

How to Use For loops in Bash

Learn how to effectively use for loops in Bash scripting to automate repetitive tasks, iterate through files, and enhance your command line productivity.

Contents

01	Introduction	3
02	forLoop Bash Syntax	3
03	CreateforLoops in Bash	4
04	Create InfiniteforLoops	6
05	Create NestedforLoops	7
06	Use Break inforLoops	8
07	Use Continue inforloops.	9
08	Use Command Substitution inforLoops	10
09	Use Arrays inforLoops	10
010	Conclusion	11

Introduction

`for` loops in Bash execute commands by iterating through a list of items repeatedly before generating a final result. You can use `for` loops to process files and directories or even to perform batch operations such as automatic backups.

This article explains how to use `for` loops in Bash.

`for` Loop Bash Syntax

A `for` loop statement executes a specific set of commands for each item in a list. Use `for` loops to iterate over a finite collection of items, such as files or strings using the following syntax.

```
for variable in list
do
    # Commands to execute for each item in the list
done
```

In the above `for` loop structure, the variable takes each value from the list and executes the commands between the `do` and `done` body section.

- `for`: Starts the loop.
- `variable`: Stores the current item from the list during each iteration.
- `in`: Specifies the list of items to iterate over.
- `list`: Sets the list of items.
- `do`: Marks the start of the loop's body and executes the specified commands.
- `done`: Marks the end of the loop.

Create `for` Loops in Bash

`for` loops work with variables or lists. The loop executes once for each item in the list. A `for` loop statement stops when it iterates over all items in the list. For example, if a list contains three files, the `for` loop runs three times before stopping. Follow the steps below to create a `for` loop statement.

1. Create a `for` loop that reads data from a variable and iterates the list to output a basic message using the `echo` command.

```
BASH

#!/bin/bash

SERVERS="Server-1 Server-2 Server-3"
for S in $SERVERS; do
    echo "Updating the packages on: $S"
done
```

Output:

```
Updating the packages on: Server-1
Updating the packages on: Server-2
Updating the packages on: Server-3
```

The above script uses a `for` loop to iterate over three server names and outputs the `Updating the packages on:` message using the current list item's value. For example, on the first iteration, the loop outputs the `Updating the packages on: Server-1` message.

2. Create a `for` loop that iterates over a range of numbers.

```
BASH

#!/bin/bash

for value in {1..20}
do
```

```
    echo "Number: $value"  
done
```

Output:

```
Number: 1  
Number: 2  
Number: 3  
Number: 4  
Number: 5  
Number: 6  
Number: 7  
Number: 8  
Number: 9  
Number: 10  
Number: 11  
Number: 12  
Number: 13  
Number: 14  
Number: 15  
Number: 16  
Number: 17  
Number: 18  
Number: 19  
Number: 20
```

The above script outputs a list of numbers from `1` to `20` before stopping. The loop updates the `value` on each iteration with a new number in the brace expansion list. This syntax is useful when processing a list of numbers, files, or repeating tasks in a loop.

3. Create a `for` loop in bash script that iterates over multiple files in a directory such as `/var/www/html` and updates the file permissions to `755`.

BASH

```
#!/bin/bash  
  
for file in /var/www/html/*  
do  
    sudo chmod 755 "$file"
```

```
    echo "Updated permissions for: $file"  
done
```

Output:

```
Updated permissions for: /var/www/html/index.html  
Updated permissions for: /var/www/html/index.php
```

The above script sets a new value for the `file` variable by retrieving the list of files from the `/var/www/html` directory. The script then updates the permissions of each file to `755` and outputs the `Updated permissions for:` message. For example, if an `index.html` file exists in the directory, the script sets its permission mode to `755` and outputs the `Updated permissions for: index.html` message and proceeds to execute the next `file` in the iteration.

Create Infinite `for` Loops

Infinite `for` loops in Bash are control structures that run continuously unless an external condition interrupts the execution flow. Infinite loops are suitable for indefinite tasks such as real-time server monitoring, automatic background processes, and applications that require continuous updates.

BASH

```
#!/bin/bash  
  
for (( ; ; ))  
do  
    echo "This is an Infinite loop [ use CTRL+C to stop it]"  
done
```

Output:

```
This is an Infinite loop [ use CTRL+C to stop it]  
This is an Infinite loop [ use CTRL+C to stop it]  
This is an Infinite loop [ use CTRL+C to stop it]  
This is an Infinite loop [ use CTRL+C to stop it]  
This is an Infinite loop [ use CTRL+C to stop it]  
This is an Infinite loop [ use CTRL+C to stop it]
```

```
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This is an Infinite loop [ use CTRL+C to stop it]
This i^C
```

The above `for loop` in bash script creates an infinite loop that executes the `echo` command continuously and outputs the `This is an Infinite loop [use CTRL+C to stop it]` message. The `for ((; ;))` declaration defines the start of an infinite loop.

Create Nested `for` Loops

Nested `for` loops define advanced functionalities by allowing you to place a `for` loop statement inside another loop. Nested loops are useful when iterating multiple data dimensions to process a combination of items.

BASH

```
#!/bin/bash

for server_id in A B C; do
    for app in apache mysql php; do
        echo "Server: $server_id can run the $app LAMP package"
    done
done
```

Output:

```
Server: A can run the apache LAMP package
Server: A can run the mysql LAMP package
```

```
Server: A can run the php LAMP package
Server: B can run the apache LAMP package
Server: B can run the mysql LAMP package
Server: B can run the php LAMP package
Server: C can run the apache LAMP package
Server: C can run the mysql LAMP package
Server: C can run the php LAMP package
```

The above script uses a nested `for` loop to iterate over two lists and output a result. The variable `$server` takes a new value from the outer loop list `A B C` on each iteration while the variable `$app` takes a new result from the inner loop list `apache mysql php`. The output combines the current inner loop and outer loop values to output a message. For example, the first iteration outputs the `Server: A can run the apache LAMP package` message.

Use Break in `for` Loops

A `break` exits a `for` loop statement prematurely after meeting a specific condition to skip the remaining iterations. For example, the following `for` loop statement iterates through files in the `/etc` directory and uses a conditional `if` statement to test if a list item's value matches the `timezone` to stop the loop.

BASH

```
#!/bin/bash

for file in /etc/*; do
    if [[ "$file" == "/etc/timezone" ]];
    then
        echo "$file is available, stopping the loop"
        break
    fi
done
```

Output:

```
/etc/timezone is available, stopping the loop
```

Use Continue in `for` loops.

A `continue` statement in a `for` loop skips any remaining commands in the current iteration and runs the next iteration in the loop. For example, the following `for` loop iterates through numbers 1 to 20 and uses a conditional `if` statement to test if the current value is equal to `5`. If the condition is true, it moves on to the next iteration without executing any other commands.

BASH

```
#!/bin/bash

for value in {1..20}
do
    if [[ $value -eq 5 ]]
    then
        echo "skipping the number 5"
        continue
    fi
    echo "The current value is equal to $value"
done
```

Output:

```
The current value is equal to 1
The current value is equal to 2
The current value is equal to 3
The current value is equal to 4
skipping the number 5
The current value is equal to 6
The current value is equal to 7
The current value is equal to 8
The current value is equal to 9
```

Use Command Substitution in `for` Loops

In a `for` loop statement, you can use command substitution to store and redirect the output of a specific command to a new variable for further processing. For example, when working with files and directories. The following `for` loop iterates through whitespace-separated entries in the `/var/log/syslog` file and outputs a `Log entry:` message with each entry.

```
BASH
```

```
#!/bin/bash

for log in $(cat /var/log/syslog); do
    echo "Log entry: $log"
done
```

Output:

```
Log entry: Main
Log entry: User
Log entry: Target.
Log entry: Sep
Log entry: 15
Log entry: 23:37:20
Log entry: Lets-Connect
Log entry: systemd[803616]:
Log entry: Startup
Log entry: finished
Log entry: in
Log entry: 137ms.
```

Use Arrays in `for` Loops

You can iterate through array elements using `for` loops. In Bash, an array is an indexed collection of values you can expand. For example, the following `for` loop expands the `app_names` array and prints each element.

```
BASH
```

```
#!/bin/bash

app_names=("apache" "mysql" "php" "nginx")

for app in "${app_names[@]}"
do
    echo "The application name is: $app"
done
```

Output:

```
The application name is: apache
The application name is: mysql
The application name is: php
The application name is: nginx
```

Conclusion

In this article you used `for` loops in Bash to iterate over lists. You have also run infinite loops, nested loops, break statements, continue statements, command substitution, and expanded arrays using `for` loops. Loops automate repetitive tasks such as monitoring files or filtering results in a large list.



VULTR

