Sunday, March 05, 2017

LIKE GEEKS

LINUX

Shell Scripting The Awesome Guide Part4

February 13, 2017 admin 0 Comments

On the previous post we've talked about **parameters and options** in detail and today we will talk about something is very important in **shell scripting** which is input & output & redirection.

So far, you've seen two methods for displaying the output from your **shell scripts**

- Displaying output on the screen
- Redirecting output to a file

Sometimes you need to display some data on the screen and other data in a file so you need to know how Linux handles input and output so you can get your shell script output to the right place

Our main points are:

Standard file descriptors

Everything is a file in Linux and that includes input and output and Linux identifies each file using the file descriptor.

Each process is allowed to have up to nine open file descriptors at a time. The bash shell reserves the first three file descriptors 0, 1, 2

0	STDIN	Standard input
1	STDOUT	Standard output
2	STDERR	Standard error

These three special file descriptors handle the input and output from your shell script.

You need to fully understand those three because those are like the backbones of your shell scripting, so we are going to describe every one of them in detail.

STDIN

This stands for the standard input to the shell. For a terminal interface, the standard input is the keyboard.

When you use the input redirect symbol (<) in shell scripting, Linux replaces the standard input file descriptor with the file referenced. It reads the file and sends the data just as if it were typed on the keyboard No magic.

Many bash commands accept input from STDIN If no files are specified on the command line like cat command.

When you enter the cat command on the command line without anything, it accepts input from STDIN. As you enter each line, the cat command print the line to the screen

STDOUT

This stands for the standard output for the shell. The standard output is the screen.

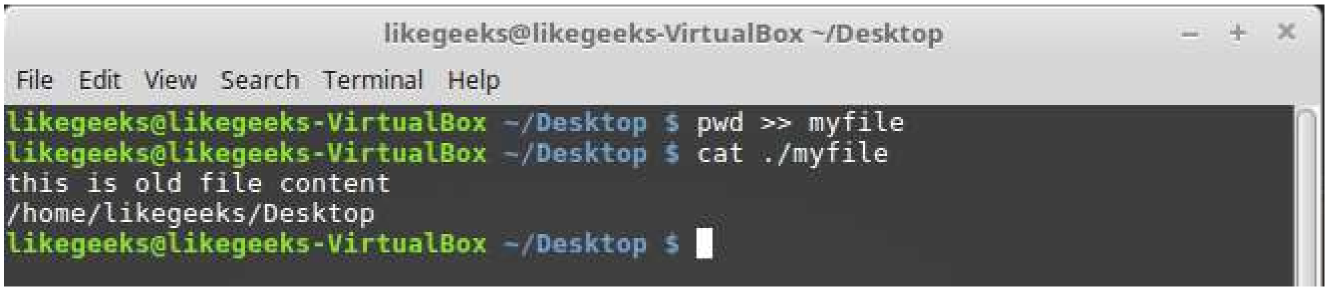
Most bash commands direct their output to the STDOUT file descriptor by default which is the screen.

You can also append data to a file. You do this using the >> symbol.

So if we have a file contains data we can append another data to it using this symbol like this

```
pwd >> myfile
```

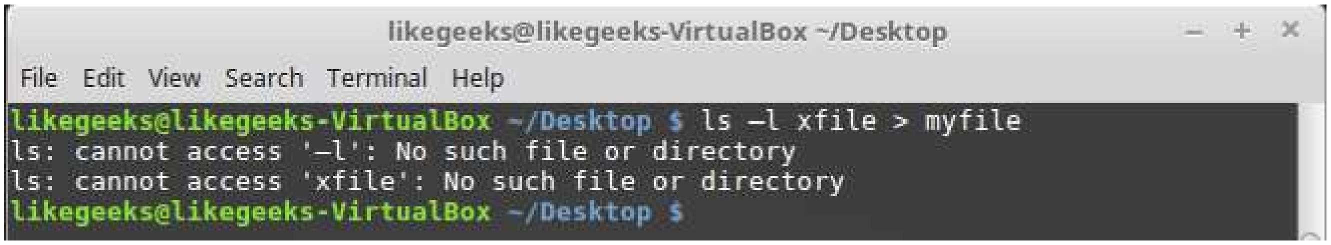
The output generated by pwd is appended to myfile without deleting the existed content.


likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ pwd >> myfile
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./myfile
this is old file content
/home/likegeeks/Desktop
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Fine but if you try to redirect something and that command run into a problem

```
ls –l xfile > myfile
```


likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ls –l xfile > myfile
ls: cannot access '–l': No such file or directory
ls: cannot access 'xfile': No such file or directory
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Here there is no file called xfile on my PC and it generates error and the shell doesn't redirect the error message to the output redirection file but the error message appeared on the screen and here is the third type of file descriptors

STDERR

This file descriptor standard error output for the shell

By default, the STDERR file descriptor points to the same place as the STDOUT file descriptor that's why when an error occurs you see the error on the screen.

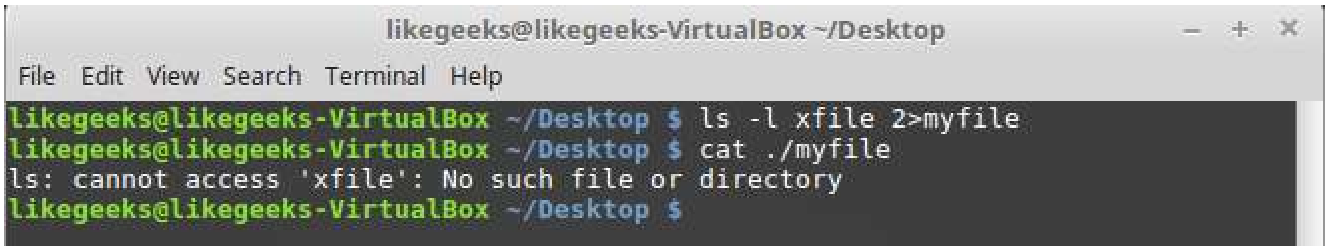
So you need to redirect the errors to maybe log file or any else instead of printing it on the screen

Redirecting errors

As we see the STDERR file descriptor is set to the value 2. We can redirect the errors by placing the file descriptor before the redirection symbol like this

```
ls -l xfile 2>myfile
```

```
cat ./myfile
```


likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help

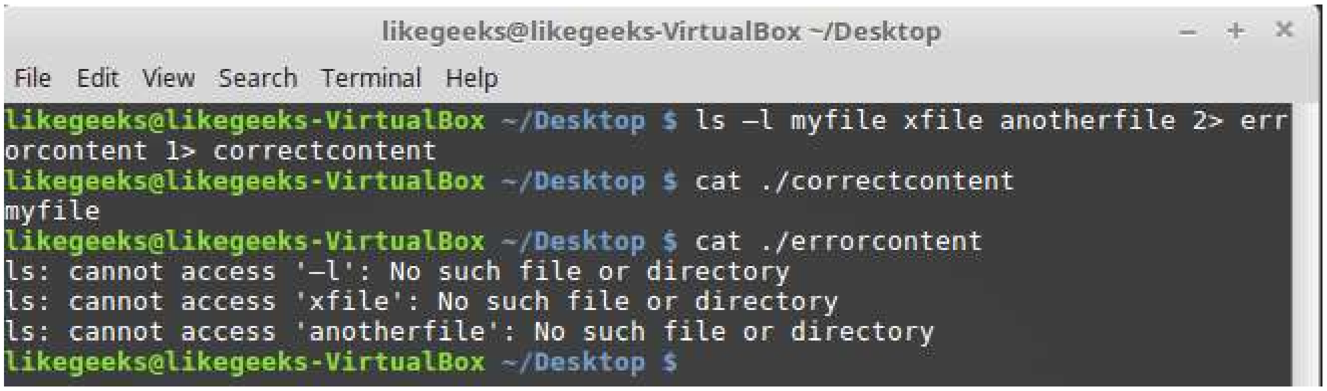
```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ls -l xfile 2>myfile
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./myfile
ls: cannot access 'xfile': No such file or directory
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

As you see the error now is in the file and nothing on the screen

Redirecting errors and normal output

In shell scripting, if you want to redirect both errors and the normal output, you need to precede each with the appropriate file descriptor for the data you want to redirect like this

```
ls –l myfile xfile anotherfile 2> errorcontent 1> correctcontent
```

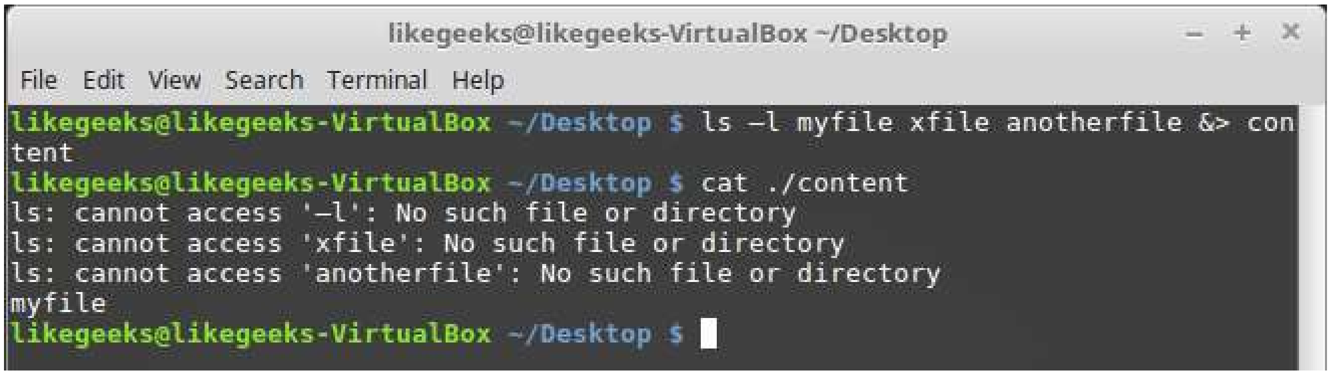

likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ls –l myfile xfile anotherfile 2> err
orcontent 1> correctcontent
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./correctcontent
myfile
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./errorcontent
ls: cannot access '–l': No such file or directory
ls: cannot access 'xfile': No such file or directory
ls: cannot access 'anotherfile': No such file or directory
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

The shell redirects the normal output of the ls command that goes to STDOUT to the correctcontent file using the 1> symbol. And error messages that would have gone to STDERR were redirected to the errorcontent file using the 2> symbol.

If you want, you can redirect both STDERR and STDOUT output to the same output file use &> symbol like this

```
ls –l myfile xfile anotherfile &> content
```


likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ls –l myfile xfile anotherfile &> con
tent
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./content
ls: cannot access '–l': No such file or directory
ls: cannot access 'xfile': No such file or directory
ls: cannot access 'anotherfile': No such file or directory
myfile
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

All error and standard output are redirected to file named content.

Redirecting Output in Scripts

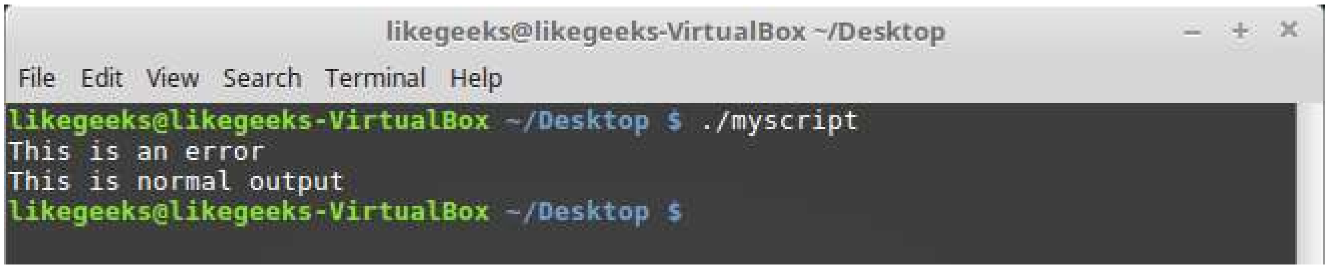
There are two methods for redirecting output in shell scripting

- Temporarily redirection
- Permanently redirection

Temporary redirections

You can redirect an individual output line to STDERR. You just need to use the output redirection symbol to redirect the output to the STDERR file descriptor and you must precede the file descriptor number with an ampersand (&) like this

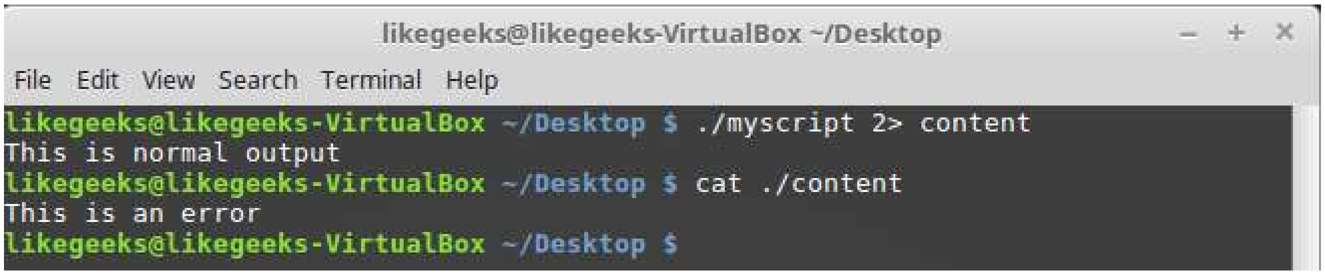
```
#!/bin/bash

echo "This is an error" >&2
echo "This is normal output"
```


likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
This is an error
This is normal output
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

So if we run it we will see both lines printed normally because as we know STDERR output to STDOUT if you redirect STDERR when running the script we should do it like this

```
./myscript 2> myfile
```


likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help

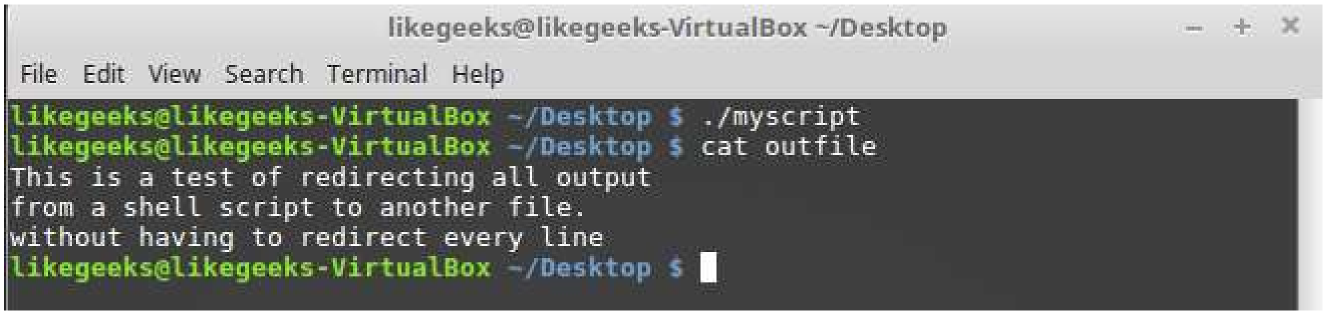
```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript 2> content
This is normal output
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./content
This is an error
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Shell scripting is Awesome! The text displayed using STDOUT appears on the screen, while the echo statement sent to STDERR is redirected to the output file

Permanent redirections

If you have lots of data that you're redirecting in your script, it would be hard to redirect every echo statement. Instead, you can redirect to a specific file descriptor for the duration of the script by using the exec command.

```
#!/bin/bash
exec 1>outfile
echo "This is a test of redirecting all output"
echo "from a shell script to another file."
echo "without having to redirect every line"
```

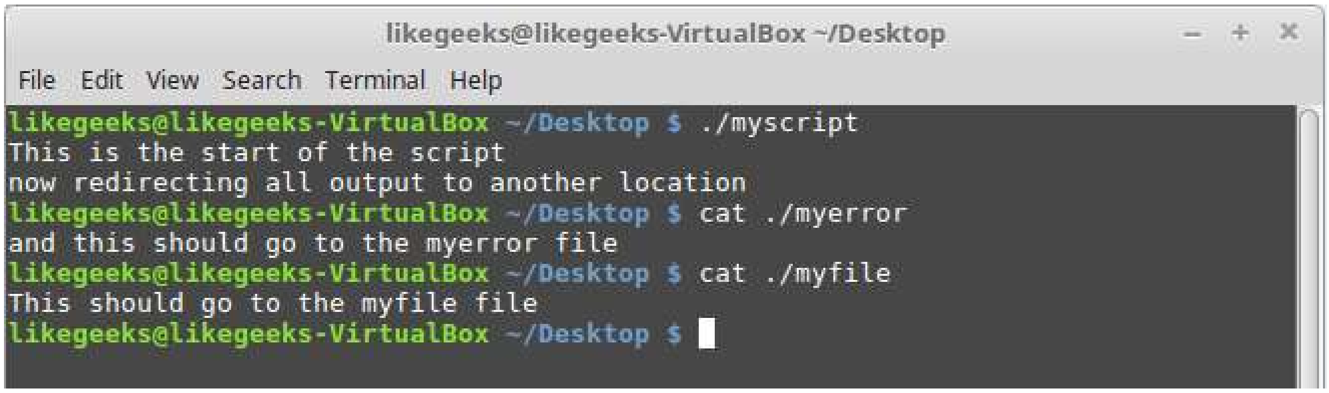


```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat outfile
This is a test of redirecting all output
from a shell script to another file.
without having to redirect every line
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

If we look at the file called outfile we will see the output of echo lines.

You can also redirect the STDOUT in the middle of a script like this

```
#!/bin/bash
exec 2>myerror
echo "This is the start of the script"
echo "now redirecting all output to another location"
exec 1>myfile
echo "This should go to the myfile file"
echo "and this should go to the myerror file" >&2
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
This is the start of the script
now redirecting all output to another location
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./myerror
and this should go to the myerror file
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./myfile
This should go to the myfile file
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

The exec command redirects any output going to STDERR to the file myerror. Then, the script uses the echo statement to display a few lines to STDOUT which is the screen.

After that, the exec command is used again to redirect STDOUT to the myfile file and finally, we redirect the error from within the echo statement to go to STDERR which in this case is myerror file.

Now you have all the ability to redirect all of your output to whatever you want Excellent!

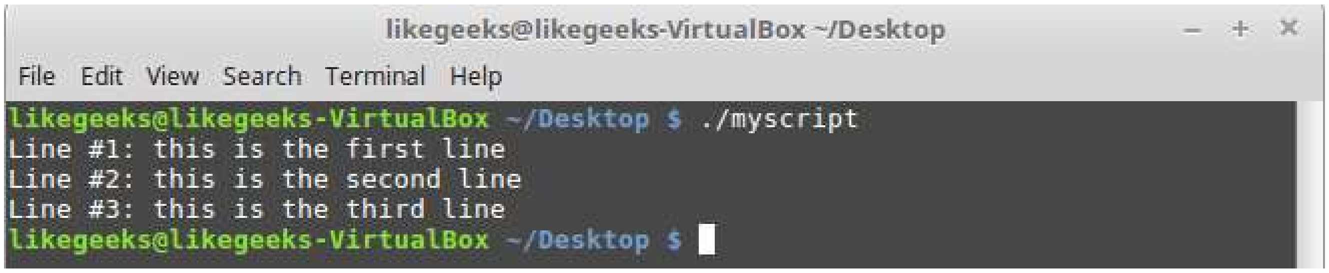
Redirecting Input in Scripts

You can use the same technique you've learned to redirect the output to redirect input. The exec command allows you to redirect STDIN from a file.

```
exec 0< myfile
```

This command tell the shell to take the input from the file called myfile instead of STDIN and here is an example

```
#!/bin/bash
exec 0< testfile
count=1
while read line
do
echo "Line #$count: $line"
count=$(( $count + 1 ))
done
```


likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Line #1: this is the first line
Line #2: this is the second line
Line #3: this is the third line
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Shell scripting is easy.

I showed you on the previous post how to use the read command to read data entered from the keyboard by a user. By redirecting STDIN from a file, when the read command attempts to read from STDIN, it retrieves data from the file instead of the keyboard.

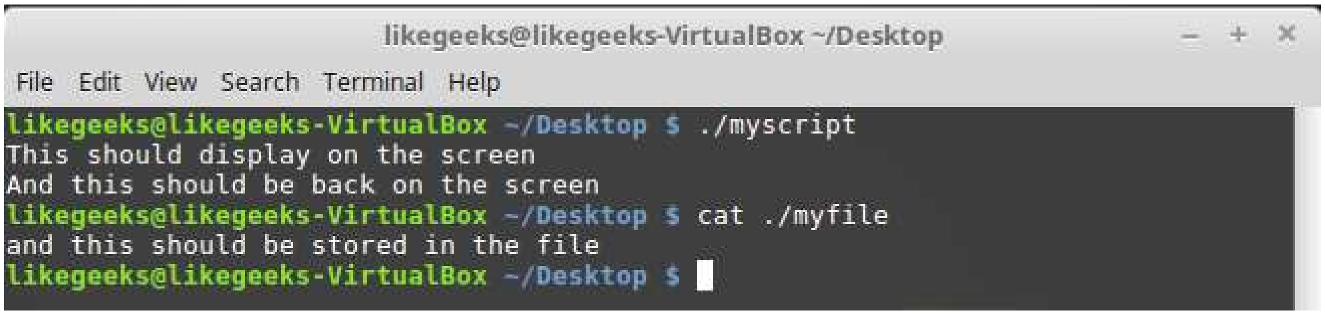
Some Linux system administrators use this technique to read the log files for processing and we will discuss more ways to read the log on the upcoming posts more professionally.

Creating Your Own Redirection

When you redirect input and output in your shell script, you're not limited to the three default file descriptors. As I mentioned that you could have up to nine open file descriptors in the shell. The other six file descriptors from 3 through 8 and are available for you to use as either input or output redirection. You can assign any of these file descriptors to a file and then use them in your shell scripts

You can assign a file descriptor for output by using the exec command and here's an example how to do that

```
#!/bin/bash

exec 3>myfile

echo "This should display on the screen"

echo "and this should be stored in the file" >&3

echo "And this should be back on the screen"
```


likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help

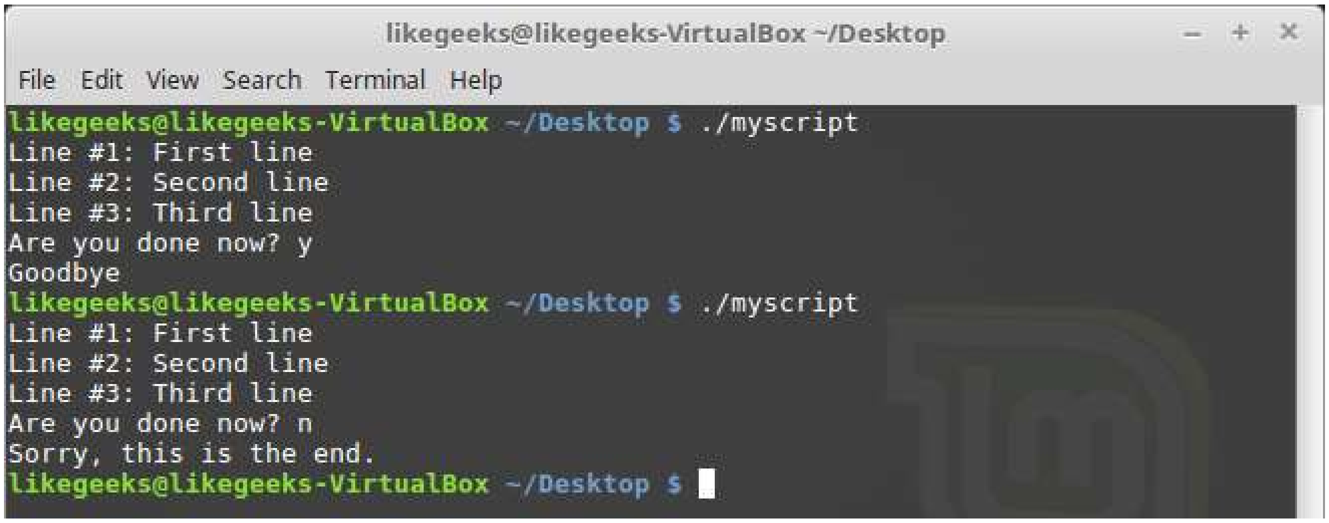
```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
This should display on the screen
And this should be back on the screen
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./myfile
and this should be stored in the file
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Creating input file descriptors

You can redirect input file descriptors in shell scripting exactly the same way as output file descriptors. Save the STDIN file descriptor location to another file descriptor before redirecting it to a file.

When you're finished reading the file, you can restore STDIN to its original location

```
#!/bin/bash

exec 6<&0
exec 0< myfile
count=1
while read line
do
echo "Line #$count: $line"
count=$(( $count + 1 ))
done
exec 0<&6
read -p "Are you done now? " answer
case $answer in
y) echo "Goodbye";;
n) echo "Sorry, this is the end.";;
esac
```


likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Line #1: First line
Line #2: Second line
Line #3: Third line
Are you done now? y
Goodbye
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Line #1: First line
Line #2: Second line
Line #3: Third line
Are you done now? n
Sorry, this is the end.
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

In this example, file descriptor 6 is used to hold the location for STDIN. The shell script then redirects STDIN to a file. All the input for the read command comes from the redirected STDIN, which is now the input file.

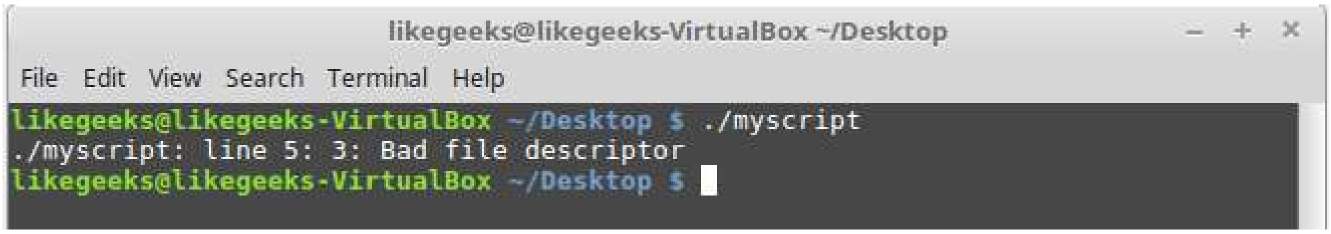
After all the lines have been read, the shell script returns STDIN to its original location by redirecting it to file descriptor 6. And the shell script makes sure that STDIN is back to normal by using another read command and now it is waiting for your keyboard input.

Closing file descriptors

The shell automatically closes the file descriptors when the script exits. There are situations you need to manually close a file descriptor before the end of the script. To close a file descriptor, redirect it to the special symbol &- like this

```
exec 3>&-
```

```
#!/bin/bash
exec 3> myfile
echo "This is a test line of data" >&3
exec 3>&-
echo "This won't work" >&3
```


likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
./myscript: line 5: 3: Bad file descriptor
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

As you can see it gives error bad file descriptor because it is no longer exist

Note: careful in shell scripting when closing file descriptors. If you open the same output file later on in your shell script, the shell replaces the existing file with a new file. This means that if you output any data, it overwrites the existing file

Listing open file descriptors

The lsof command lists all the open file descriptors on the entire Linux system

On many Linux systems like Fedora, the lsof command is located in the /usr/sbin.

This command is very useful actually it displays information about every file currently open on the Linux system. This includes all the processes running on background, as well as any user accounts logged into the system.

This command has a lot of options so I think I will make a special post about it later but let's take the important parameters we need

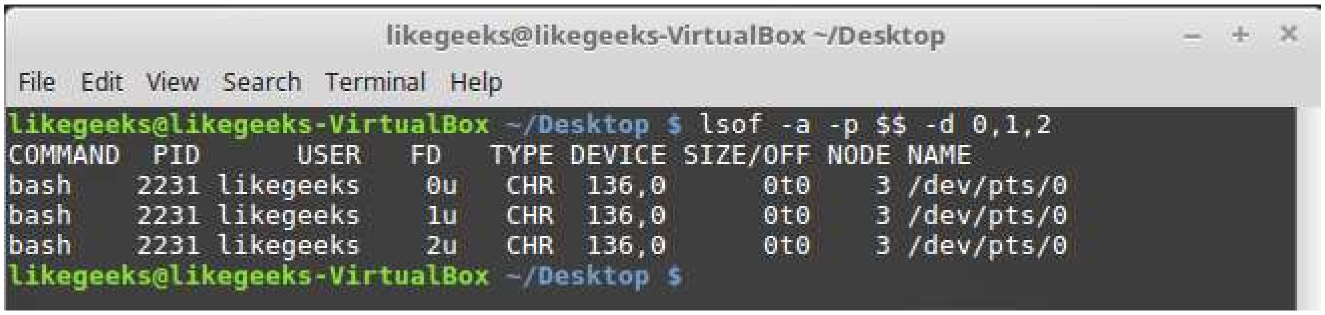
-p, allows you to specify a process ID

-d, allows you to specify the file descriptor numbers to display

To get the current PID of the process, you can use the special environment variable $$, which the shell sets to the current PID.

The -a option is used to perform a Boolean AND of the results of the other two options ONLY, to produce the following

```
lsof -a -p $$ -d 0,1,2
```

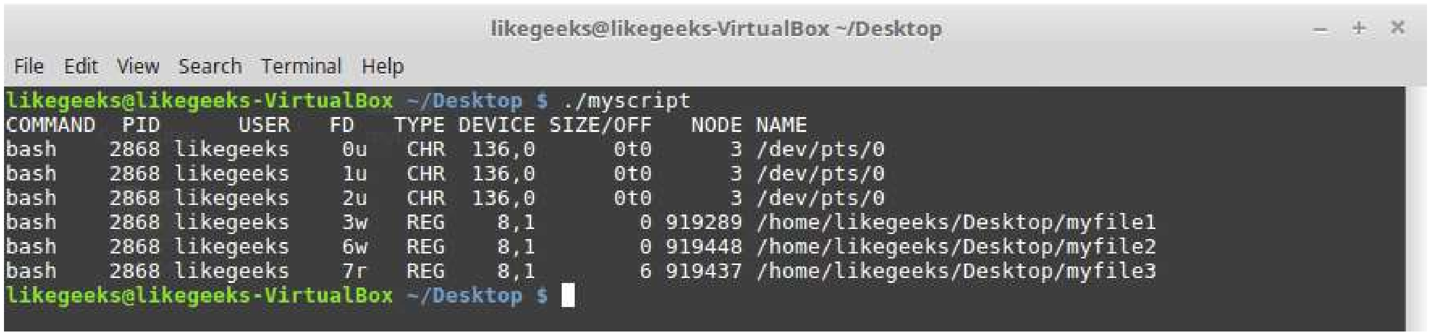

likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ lsof -a -p $$ -d 0,1,2
COMMAND  PID      USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
bash    2231 likegeeks   0u   CHR  136,0      0t0    3 /dev/pts/0
bash    2231 likegeeks   1u   CHR  136,0      0t0    3 /dev/pts/0
bash    2231 likegeeks   2u   CHR  136,0      0t0    3 /dev/pts/0
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

The file type associated with STDIN, STDOUT, and STDERR is character mode. Because the STDIN, STDOUT, and STDERR file descriptors all point to the terminal, the name of the output file is the device name of the terminal. All three standard files are available for both reading and writing.

Now, let's look at the results of the lsof command from inside a script that's opened a couple of alternative file descriptors

```
#!/bin/bash
exec 3> myfile1
exec 6> myfile2
exec 7< myfile3

lsof -a -p $$ -d 0,1,2,3,6,7
```


likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
COMMAND  PID      USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
bash    2868 likegeeks   0u   CHR  136,0      0t0      3 /dev/pts/0
bash    2868 likegeeks   1u   CHR  136,0      0t0      3 /dev/pts/0
bash    2868 likegeeks   2u   CHR  136,0      0t0      3 /dev/pts/0
bash    2868 likegeeks   3w   REG    8,1        0 919289 /home/likegeeks/Desktop/myfile1
bash    2868 likegeeks   6w   REG    8,1        0 919448 /home/likegeeks/Desktop/myfile2
bash    2868 likegeeks   7r   REG    8,1        6 919437 /home/likegeeks/Desktop/myfile3
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

The shell script creates three file descriptors, two for output (3 and 6) and one for input (7).

And you can see the pathname for the files used in the file descriptors.

Suppressing Command Output

Sometimes you don't want to see any output this often occurs if you're running a script as a background process (we will discuss how to make you shell script run in the background in the next posts)

We redirect the output to the hole which is /dev/null

For example, we can suppress errors like this

```
ls -al badfile anotherfile 2> /dev/null
```

And this idea is also used when you want to truncate a file without deleting it completely

```
cat /dev/null > myfile
```

Now you understand the input and output and how to redirect them and how to create your own file descriptor and redirect to it. This is very important in shell scripting.

I hope you enjoy it; the next post will be how to control our running script and how to run your shell script in the background without interruption and how to pause them while they are running and some other cool stuff, Stay tuned.

Thanks

9

Admin

https://likegeeks.com

RELATED ARTICLES

LINUX

How to write practical shell script

February 25, 2017 admin

In the last post, we've talked about regex and we see how to use them in sed and awk for text processing and we discussed before Linux sed command and awk command. During the series, we write small shell scripts but we didn't mix things up, I think we should take a small step and write […]

LINUX

Best Linux Distro For 2017 That Fits Your Needs

January 29, 2017 admin

What Is The Best Linux Distro For 2017? So what is the best Linux distro? If you know Linux you may know that there are a lot and a lot of Linux distros out there and you can check most of them from distro watch website https://distrowatch.com/ You may try few of them so […]

LINUX

Bash scripting the awesome guide Part2

February 9, 2017 admin

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 [...]

41

2

8

‹ Linux bash scripting the awesome guide part3

Linux bash scripting the awesome guide part5 ›

0 Comments **likegeeks** 1 **Login**

Recommend **Share** **Sort by Best**

Start the discussion…

Be the first to comment.

Subscribe **Add Disqus to your site Add Disqus Add** **Privacy**

SEARCH

Search …

Search