GETTING STARTED
WITH
LOOP CONTROL STRUCTURE

➢ THE LOOP CONTROL STRUCTURE:

- The loop involves repeating some portion of the program either a specified number of times or until a particular condition is being satisfied.
- There are three methods by way of which we can repeat a part of a program. They are:
  - While Statement
  - Until Statement
  - For Statement

WHILE STATEMENT:

- The **bash while loop** is a control flow **statement** that allows code or commands to be executed repeatedly based on a given condition. (or)
- Looping is repeatedly executing a section of your program based on a condition.
- The form of while loop is:

```
while [ condition ]
do
-----
-----
done
-----
-----
```

Example: Print numbers from 1 to 10:

```
#!/bin/sh
#Print 1 to 10 Numbers
i=1
while [ $i -le 10 ]
do
echo $i
c=`expr $i+1`
done
```

THE UNTIL LOOP:

- The statement within the until loop keep on getting executed till the exit status of the control command remains **false(1).** When the exit status become **true(0),** the control passes to the first command that follows the body of the until loop.
- The general form of until loop is:

```
until [ condition ]
do
------
------
done
-------
-------
```

NOTE:

- There is a minor difference between the working of while and until loops. The while loop executes till the exit status of the control command is true and terminates when the exit status becomes false. Unlike this until loop executes till the exit status of the control command is false and terminates when this status becomes true.

  Example: Prints numbers from 1 to 10 using until

```
#!/bin/sh
i=1
until [ $i -gt 10 ]
do
echo $i
i=`expr $i+1`
done
```

  As a rule the until must have a control command that will eventually return an exit status 0(true), otherwise the loop would be executed forever, indefinitely.

THE BREAK STATEMENT:

- We often come across situations where we want to jump out of a loop instantly, without waiting to get back to the control command. The keyword break allows us to do this. (or)

- Break is keyword, to terminate the loop. The form of break keyword is:

```
while [ condition ]
do
--------
--------
break
done
```

Example:

```
#!/bin/bash
#Break statement example
count=1
while [ $count -le 10 ]
do
if [ $count -eq 4 ]
then
break
fi
echo $count
count=`expr $count + 1`
done
```

THE CONTINUE STATEMENT:

- Continue is the keyword to start the loop again.
- The general form is:

```
while [ condition ]
do
-------
-------
continue
---------
---------
done
```

```
#!/bin/sh
#Example of Continue statement
count=0
while [ $count -le 9 ]
do
count=`expr $count + 1`
if [ $count -eq 5 ]
then
continue
fi
echo $count
done
echo "We are out of the loop now"
```

1. **Write a program accept a string and display reverse of the given string.**

```
#!/bin/sh
echo -n "Enter a string"
read string
l=`echo $string | wc -c`
while [ $string -ne 0 ]
do
ch=`echo $string | cut -c $l`
temp=$temp$ch
l=`expr $l-l`
done
echo "Reverse of $string is: $temp"
```

2. **Write a script accept file names and open:**

```
#!/bin/sh
answer="y"
while [ $ans = "y" ]
do
echo "Enter a filename to open:"
read fname
if [ -e $fname -a -f $fname ]
then
cat $fname
else
echo "No such file"
fi
```

```
echo "Do you want to open one more file [y/n]:"
read answer
done
```

Example of break and continue:

```
#!/bin/sh
while true #until false
do
echo "enter a file name to open:"
read fname
if [ -e $fname -a -f $fname ]
then
cat $fname
break
else
continue
fi
done
```

3. **Write a script create "n" number of users**

```
#!/bin/sh
echo -n "Enter number of users to create:"
read n
$i=1
while [ $i -le $n ]
do
$u=raju$i
useradd $u
i=`expr $i + 1`
done
```

Example 5:

```
#!/bin/sh
i=1 j=1
while [ $i -le 100 ]
do
while [ $j -lt 200 ]
do
if [ $j -eq 150 ]
then
break
else
echo $i $j
fi
```

```
j=`expr $j+1`
done
i=`expr $i+1`
done
```

4. Example:

```
#!/bin/sh
i=1
while [ $i -le 2 ]
do
j=1
while [ Sj -le 2 ]
do
if [ $i -eq $j ]
then
j=`expr $j+1`
continue
fi
j=`expr $j+1`
echo $i $j
done
i=`expr $i+1'
done
```

5. **Write a program when users log in**

```
#!/bin/sh
#When users log in
echo "Enter a username:\c"
read logname
time=0
while true
do
who | grep "$logname" > /dev/null
if [ $time -ne 0 ]
then
echo "$logname was $time miutes late in logged in."
fi
exit
else
time=`expr $time + 1`
sleep 60s
fi
```

```
done
```

SLEEP COMMAND:

- Delay for a specified amount of time.
- Pause for NUMBER seconds. SUFFIX may be ‘s’ for seconds (the default), ‘m’ for minutes, ‘h’ for hours or ‘d’ for days. Unlike most implementations that require NUMBER be an integer, here NUMBER may be an arbitrary floating point number. Given two or more arguments, pause for the amount of time specified by the sum of their values.

Example 1:

```
#!/bin/sh
#sleep example
echo "Enter a sentence:\c"
read str
for word in $str
do
echo $word
sleep 2
```

Example 2:

```
#!/bin/bash
x=10
while [ $x -gt 0 ]
do
sleep 1s
clear
echo "$x seconds until blast off"
x=$(( $x - 1 ))
done
```

THE FOR LOOP:

• The for loop is more frequently used as compared to the while and until loops. Its working is also different than the other two loops.
• The for allows us to specify a list of values which the control variable in the loop can take. The loop is then executed for each value mentioned in the list.
• The general form of for statement is:

```
for variable in val1 val2 val3.....valn
do
-----
-----
done
```

Example:

```
#!/bin/sh
for word in High on a hill
do
echo $word
done
```

```
for file in mydir/letters/*.let
for var in mydir/a??
for entry in ../../??.*
for var in * #files in the present working directory
for var in $* #command line arguments.
```

The control variables can take values from a shell variables:

```
name="India is a great"
for word $name
do
echo $word
done
```

Control variables can take values from the output of a command:

```
for cmd in `cat sample`
do
man $cmd >>helpfile
done
```

Write a program to display all empty files in the current directory:

```
#!/bin/sh
for i in *
if [ ! -s $i ]
then
echo $i #rm $i :To delete empty files
fi
```

Reverse string:

```
#!/bin/bash
input="$1"
reverse=""
len=${#input}
for (( i=$len-1; i>=0; i-- ))
do
reverse="$reverse${input:$i:1}"
done
echo "$reverse"
```

Nesting of Loops: The way if statements can be nested, similarly while, until and fors can also be nested.

Example: Demonstration of nested loops

```
#!/bin/sh
r=1
while [ $r -le 3 ]
do
c=1
while [ $c -le 2 ]
do
sum=`expr $r + $c`
echo r=$r c=$c sum=$sum
c=`expr $r + 1`
done
r=`expr $r+1`
done
```

This program will add 1+2+3+4+5 and result will be 15

```
#!/bin/sh
#will add 1+2+3+4+5
for i in 1 2 3 4 5
do
sum=`expr $sum + $i`
done
echo $sum
```

Implementing for loop with break statement

```
#!/bin/sh
#Implementing for loop with break statement
for a in 1 2 3 4 5 6 7 8 9 10
do
# if a is equal to 5 break the loop
if [ $a == 5 ]
then
break
fi
# Print the value
echo "Iteration no $a"
done
```

Iteration Program:

```
#!/bin/sh
#Implementing for loop with continue statement
for a in 1 2 3 4 5 6 7 8 9 10
do
# if a = 5 then continue the loop and
# don't move to line 8
if [ $a == 5 ]
then
continue
fi
echo "Iteration no $a"
done
```

Nameservers Program:

```
#!/bin/bash
for file in /etc/*
do
if [ "${file}" == "/etc/resolv.conf" ]
then
countNameservers=$(grep -c nameserver /etc/resolv.conf)
echo "Total  ${countNameservers} nameservers defined in ${file}"
break
fi
done
```