Method 1: Bash For Loop using “in” and list of values

Syntax:

```
for varname in list

do

 command1

 command2

 ..

done
```

In the above syntax:

- for, in, do and done are keywords
- “list” contains list of values. The list can be a variable that contains several words separated by spaces. If list is missing in the for statement, then it takes the positional parameter that were passed into the shell.
- varname is any Bash variable name.

In this form, the for statement executes the commands enclosed in a body, once for each item in the list. For example, if the list of values contains 5 items, the for loop will be executed a total of 5 times, once for each item in the list. The current item from the list will be stored in a variable “varname” each time through the loop. This “varname” can be processed in the body of the for loop.

Method 2: Bash For Loop using C like syntax

The second form of the for loop is similar to the for loop in “C” programming language, which has three expressions (initialization, condition and updation).

```
for (( expr1; expr2; expr3 ))

do

 command1

 command2

 ..

done
```

In the above bash for command syntax,

- Before the first iteration, expr1 is evaluated. This is usually used to initialize variables for the loop.
- All the statements between do and done are executed repeatedly until the value of expr2 is TRUE.
- After each iteration of the loop, expr3 is evaluated. This is usually used to increment a loop counter.

The following 12 examples shows how to bash for loops in different ways.

1. Static values for the list after “in” keyword

In the following example, the list of values (Mon, Tue, Wed, Thu and Fri) are directly given after the keyword “in” in the bash for loop.

```
$ cat for1.sh

i=1
```

```
for day in Mon Tue Wed Thu Fri

do

 echo "Weekday $((i++)) : $day"

done



$ ./for1.sh

Weekday 1 : Mon

Weekday 2 : Tue

Weekday 3 : Wed

Weekday 4 : Thu

Weekday 5 : Fri
```

Caution: The list of values should not be separated by comma (Mon, Tue, Wed, Thu, Fri). The comma will be treated as part of the value. i.e Instead of "Mon", it will use "Mon," as value as shown in the example below.

```
$ cat for1-wrong1.sh

i=1

for day in Mon, Tue, Wed, Thu, Fri

do

 echo "Weekday $((i++)) : $day"

done



$ ./for1-wrong1.sh

Weekday 1 : Mon,

Weekday 2 : Tue,

Weekday 3 : Wed,

Weekday 4 : Thu,

Weekday 5 : Fri
```

Caution: The list of values should not be enclosed in a double quote. ("Mon Tue Wed Thu Fri"). If you enclose in double quote, it will be treated as a single value (instead of 5 different values), as shown in the example below.

```
$ cat for1-wrong2.sh

i=1

for day in "Mon Tue Wed Thu Fri"

do

 echo "Weekday $((i++)) : $day"

done
```

```
$ ./for1-wrong2.sh

Weekday 1 : Mon Tue Wed Thu Fri
```

2. Variable for the list after “in” keyword

Instead of providing the values directly in the for loop, you can store the values in a variable, and use the variable in the for loop after the “in” keyword, as shown in the following example.

```
$ cat for2.sh

i=1

weekdays="Mon Tue Wed Thu Fri"

for day in $weekdays

do

 echo "Weekday $((i++)) : $day"

done



$ ./for2.sh

Weekday 1 : Mon

Weekday 2 : Tue

Weekday 3 : Wed

Weekday 4 : Thu

Weekday 5 : Fri
```

Caution: As a best practice, you should always quote the bash variables when you are referring it. There are few exceptions to this best practice rule. This is one of them. If you double quote the variable in this for loop, the list of values will be treated as single value. Lot of people fall into this trap. Be careful and do not double quote your variable in the for loop.

```
$ cat for2-wrong.sh

i=1

weekdays="Mon Tue Wed Thu Fri"

for day in "$weekdays"

do

 echo "Weekday $((i++)) : $day"

done



$ ./for2-wrong.sh

Weekday 1 : Mon Tue Wed Thu Fri
```

3. Don’t specify the list; get it from the positional parameters

If you don’t specify the keyword “in” followed by any list of values in the bash for loop, it will use the positional parameters (i.e the arguments that are passed to the shell script).

```
$ cat for3.sh

i=1

for day

do

 echo "Weekday $((i++)) : $day"

done



$ ./for3.sh Mon Tue Wed Thu Fri

Weekday 1 : Mon

Weekday 2 : Tue

Weekday 3 : Wed

Weekday 4 : Thu

Weekday 5 : Fri
```

Caution: Please be careful if you use this method. You should not include the keyword “in” in the for loop. If you leave the keyword “in” without any values, it will not use the positional parameter as shown below. It will not go inside the loop. i.e for loop will never get executed as shown in the example below.

```
$ cat for3-wrong.sh

i=1

for day in

do

 echo "Weekday $((i++)) : $day"

done



$ ./for3-wrong.sh Mon Tue Wed Thu Fri
```

Note: Refer to our earlier article to understand more about bash positional parameters.

4. Unix command output as list values after “in” keyword

You can use the output of any UNIX / Linux command as list of values to the for loop by enclosing the command in back-ticks ` ` as shown below.

```
$ cat for4.sh

i=1

for username in `awk -F: '{print $1}' /etc/passwd`
```

```
do

 echo "Username $((i++)) : $username"

done



$ ./for4.sh

Username 1 : ramesh

Username 2 : john

Username 3 : preeti

Username 4 : jason

..
```

5. Loop through files and directories in a for loop

To loop through files and directories under a specific directory, just cd to that directory, and give * in the for loop as shown below.

The following example will loop through all the files and directories under your home directory.

```
$ cat for5.sh

i=1

cd ~

for item in *

do

 echo "Item $((i++)) : $item"

done



$ ./for5.sh

Item 1 : positional-parameters.sh

Item 2 : backup.sh

Item 3 : emp-report.awk

Item 4 : item-list.sed

Item 5 : employee.db

Item 8 : storage

Item 9 : downloads
```

Usage of * in the bash for loop is similar to the file globbing that we use in the linux command line when we use ls command (and other commands).

For example, the following will display all the files and directories under your home directory. This is the concept that is used in the above for5.sh example.

```
cd ~

ls *
```

The following will display all the *.conf file that begins with either a, b, or, c or d under /etc directory.

```
$ ls -1 /etc/[abcd]*.conf

/etc/asound.conf

/etc/autofs_ldap_auth.conf

/etc/cas.conf

/etc/cgconfig.conf

/etc/cgrules.conf

/etc/dracut.conf
```

The same argument that is used in the ls command above, can be used in a bash for loop, as shown in the example below.

```
$ cat for5-1.sh

i=1

for file in /etc/[abcd]*.conf

do

 echo "File $((i++)) : $file"

done



$ ./for5-1.sh

File 1 : /etc/asound.conf

File 2 : /etc/autofs_ldap_auth.conf

File 3 : /etc/cas.conf

File 4 : /etc/cgconfig.conf

File 5 : /etc/cgrules.conf

File 6 : /etc/dracut.conf
```

6. Break out of the for loop

You can break out of a for loop using ‘break’ command as shown below.

```
$ cat for6.sh

i=1

for day in Mon Tue Wed Thu Fri

do
```

```
 echo "Weekday $((i++)) : $day"

 if [ $i -eq 3 ]; then

   break;

 fi

done



$ ./for6.sh

Weekday 1 : Mon

Weekday 2 : Tue
```

7. Continue from the top of the for loop

Under certain conditions, you can ignore the rest of the commands in the for loop, and continue the loop from the top again (for the next value in the list), using the continue command as shown below.

The following example adds "(WEEKEND)" to Sat and Sun, and "(weekday)" to rest of the days.

```
$ cat for7.sh

i=1

for day in Mon Tue Wed Thu Fri Sat Sun

do

 echo -n "Day $((i++)) : $day"

 if [ $i -eq 7 -o $i -eq 8 ]; then

   echo " (WEEKEND)"

   continue;

 fi

 echo " (weekday)"

done



$ ./for7.sh

Day 1 : Mon (weekday)

Day 2 : Tue (weekday)

Day 3 : Wed (weekday)

Day 4 : Thu (weekday)

Day 5 : Fri (weekday)

Day 6 : Sat (WEEKEND)
Day 7 : Sun (WEEKEND)
```

8. Bash for loop using C program syntax

This example uses the 2nd method of bash for loop, which is similar to the C for loop syntax. The following example generates 5 random number using the bash C-style for loop.

```
$ cat for8.sh

for (( i=1; i <= 5; i++ ))

do

 echo "Random number $i: $RANDOM"

done



$ ./for8.sh

Random number 1: 23320

Random number 2: 5070

Random number 3: 15202

Random number 4: 23861

Random number 5: 23435
```

9. Infinite Bash for loop

When you don't provide the start, condition, and increment in the bash C-style for loop, it will become infinite loop. You need to press Ctrl-C to stop the loop.

```
$ cat for9.sh

i=1;

for (( ; ; ))

do

   sleep $i

   echo "Number: $((i++))"

done
```

Like we said above, press Ctrl-C to break out of this bash infinite for loop example.

```
$ ./for9.sh

Number: 1

Number: 2
```

```
Number: 3
```

10. Using comma in the bash C-style for loop

In the bash c-style loop, apart from increment the value that is used in the condition, you can also increment some other value as shown below.

In the initialize section, and the increment section of the bash C-style for loop, you can have multiple value by separating with comma as shown below.

The following for loop is executed a total of 5 times, using the variable i. However the variable j start with 5, and getting increment by 5 every time the loop gets executed.

```
$ cat for10.sh

for ((i=1, j=10; i <= 5 ; i++, j=j+5))

do

 echo "Number $i: $j"

done



$ ./for10.sh

Number 1: 10

Number 2: 15

Number 3: 20

Number 4: 25

Number 5: 30
```

11. Range of numbers after “in” keyword

You can loop through using range of numbers in the for loop “in” using brace expansion.

The following example loops through 10 times using the values 1 through 10.

```
$ cat for11.sh

for num in {1..10}

do

 echo "Number: $num"

done



$ ./for11.sh

Number: 1

Number: 2
```

```
Number: 3

Number: 4

Number: 5

...
```

12. Range of numbers with increments after “in” keyword

The following example loops through 5 times using the values 1 through 10, with an increment of 2. i.e It starts with 1, and keeps incrementing by 2, until it reaches 10.

```
$ cat for12.sh

for num in {1..10..2}

do

 echo "Number: $num"

done



$ ./for12.sh

Number: 1

Number: 3

Number: 5

Number: 7

Number: 9
```