start the program.
- STEP 2: Read the value of n.
- STEP 3: Calculate “r=expr $n%2‟.
- STEP 4: If the value of r equals 0 then print the number is even
- STEP 5: If the value of r not equal to 0 then print the number is odd.

1: Write a Shell program to check the given number is even or odd



```
echo "Enter the Number"
read n
r=`expr $n % 2`
if [ $r -eq 0 ]
then
echo "$n is Even number"
else
echo "$n is Odd number"
fi
```

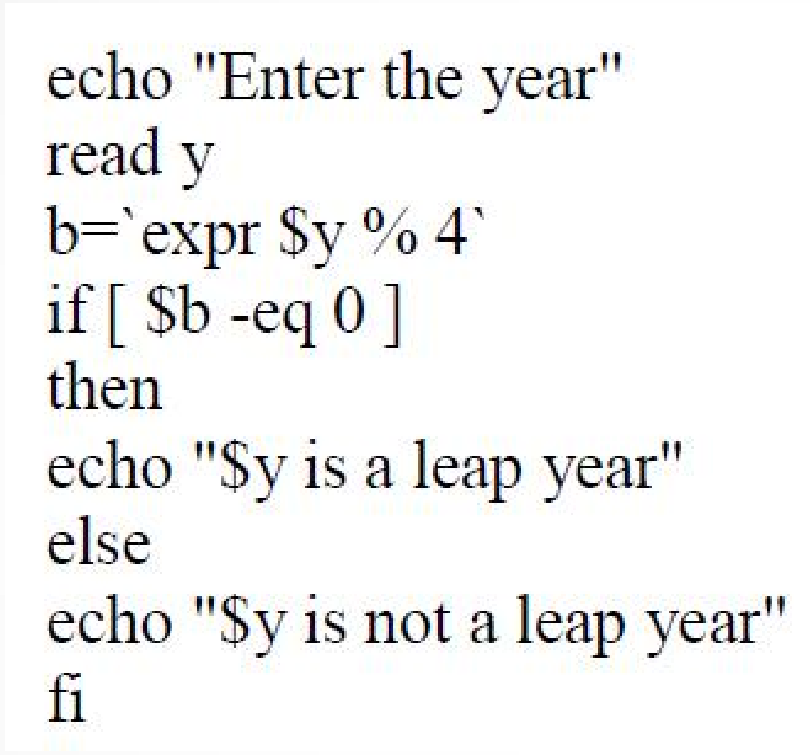
2.Write a Shell program to check the given year is leap year or not (Using only divisible by 4)



ALGORITHM:

SEPT 1: Start the program.
STEP 2: Read the value of year.
STEP 3: Calculate ‘b=expr $y%4’.
STEP 4: If the value of b equals 0 then print the year is a leap year
STEP 5: If the value of r not equal to 0 then print the year is not a leap year.

PROGRAM-2:



```
echo "Enter the year"
read y
b=`expr $y % 4`
if [ $b -eq 0 ]
then
echo "$y is a leap year"
else
echo "$y is not a leap year"
fi
```

Lab Tasks:

3. Write a Shell program to check the given number is positive or negative.
4. WAP to take marks of three subjects and find the average mark obtained.
5. Write a program to swap two values using third variable.
6. WAP to calulate the area and perimeter of a circle from its radius.

Floating Point Arithmetic

- `SIEGFRIE@panther:~$ n=`echo "scale=3; 13 / 2" | bc``

Logical

- **-o for OR**
- **-a for AND**

test Command Operators – Integer Tests

Comparing numbers

- remember (( ))
- -eq , -ne, -gt, -ge, -lt, -le

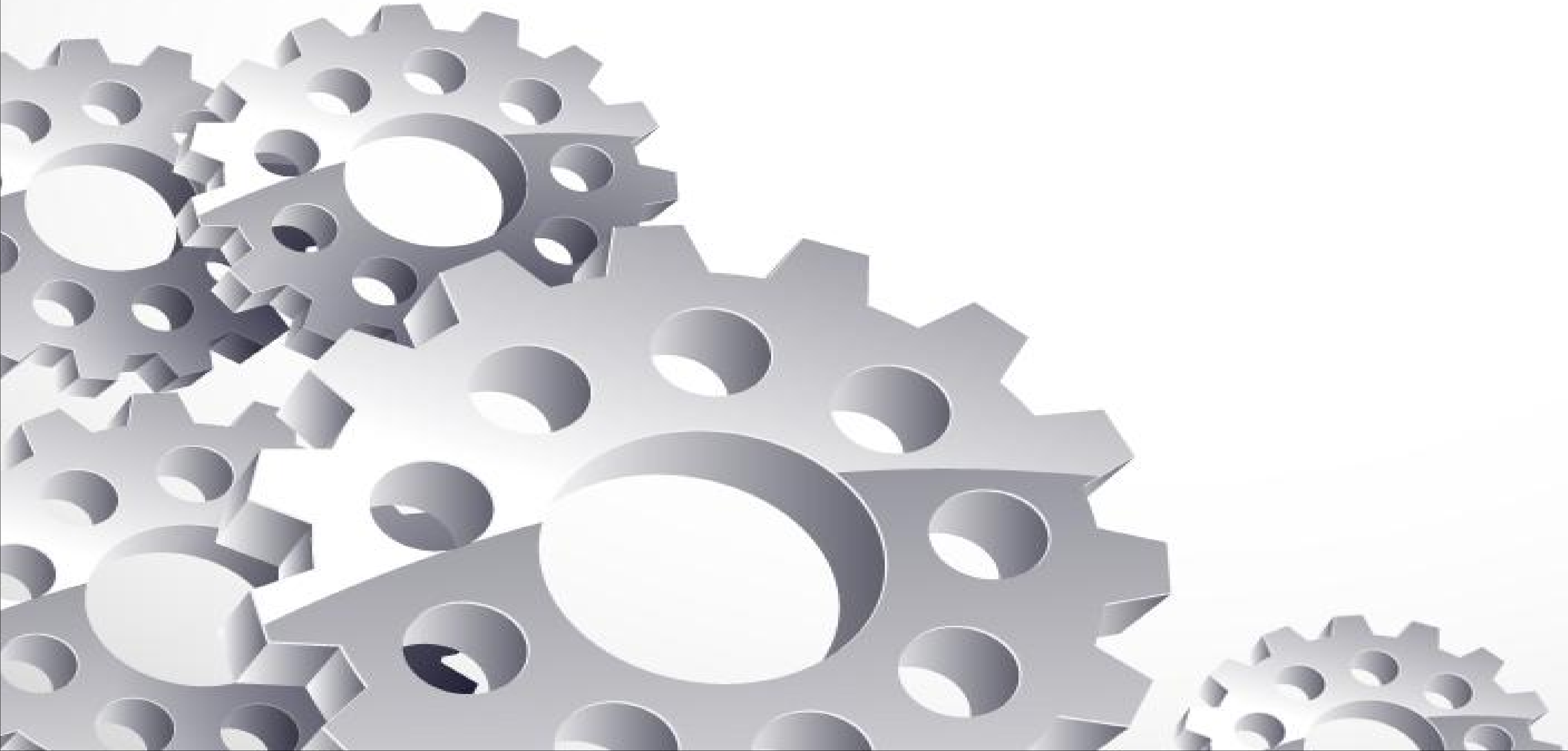
Test operator	Tests True if
[*int1* **-eq** *int2* **]**	int1 = int2
[*int1* **-ne** *int2* **]**	int1 ≠ int2
[*int1* **-gt** *int2* **]**	int1 > int2
[*int1* **-ge** *int2* **]**	int1 ≥ int2
[*int1* **-lt** *int2* **]**	int1 < int2
[*int1* **-le** *int2* **]**	int1 ≤ int2

test Command Operators – Logical Tests

Test Operator	Test True If
[*string1* **-a** *string2* **]**	Both string1 and string 2 are true.
[*string1* **-o** *string2* **]**	Both string1 or string 2 are true.
[! *string* **]**	Not a string1 match

Test operator	Tests True if
[[*pattern1* **&&** *Pattern2* **]]**	Both pattern1 and pattern2 are true
[[*pattern1* **||** *Pattern2* **]]**	Either pattern1 or pattern2 is true
[[!*pattern* **]]**	Not a pattern match

Loops

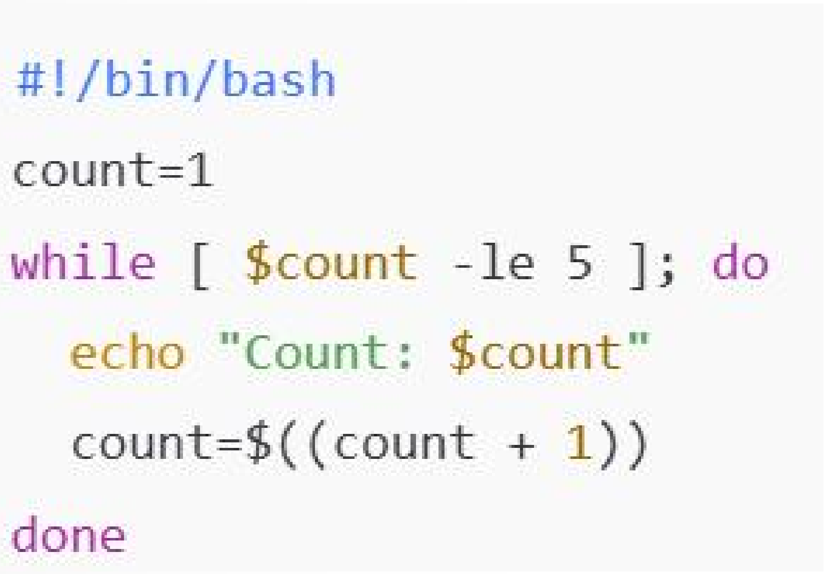


Loopings

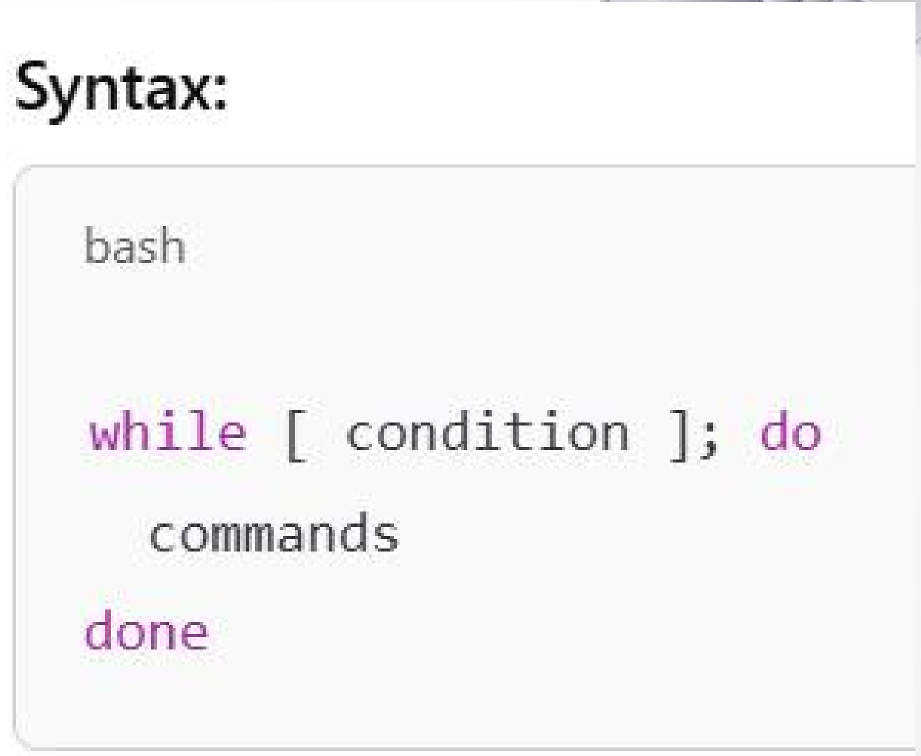
- In UNIX shell scripting, loops like while, for, until, and conditional branching like case are essential for automating repetitive tasks and handling logic.

while Loop

- Executes commands as long as a specified condition evaluates to true.



```bash
#!/bin/bash
count=1
while [ $count -le 5 ]; do
  echo "Count: $count"
  count=$((count + 1))
done
```



Syntax:

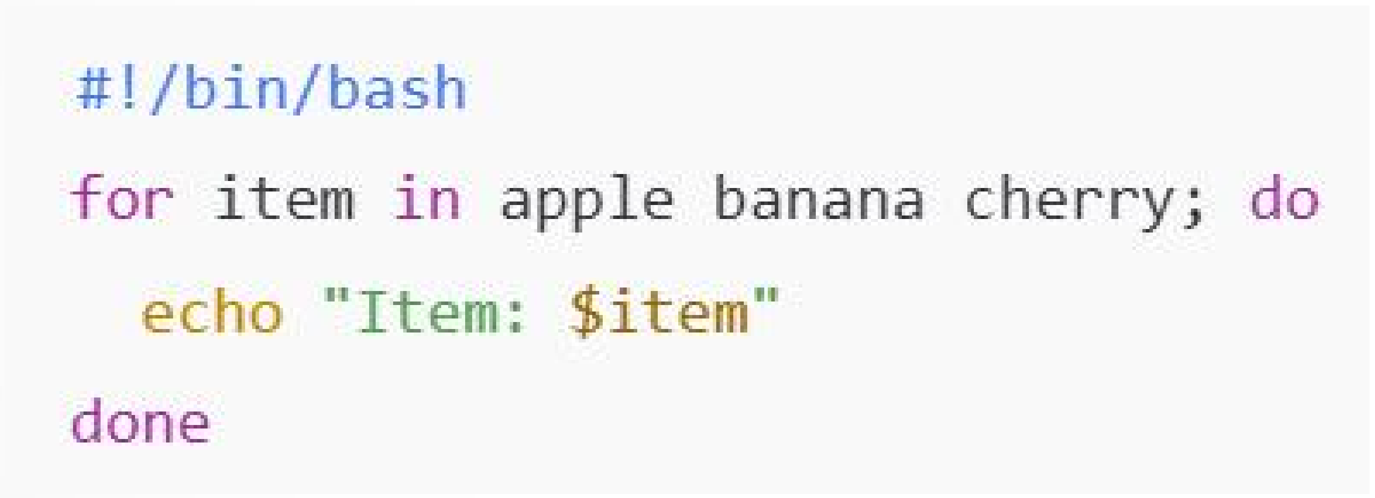
```
bash

while [ condition ]; do
  commands
done
```

- Explanation:

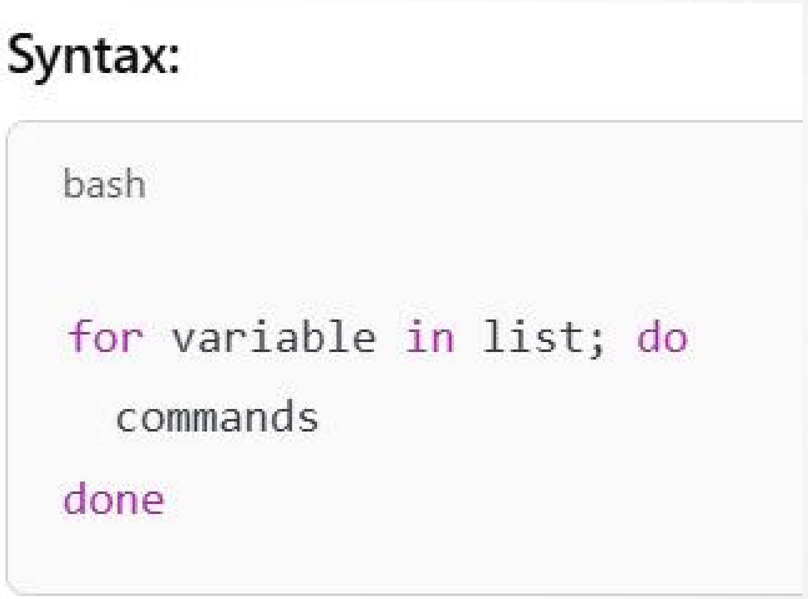
- [$count -le 5] checks if count is less than or equal to 5.
- The loop increments count by 1 in each iteration.

for Loop

- Iterates over a list of items or a range.
- Example 1: Iterating over a list



```bash
#!/bin/bash
for item in apple banana cherry; do
  echo "Item: $item"
done
```

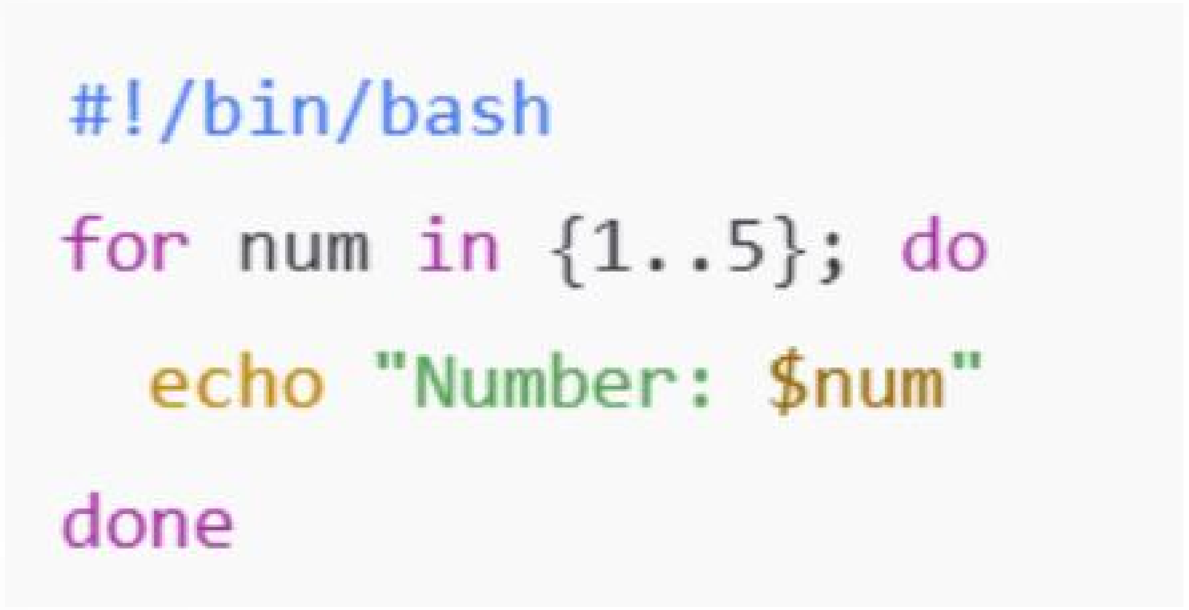


Syntax:

bash

```bash
for variable in list; do
  commands
done
```

Example 2: Iterating over a range

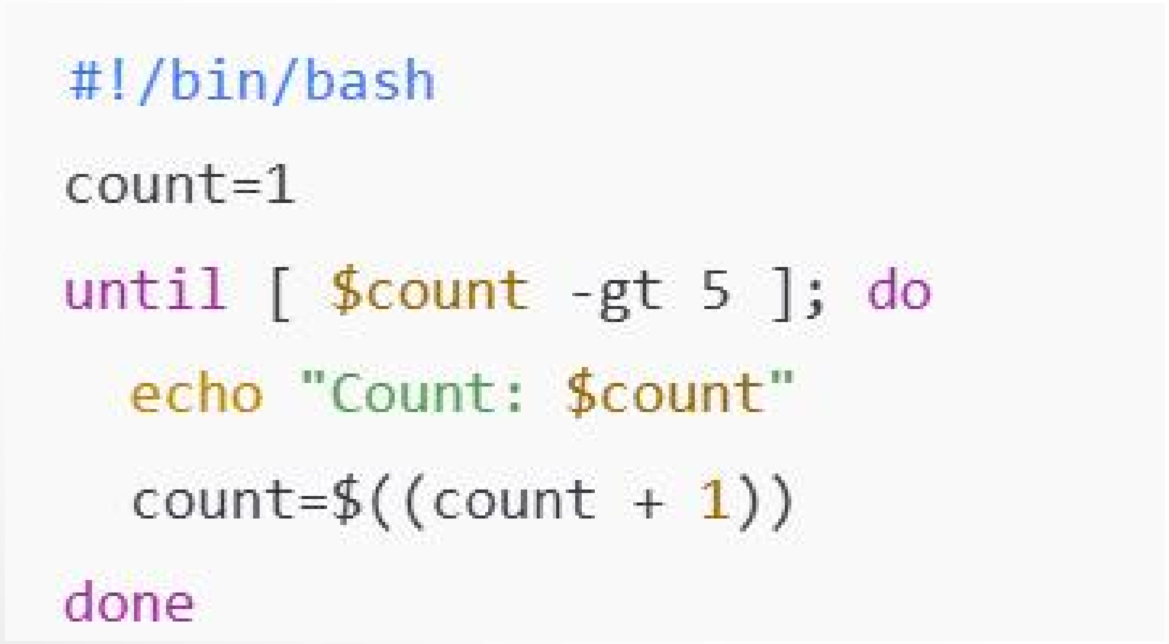


```bash
#!/bin/bash
for num in {1..5}; do
  echo "Number: $num"
done
```

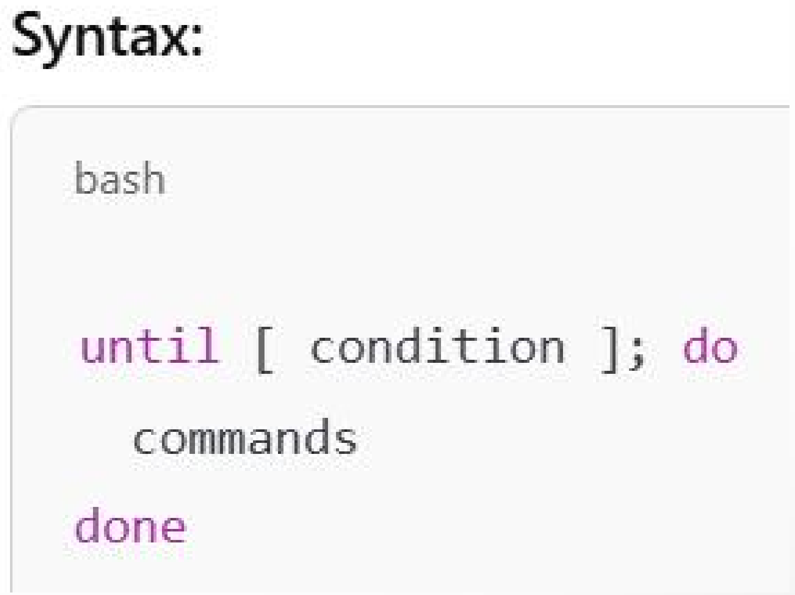
Explanation:
•item takes on each value in the list (apple, banana, cherry).
•{1..5} generates numbers from 1 to 5.

until Loop

- Executes commands until the specified condition evaluates to true. (opposite of while).
- Example: Counting to 5 using until



```bash
#!/bin/bash
count=1
until [ $count -gt 5 ]; do
  echo "Count: $count"
  count=$((count + 1))
done
```



Syntax:

bash

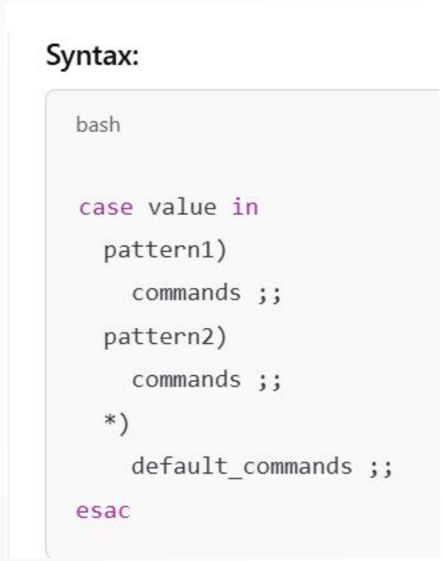
```bash
until [ condition ]; do
  commands
done
```

Explanation:

•The loop runs until count becomes greater than 5.

case Statement

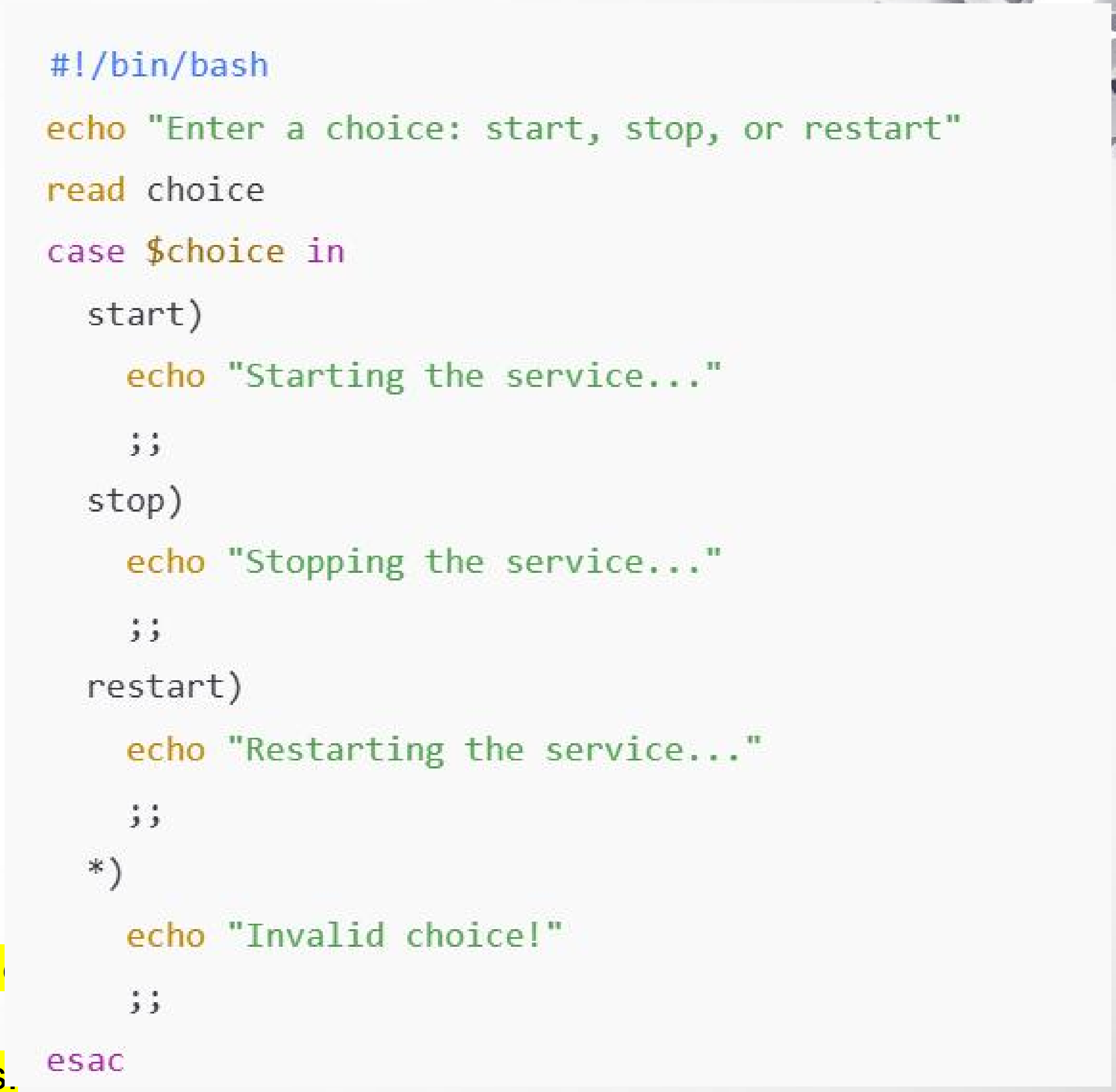
- Handles multiple conditional branches.



Syntax:

```bash
case value in
  pattern1)
    commands ;;
  pattern2)
    commands ;;
  *)
    default_commands ;;
esac
```

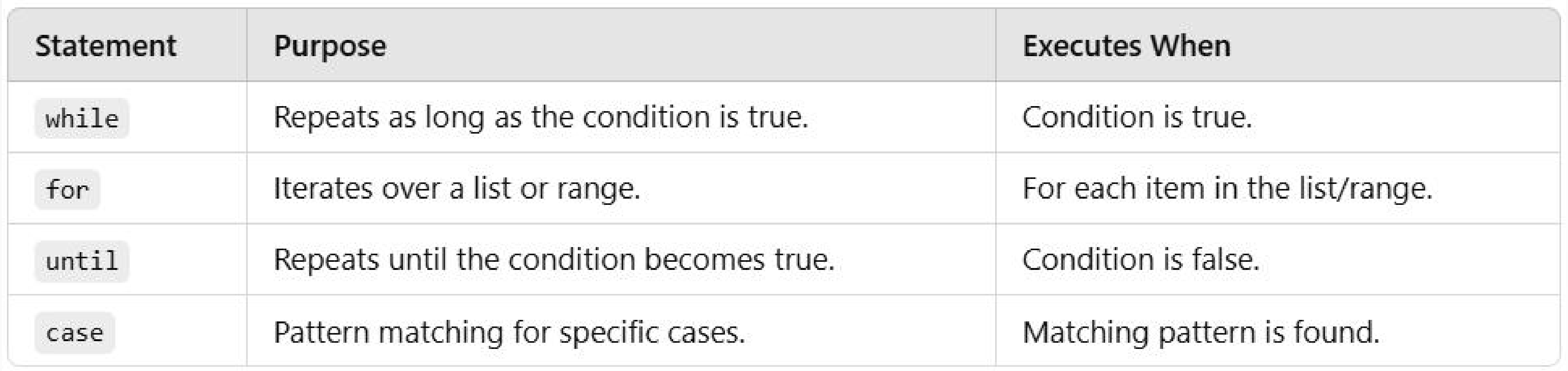
- Explanation:
- case evaluates the user input (choice) an executes the matching block.
- * is the default case for unmatched inputs.



```bash
#!/bin/bash
echo "Enter a choice: start, stop, or restart"
read choice
case $choice in
  start)
    echo "Starting the service..."
    ;;
  stop)
    echo "Stopping the service..."
    ;;
  restart)
    echo "Restarting the service..."
    ;;
  *)
    echo "Invalid choice!"
    ;;
esac
```

case

```bash
#!/bin/bash
# Respond based on user input
echo "Enter a number (1-3):"
read num
case $num in
  1)
    echo "You selected One" ;;
  2)
    echo "You selected Two" ;;
  3)
    echo "You selected Three" ;;
  *)
    echo "Invalid choice" ;;
esac
```



Statement	Purpose	Executes When
`while`	Repeats as long as the condition is true.	Condition is true.
`for`	Iterates over a list or range.	For each item in the list/range.
`until`	Repeats until the condition becomes true.	Condition is false.
`case`	Pattern matching for specific cases.	Matching pattern is found.

Shell Script Program Assignment

1. Write a shell program to convert distance from meter to km.
2. Write a shell program to display the prime number between 1 and hundred.
3. Write a shell script to reverse a given integer.
4. Write a shell program to find greatest among three numbers.
5. Write a shell program to input a day number from 1 - 7 display its day name using case statements
6. Write a shell program to input a character test it an alphabet or digit or special character using case statements.
7. Write a shell program to input an alphabet test it is a vowel or not using case statement.