

🕒 Sunday, March 05, 2017



LIKE GEEKS



LINUX

Bash Scripting The Awesome Guide Part2

📅 February 9, 2017 👤 admin 💬 4 Comments

In the previous post, we talked about how to write a **bash script**. And we've seen how **bash scripting** is awesome. In this post, we continue to look at structured commands that control the flow of your shell scripts. You'll see how you can perform repeating processes; this post demonstrates for loop, while in **bash scripts**

we will discuss the following:

for command

Iterating over simple values

Iterating over complex values

Reading values from a command

The field separator

Iterating over directory files

for Command C-Style

The while Command

Nesting Loops

Looping on File Data

Controlling the Loop

The break command

The continue command

Processing the Output of a Loop

Useful Examples

for Command

The bash shell provides the for command to allow you to create a loop that iterates through a series of values. This is the basic format of the bash shell for command

```
for var in list
```

```
do
```

commands

done

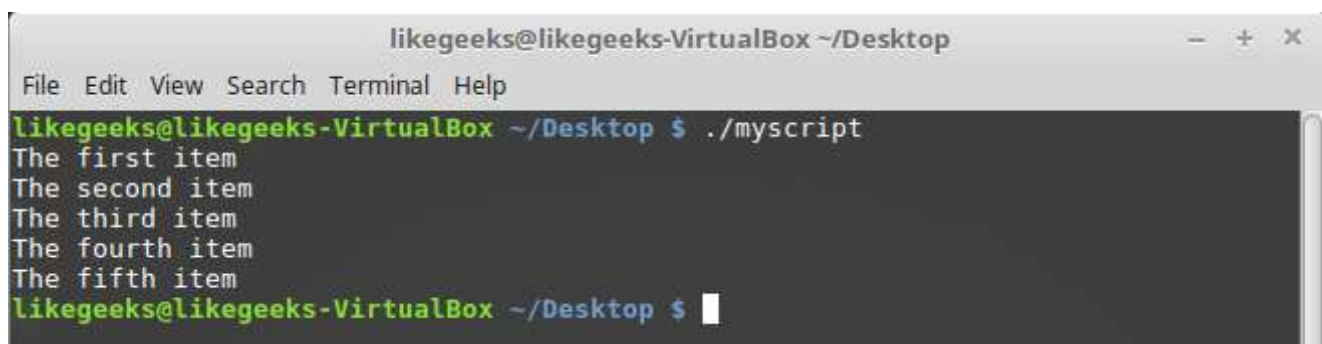
In each iteration, the variable *var* contains the current value in the list. The first iteration uses the first item in the list; the second iteration contains the second item, and so on until the end of the list items

Iterating over simple values

The most basic use of the `for` command in bash scripting is to iterate through a list of simple values like this

```
#!/bin/bash
for var in first second third fourth fifth
do
echo The $var item
done
```

As you can see from the output the `$var` variable is changed on every loop cycle till the last item on the list.

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the command './myscript' being executed, which outputs five lines: 'The first item', 'The second item', 'The third item', 'The fourth item', and 'The fifth item'. The prompt 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' is visible at the bottom.

Notice that the `$var` variable retained its value and allowed us to change the value and use it outside of the `for` command loop, like any variable.

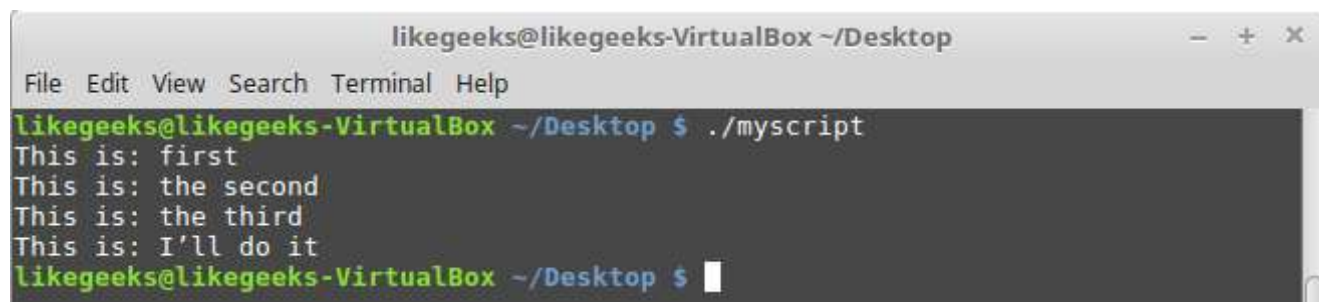
Iterating over complex values

Your list maybe contains some comma or two words but you want to deal with them as one item on the list.

This example we have some of those

```
#!/bin/bash
for var in first "the second" "the third" "I'll do it"
do
echo "This is: $var"
done
```

We play nice till now, always we do. Just keep reading and practicing.

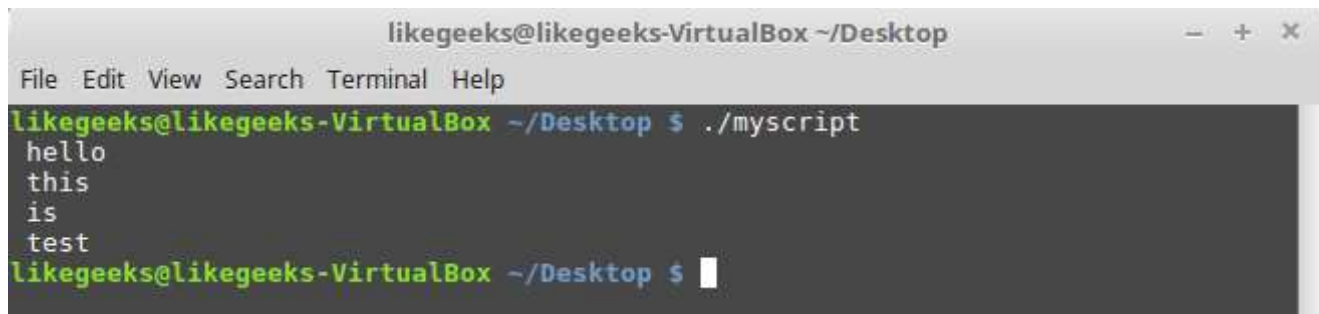
A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the command './myscript' being executed, which outputs four lines: 'This is: first', 'This is: the second', 'This is: the third', and 'This is: I'll do it'. The prompt 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' is visible at the bottom.

Reading values from a command

Another way to a list is to use the output of a command. You use command substitution to execute any command that produces output.

```
#!/bin/bash
file="myfile"
for var in $(cat $file)
do
echo " $var"
done
```

This example uses the cat command in the command substitution to display the contents of the file states. Notice that our file contains one word per line, not separated by spaces.

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows a prompt 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' followed by the command './myscript'. The output of the script is displayed on four lines: 'hello', 'this', 'is', and 'test'. The prompt returns to 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' with a cursor.

Notice that our file contains one word per line, not separated by spaces.

The for command still iterates through the output of the cat command one line at a time, assuming that each line has one word. However, this doesn't solve the problem of having spaces in data.

If you list that contains words with spaces in it, the for command still takes each word as a separate value. There's a reason for this, which we look at now.

The field separator

The cause of this problem is the special environment variable IFS, called the internal field separator. By default, the bash shell considers the following characters as field

- Space
- Tab
- newline

If the bash shell sees any of these characters in the data, it assumes that you're starting a new data field in the list.

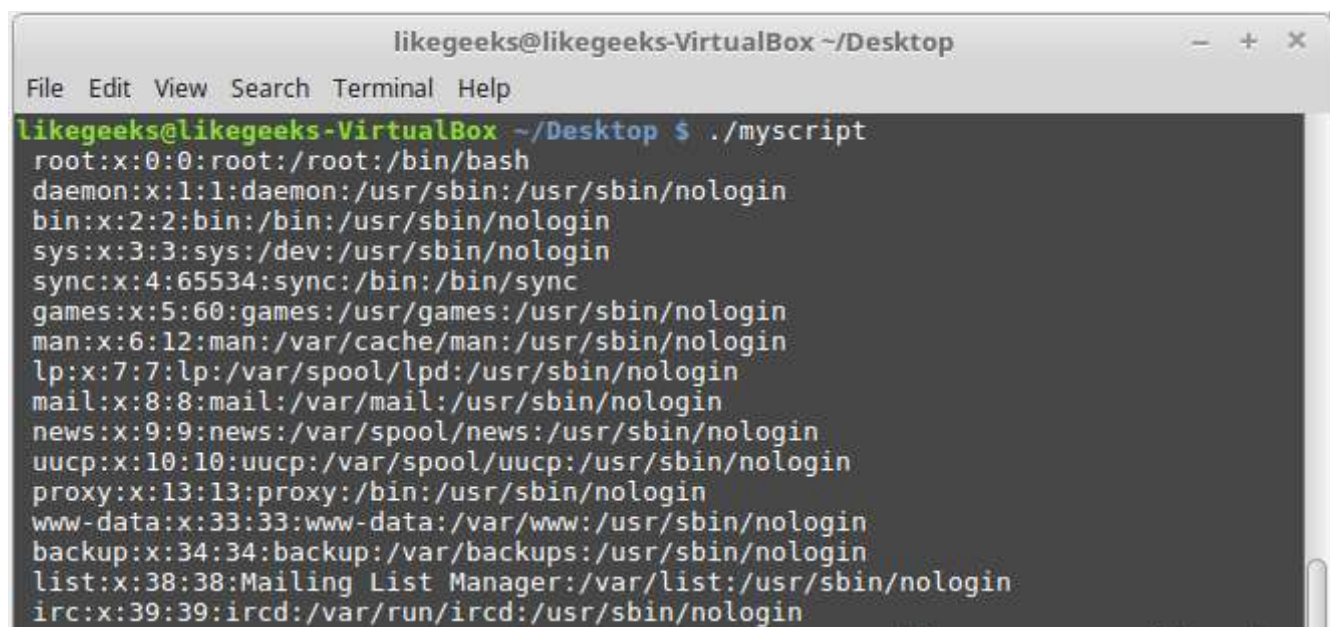
To solve this problem, you can temporarily change the IFS environment variable values in your bash script suppose that you want to separate by new lines so it will be like this

```
IFS=$'\n'
```

So after you add this to your bash script it will ignore spaces and tabs and consider new lines as a separator.

```
#!/bin/bash
file="/etc/passwd"
IFS=$'\n'
for var in $(cat $file)
do
echo " $var"
done
```

You got it. Bash scripting is easy just little attention

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The terminal shows the command './myscript' being executed. The output lists system users and their home directories, separated by newlines, matching the script's logic. The users listed are: root, daemon, bin, sys, sync, games, man, lp, mail, news, uucp, proxy, www-data, backup, list, and irc.

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
```

In this case, the separator is colon like the case of /etc/passwd file which contains the user's information you can assign it like this

```
IFS=:
```

How bash scripting is awesome?

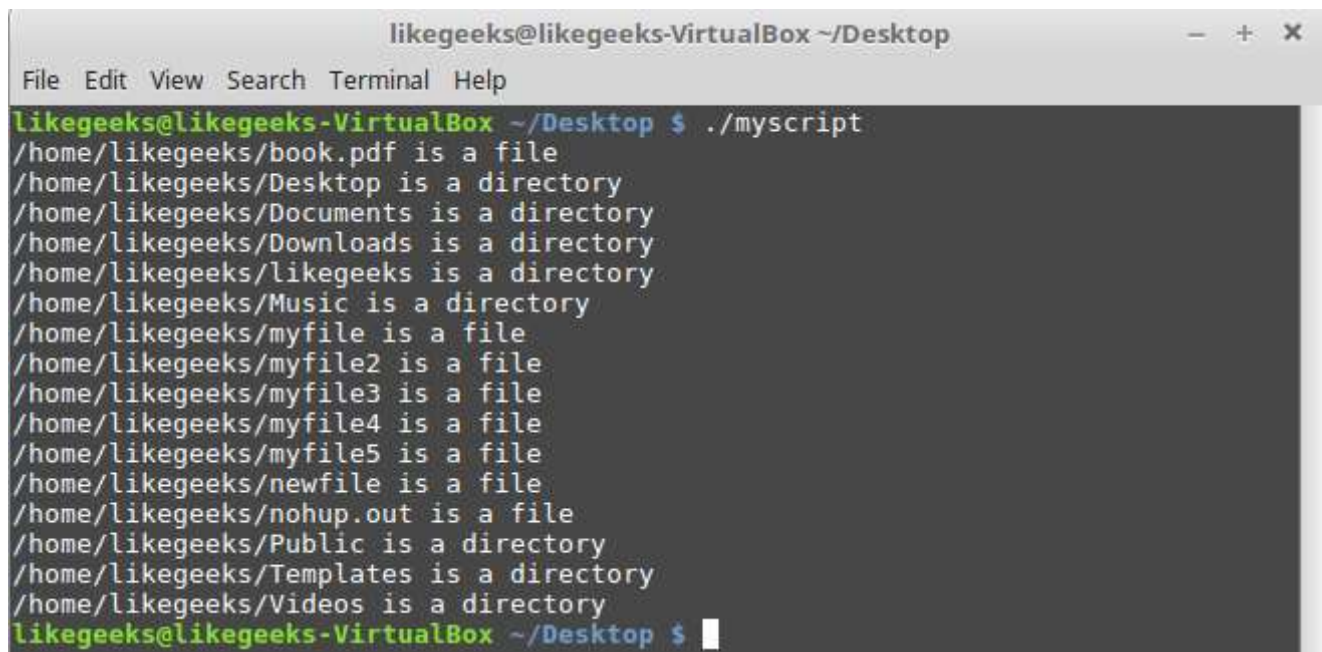
Iterating over directory files

One of the most common things when using for loop in bash scripting is iterate over files in a directory and deal with them.

For example, we want to list the file inside /home directory so the code will be like this

```
#!/bin/bash
for file in /home/likegeeks/*
do
if [ -d "$file" ]
then
echo "$file is a directory"
elif [ -f "$file" ]
then
echo "$file is a file"
fi
done
```

From the previous post, you should know the if statement and how to differentiate between files and folders, so if you don't know I recommend you to review it [bash script step by step](#).

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The terminal shows the execution of a script named './myscript'. The output of the script lists various files and directories in the /home/likegeeks/ directory, identifying them as either files or directories. The files listed are book.pdf, myfile, myfile2, myfile3, myfile4, myfile5, newfile, and nohup.out. The directories listed are Desktop, Documents, Downloads, likegeeks, Music, Public, Templates, and Videos. The prompt at the bottom shows the user is still in the ~/Desktop directory.

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
/home/likegeeks/book.pdf is a file
/home/likegeeks/Desktop is a directory
/home/likegeeks/Documents is a directory
/home/likegeeks/Downloads is a directory
/home/likegeeks/likegeeks is a directory
/home/likegeeks/Music is a directory
/home/likegeeks/myfile is a file
/home/likegeeks/myfile2 is a file
/home/likegeeks/myfile3 is a file
/home/likegeeks/myfile4 is a file
/home/likegeeks/myfile5 is a file
/home/likegeeks/newfile is a file
/home/likegeeks/nohup.out is a file
/home/likegeeks/Public is a directory
/home/likegeeks/Templates is a directory
/home/likegeeks/Videos is a directory
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Here we use wildcard character which is asterisk * and this is called in bash scripting **file globbing** which is a process of producing filenames automatically that matches the wildcard character in our case asterisk means All files with all names

Notice that in the if statements here we quote our variables with quotations because maybe the file or the folder name contains spaces

Sure enough, it lists all files and directories in that folder

for Command C-Style

If you know c language you may found that the for loop here is weird because you are familiar with this syntax

```
for (i = 0; i < 10; i++)  
  
{  
  
printf("number is %d\n", i);  
  
}
```

The bash scripting also supports a version of the for loop that looks similar to the C-style for loop with little difference here's the syntax.

```
for (( variable assignment ; condition ; iteration process ))
```

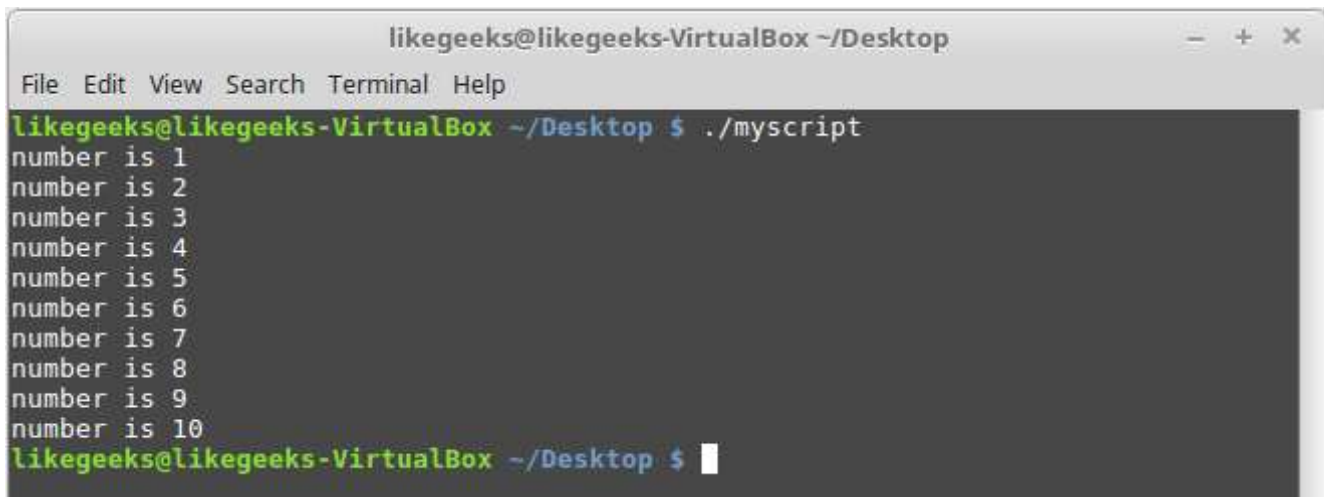
So it looks like this

```
for (( a = 1; a < 10; a++ ))
```

And if you want to write in that style go ahead and do that

```
#!/bin/bash  
for (( i=1; i <= 10; i++ ))  
do  
echo "number is $i"  
done
```


And this is the output

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the command './myscript' being executed, which outputs ten lines: 'number is 1' through 'number is 10'. The prompt 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' is visible at the bottom.

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
number is 1
number is 2
number is 3
number is 4
number is 5
number is 6
number is 7
number is 8
number is 9
number is 10
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

The while Command

The for loop is not the only way of looping in bash scripting. The while command allows you to define a command to test and then loop through a set of commands as long as the defined test command returns a zero exit status which means success. It tests the test command at the start of each iteration. When the test command returns a nonzero exit status means fail, the while command stops executing the commands

And this is the format of while loop command

while *test* command

do

other commands

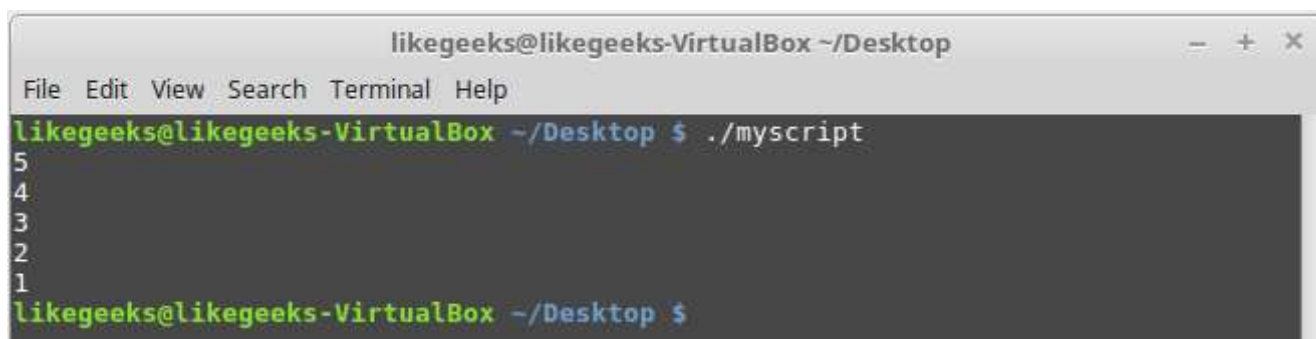
done

and here is an example

```
#!/bin/bash
var1=5
while [ $var1 -gt 0 ]
```

```
do
echo $var1
var1=$(( $var1 - 1 ))
done
```

The script is simple; it starts with while command to check if var1 is greater than zero then the loop will run and the var1 value will be decreased every time by 1 and on every loop iteration it will print the value of var1, Once the var1 value is zero the loop will exit.

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the command './myscript' being executed, which outputs the numbers 5, 4, 3, 2, and 1 on separate lines. The prompt then returns to the shell.

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
5
4
3
2
1
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

If we don't decrease the value of var1 it will be the same value and the loop will be infinite.

Nesting Loops

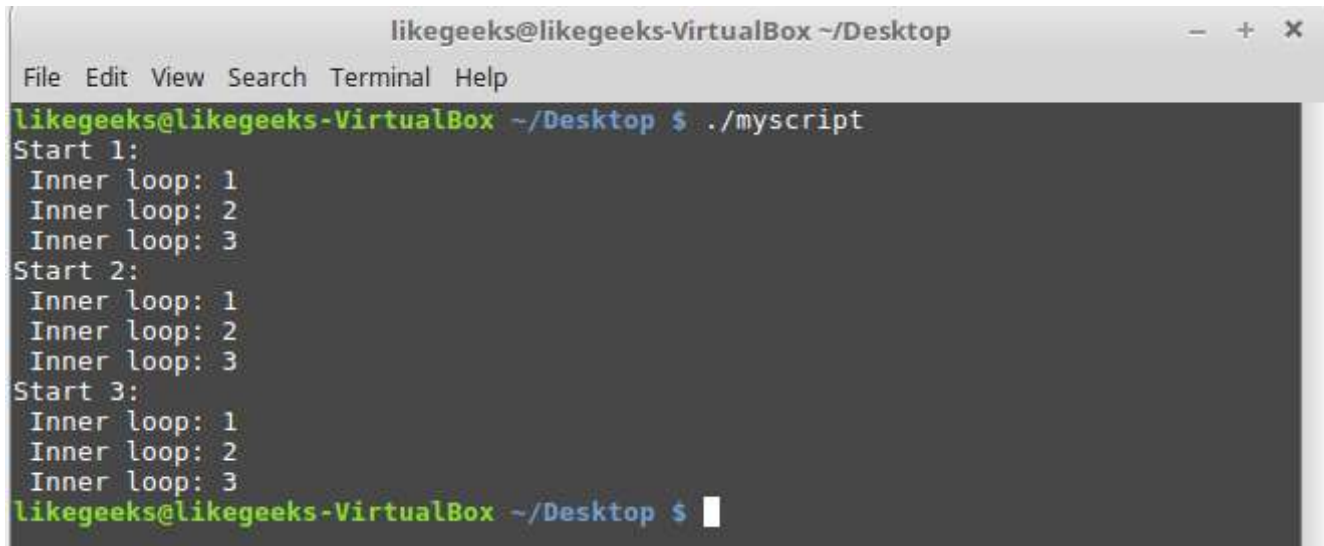
A loop statement can use any other type of command within the loop, including other loop commands. This is called a nested loop.

Here's an example of nested loops.

```
#!/bin/bash
for (( a = 1; a <= 3; a++ ))
do
echo "Start $a:"
for (( b = 1; b <= 3; b++ ))
do
echo " Inner loop: $b"
done
done
```

done

As you can see from the results the outer loop hits first then goes into the inner loop and completes it and go back to the outer loop and so on.

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The terminal shows the execution of a script named 'myscript'. The output consists of three main sections, each starting with 'Start X:' followed by three lines of 'Inner loop: Y'. The first section is 'Start 1:' with inner loops 1, 2, and 3. The second is 'Start 2:' with inner loops 1, 2, and 3. The third is 'Start 3:' with inner loops 1, 2, and 3. The prompt 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' is visible at the bottom.

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Start 1:
Inner loop: 1
Inner loop: 2
Inner loop: 3
Start 2:
Inner loop: 1
Inner loop: 2
Inner loop: 3
Start 3:
Inner loop: 1
Inner loop: 2
Inner loop: 3
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

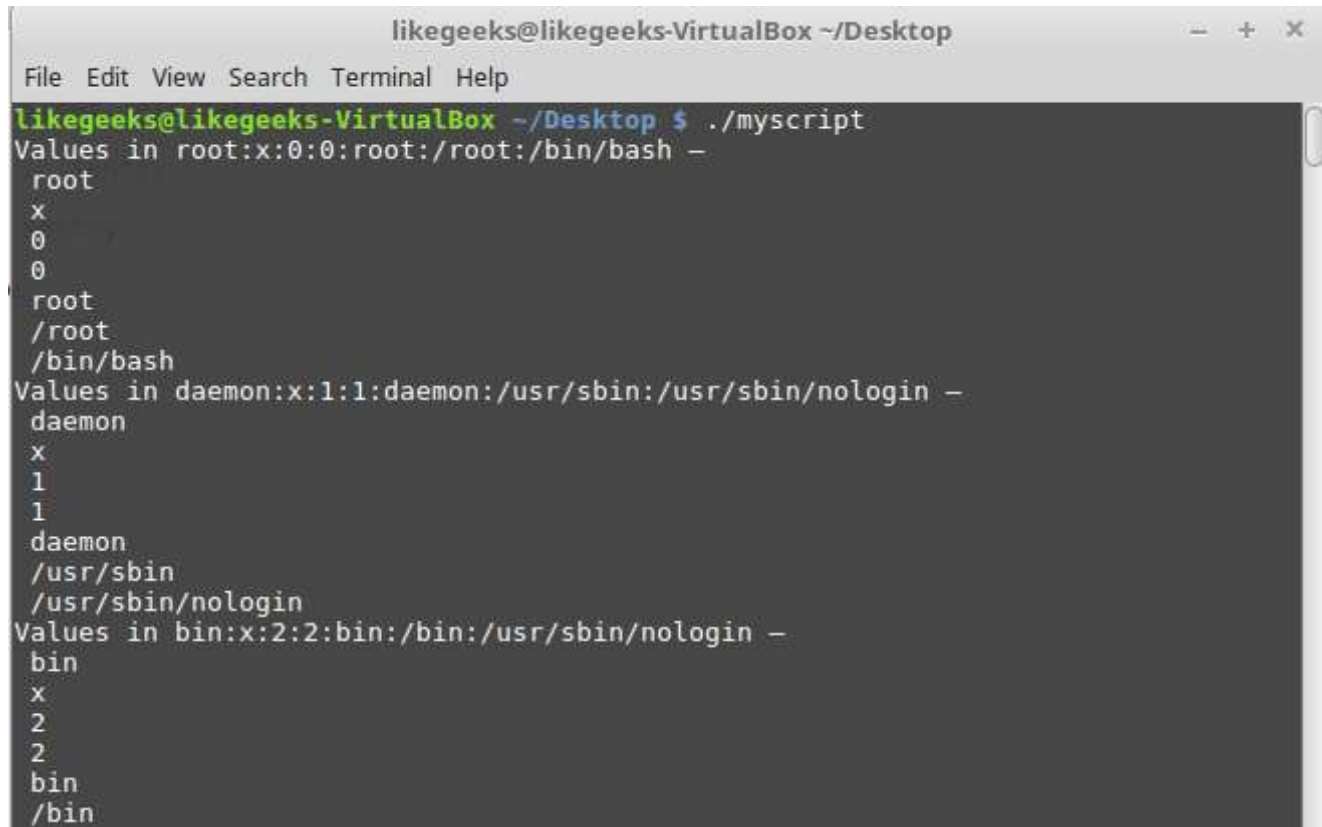
Looping on File Data

This is the most common usage for the for loop in bash scripting.

We can iterate over file content for example /etc/passwd file and see what we will get

```
#!/bin/bash
IFS=$'\n'
for entry in $(cat /etc/passwd)
do
echo "Values in $entry -"
IFS=:
for value in $entry
do
echo " $value"
done
done
```

Here we have two loops, the first loop iterate over the lines of the file and the separator is the newline, the second iteration is over the words on the line itself and the separator is the colon :

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The window shows the execution of a script './myscript'. The output is divided into three sections, each starting with a header line. The first section is 'Values in root:x:0:0:root:/root:/bin/bash -', followed by four lines of output: 'root', 'x', '0', and '0'. The second section is 'Values in daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin -', followed by four lines of output: 'daemon', 'x', '1', and '1'. The third section is 'Values in bin:x:2:2:bin:/bin:/usr/sbin/nologin -', followed by four lines of output: 'bin', 'x', '2', and '2'. The terminal has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'.

You can apply this idea when you have CSV or any comma separated values file or whatever. The idea is the same; you just will change the separator to fit your needs.

Controlling the Loop

Maybe after the loop starts you want to stop at a specific value, will you wait until the loop is finished? Of course not, there are two commands help us in this.

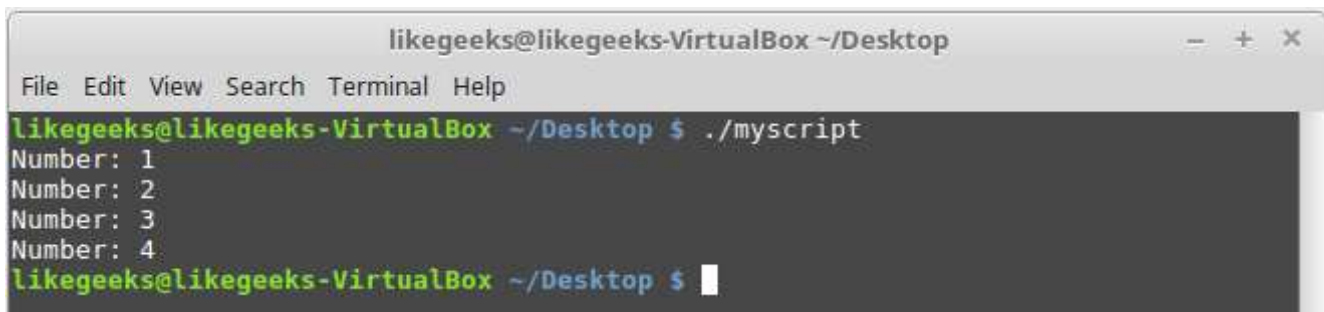
- break command
- continue command

The break command

The break command is a simple way to escape a loop. You can use the break command to exit any type of loop, including while and until loops.

```
#!/bin/bash
for var1 in 1 2 3 4 5 6 7 8 9 10
do
if [ $var1 -eq 5 ]
then
break
fi
echo "Number: $var1"
done
```

The for loop should normally have iterated through all the values in the list.



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Number: 1
Number: 2
Number: 3
Number: 4
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

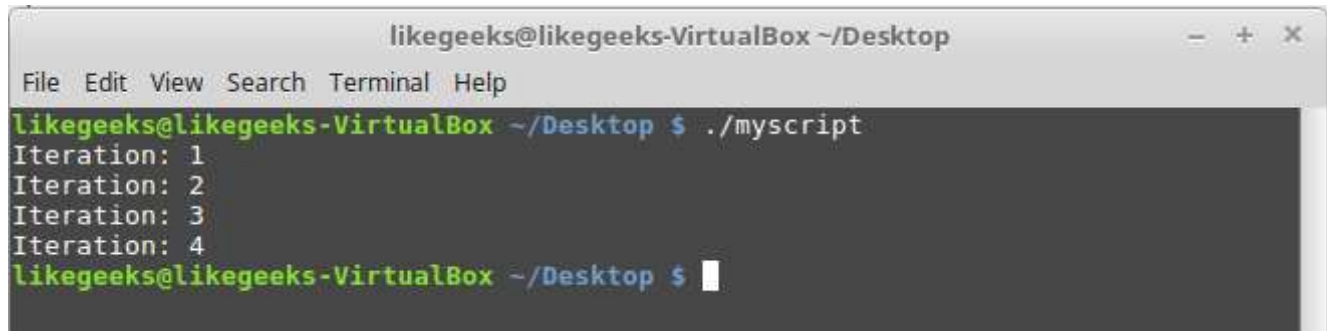
However, the shell executed the break command, which stopped the for loop

And the same for the while loop

```
#!/bin/bash
var1=1
while [ $var1 -lt 10 ]
do
if [ $var1 -eq 5 ]
then
break
fi
echo "Iteration: $var1"
```

```
var1=$(( $var1 + 1 ))  
done
```

The while loop terminated when the if-then condition was met, executing the break command.

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the command './myscript' being executed, which outputs 'Iteration: 1', 'Iteration: 2', 'Iteration: 3', and 'Iteration: 4'. The prompt then returns to the shell: 'likegeeks@likegeeks-VirtualBox ~/Desktop \$'.

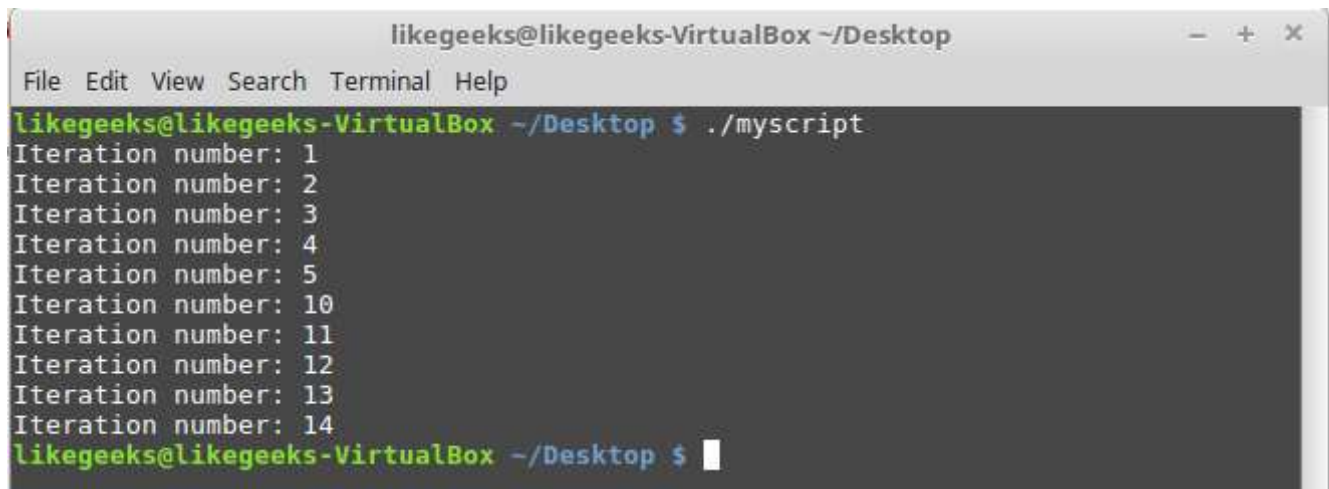
The continue command

The continue command is a way to prematurely stop processing commands inside of a loop but not terminate the loop completely.

Here's a simple example of using the continue command in a for loop

```
#!/bin/bash  
for (( var1 = 1; var1 < 15; var1++ ))  
do  
if [ $var1 -gt 5 ] && [ $var1 -lt 10 ]  
then  
continue  
fi  
echo "Iteration number: $var1"  
done
```

When the conditions of the if-then statement are met (the value is greater than 5 and less than 10), the shell executes the continue command, which skips the rest of the commands in the loop, but keeps the loop going.

A terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'likegeeks@likegeeks-VirtualBox ~/Desktop \$'. The command './myscript' has been entered. The output shows a loop printing 'Iteration number:' followed by values 1, 2, 3, 4, 5, 10, 11, 12, 13, and 14. The prompt is now 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' with a cursor.

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Iteration number: 1
Iteration number: 2
Iteration number: 3
Iteration number: 4
Iteration number: 5
Iteration number: 10
Iteration number: 11
Iteration number: 12
Iteration number: 13
Iteration number: 14
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

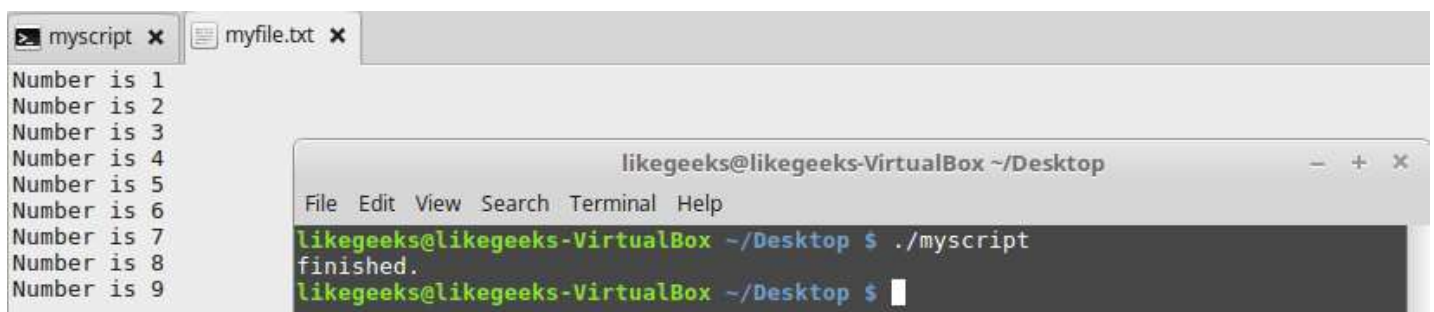
Processing the Output of a Loop

You can either pipe or redirect the output of a loop within your shell script. You do this by adding the processing command to the end of the done command.

So instead of displaying the results on screen, the shell redirects the results of the for command to the file or whatever

```
#!/bin/bash
for (( a = 1; a < 10; a++ ))
do
echo "Number is $a"
done > myfile.txt
echo "finished."
```

The shell creates the file myfile.txt and redirects the output of the for command to the file. And if we check that file we will find our loop output inside it.

Two terminal windows are shown. The top window, titled 'myscript', displays the output of the script: 'Number is 1' through 'Number is 9'. The bottom window, titled 'likegeeks@likegeeks-VirtualBox ~/Desktop', shows the command './myscript' being executed, followed by the output 'finished.' and the prompt 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' with a cursor.

```
myscript
Number is 1
Number is 2
Number is 3
Number is 4
Number is 5
Number is 6
Number is 7
Number is 8
Number is 9

likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
finished.
likegeeks@likegeeks-VirtualBox ~/Desktop $
```


Let's employ out bash scripting knowledge in something useful.

Useful Examples

Finding executables

If you want to find what executable files are available on your system for you to use, just scan all the folders in the PATH environment variable. We discussed for loop and if statements and file separator so our toolset is ready. Let's combine them together and make something pretty

```
#!/bin/bash
IFS=:
for folder in $PATH
do
echo "$folder:"
for file in $folder/*
do
if [ -x $file ]
then
echo " $file"
fi
done
done
```

This is just awesome. We were able to get all the executables on the system that we can run them.

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
/home/likegeeks/bin:
/home/likegeeks/.local/bin:
/usr/local/sbin:
/usr/local/bin:
/usr/local/bin/apt
/usr/local/bin/gnome-help
/usr/local/bin/highlight
/usr/local/bin/mint-sha256sum
/usr/local/bin/pastebin
/usr/local/bin/search
/usr/local/bin/yelp
/usr/sbin:
/usr/sbin/accept
/usr/sbin/accessdb
/usr/sbin/acpid
/usr/sbin/add-apt-key
/usr/sbin/addgnupghome
/usr/sbin/addgroup
/usr/sbin/add-shell
/usr/sbin/adduser
/usr/sbin/alsactl
/usr/sbin/alsa-info.sh
/usr/sbin/anacron
```

Now nothing stops you except your imagination. And this is the beauty of bash scripting

This for now, I hope you enjoy the article and learn a new thing or at least review your knowledge if you forget it. My last word is keep reading and practicing.

Thanks

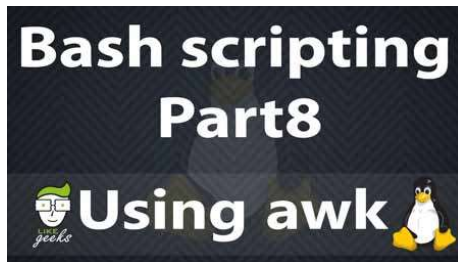
8



Admin

<https://likegeeks.com>

RELATED ARTICLES



LINUX

30 Examples for awk command in text processing

📅 February 21, 2017 👤 admin

On the previous post we've talked about sed Linux command and we've seen many examples of using it in text processing and how it is good in this, nobody can deny that sed is very handy tool but it has some limitations, sometimes you need a more advanced tool for manipulating data, one that provides [...]

21

◀ Bash Script Step By Step, You will love it



LINUX

Expect command and how to automate shell scripts like magic

📅 February 27, 2017 👤 admin

In the previous post we've talked about writing practical shell scripts and we've seen how it is easy to write a shell script and we've used most of our knowledge we've discussed on the previous posts, today we are going to talk about a tool that does magic to our shell scripts, that tool is expect command [...]

7



LINUX

What is Linux File System ? Easy Guide

📅 January 30, 2017 👤 admin

So What Is Linux File System? As we've talked about Linux on previous post and we choose the best linux distro already and also we learn how to install Linux not it is the time to dig down and understand Linux from inside and we will start with what we've mentioned on load post which is [...]

0

Linux bash scripting the awesome guide part3 ▶

4 Comments **likegeeks****1** **Login** ▾**♥ Recommend** **🔗 Share****Sort by Best** ▾

Join the discussion...

**clfapujc** • 21 days ago

In the last example there is:

for folder in \$PATH

How does the script know to look in actual folders. We never used -d to specify we wanted to look in directories.

^ | ▾ • Reply • Share ›

**likegeeks** Mod → **clfapujc** • 21 days ago

When we use the asterisk * it means all files and directories in that folder.
so we get files and directories BOTH of them without filtering .
the filtration to get the folders is on the second if statement which is.

for file in \$folder/*

then we search for the executable with -x

^ | ▾ • Reply • Share ›

**clfapujc** → **likegeeks** • 21 days ago

Now I get it. Thanks.

For a nearly total noob it is confusing :P

1 ^ | ▾ • Reply • Share ›

**likegeeks** Mod → **clfapujc** • 20 days ago

You are always welcome

^ | ▾ • Reply • Share ›

✉ Subscribe **D Add Disqus to your site** **Add Disqus** **Add** **🔒 Privacy**

SEARCH

Search ...

Search