Assignment: Lab 4 - Who Can Do What? (rwxrwxrwt)

New Attempt

Due Sunday by 11pm **Points** 50 **Submitting** a text entry box or a file upload

Purpose: This lab will allow the student to get a basic understanding of file content manipulation using the sed and grep commands, using file permissions, and cron jobs in the Unix file system as used by a system administrator. As a system administrator you will need to be resourceful, so feel free to Google the command, use any reference text, and/or collaborate with classmates on executing the commands and seeing what results are expected and also what options can be used with each command.

Instructions: Follow the steps below to perform each task on your VM. Be sure you understand what each command does and when to use them as they will be used in future lab assignments. Answer any follow-up questions related to the lab and then turn in the document as an attachment to the lab assignment.

THE GREP AND SED COMMANDS

1) Login to your VM and use an editor to create a file called grep_example1.txt and enter the following lines of data:

```
Unix is a great operating system.
Unix is multi-tasking.
Unix is reliable.
Unix is widely used.
Unix is fun to learn.
Unix is easy to learn.
Unix is good to learn.
```

Save and exit the file.

Make a copy of the file to a second file called grep_example2.txt
Make a copy of the file to a third file called grep_example3.txt

2) Login to your VM and use an editor to create a file called sed_example1.txt and enter the following lines of data:

```
Unix is a great operating system.
Unix is multi-tasking.
Unix is reliable.
```

Unix is widely used.
Unix is fun to learn.
Unix is easy to learn.
Unix is good to learn.

Save and exit the file.

Make a copy of the file to a second file called sed_example2.txt
Make a copy of the file to a third file called sed_example3.txt

3) Using the grep Command (Provide a screen shot showing each command below):

Use the grep command to search for the string ‘Unix’ in the file grep_example1.txt

Did all lines in the file display? Why? (Record the answer for question 1 in the lab submission)

Use the grep command to search for the string ‘reliable’ in the file grep_example2.txt

Did all lines in the file display? Why? (Record the answer for question 2 in the lab submission)

Use the grep command to search for the string ‘learn’ in the file grep_example3.txt

Did all lines in the file display? Why? (Record the answer for question 3 in the lab submission)

4) Using the sed Command (Provide a screen shot showing each command below):

Use the sed command to search for the string ‘Unix’ and replace it with the string ‘Linux’ in the file sed_example1.txt

Did all lines in the file change? Why? (Record the answer for question 4 in the lab submission)

Use the sed command to change only the third occurrence of the string ‘Unix’ with the string ‘Linux’ in the sed_example2.txt

Did all lines in the file change? Why? (Record the answer for question 5 in the lab submission)

Use the sed command to search for the string ‘Unix is’ and replace it with the string ‘Windows is not’ in the file sed_example3.txt

Did all lines in the file change? Why? (Record the answer for question 6 in the lab submission)

<u>SETTING FILE PERMISSIONS</u>

1) Create a file called fileperm1.txt that only has read write access for you and nobody else.

Use ls -l to show this and provide a screen shot for question 7 in the lab submission.

2) Create another file called fileperm2.txt that has read, write, and execute permissions for you and only read and execute permissions for everyone else.

Use ls -l to show this and provide a screen shot for question 8 in the lab submission.

3) Create another file called fileperm3.txt that is wide-open for everyone. Is this good practice? Explain why or why not?

Use ls -l to show this and provide a screen shot for question 9 in the lab submission.

4) Create another file called fileperm4.txt that is has the sticky-bit set to non-execute.

Use ls -l to show this and provide a screen shot for question 10 in the lab submission.

5) Create another file called fileperm5.txt that is has the sticky-bit set to execute.

Use ls -l to show this and provide a screen shot for question 11 in the lab submission.

CRONTAB AND CRON JOBS

1) Write a script file called testscript.sh and enter the following lines:

```
#!/bin/bash
# Declare array with 5 elements
ARRAY=( 'Debian Linux' 'Redhat Linux' 'Ubuntu Linux' 'SUSE Linux' 'CentOS Linux' )
```
<---- (Note: There are spaces between the array values)

```
# get number of elements in the array
ELEMENTS=${#ARRAY[@]}

# echo each element in array in for loop
for (( i=0;i<$ELEMENTS;i++ )); do
   echo ${ARRAY[${i}]}
done
```

Save and exit the file (Also, don't forget to make the file executable).

Run the script and see if it executes as expected. To run the script execute it as follows:

```
./testscript.sh
```

Did the script execute successfully? Why or why not? (Record the answer for question 12 in the lab submission)

2) Modify the script to redirect the output of the echo command into a file called cron_output.txt in the current user directory:

```
#!/bin/bash
# Declare array with 5 elements
ARRAY=( 'Debian Linux' 'Redhat Linux' 'Ubuntu Linux' 'SUSE Linux' 'CentOS Linux' )
```

```
# get number of elements in the array
ELEMENTS=${#ARRAY[@]}

# echo each element in array in for loop
for (( i=0;i<$ELEMENTS;i++ )); do
   echo ${ARRAY[${i}]} >> cron_output.txt      <---- (Note: This the modification needed; Do
not include this message in the script!)
done
```

Save and exit the file (Verify the file is executable).

Run the script and see if it executes as expected.

Did the script execute successfully? Why or why not? (Record the answer for question 13 in the lab submission)

3) Create a cron job to run the script every two minutes (Research how to do this on the internet). When you use crontab for the first time, it will ask you to select an editor...I recommend sticking with nano (option 1).

Wait five to ten minutes, then use the more command to check the cron_output.txt file.

Does the file contain any data? (Record the answer for question 14 in the lab submission)

Wait another five to ten minutes, then use the more command to check the cron_output.txt file.

Does the file continue to grow with data? Why? (Record the answer for question 15 in the lab submission)

Make sure you remove the crontab entry after completion of the lab to prevent continuous running.

Did you remove the crontab entry? How did you remove it? (Record the answer or question 16 in the lab submission)

USING A RPM

1) To understand what packages are installed on the VM, use the command:

```
sudo apt --installed list
```

A long list of all the packages currently installed should show on the screen.

A common utility package used by system administrators is sysstat. The sysstat package provides a number of utilities for collecting the system use activities and system performance.

- iostat – Used for CPU statistics and input/output statistics for the block devices and partitions and generate report.

- mpstat – Used for processor related statistics and reports.pidstat – Used for I/O, CPU, memory statistics for Linux processes and generate report.
- tapestat – Used for the statistics for tape drives attached to Linux system.
- cifsiostat – Used for generating reports CIFS statistics.
- sar – Used for collects and saves all the system activities and report.

2) Run the command:

```
sudo apt --installed list | grep sysstat
```

Notice that nothing is returned indicating that the finger package is not installed.

Use the advanced package tool (apt) to install sysstat:

```
sudo apt install sysstat
```

Run the command:

```
iostat
```

What did the iostat command show when running it? (Record the answer for question 17 in the lab submission)

USING SYSTEM PERFORMANCE AND TROUBLESHOOTING COMMANDS

1) Run these very common system performance commands and observe the output from each.

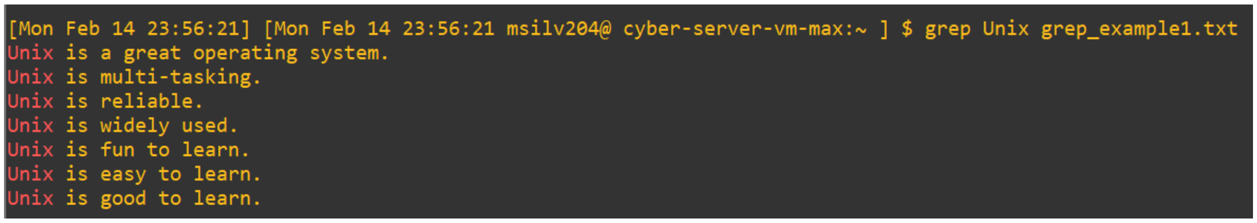
```
top
ps -ef
uptime
netstat -a
ifconfig
```

As a system administrator, which command do you feel is the most useful in debugging server issues and why? (Record the answer for question 18 in the lab submission)

LAB FOLLOW-UP QUESTIONS

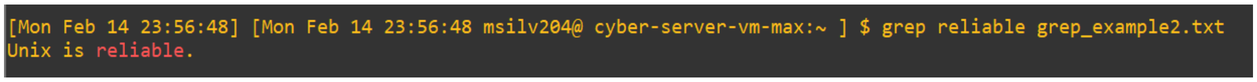
Answer the questions as they were encountered during the lab and record your answers or screenshots in a document and attach it to the lab submission.

You should have eighteen (18) answers or screenshots.

1. 

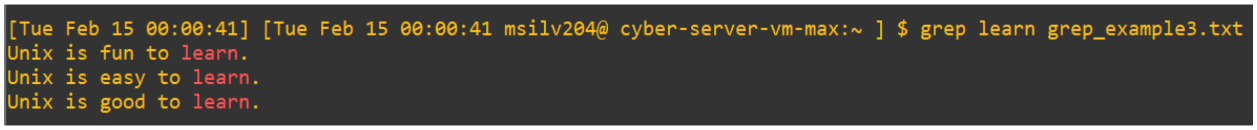
```
[Mon Feb 14 23:56:21] [Mon Feb 14 23:56:21 msilv204@ cyber-server-vm-max:~ ] $ grep Unix grep_example1.txt
Unix is a great operating system.
Unix is multi-tasking.
Unix is reliable.
Unix is widely used.
Unix is fun to learn.
Unix is easy to learn.
Unix is good to learn.
```

All lines in the file display because the *Unix* string occurs in every line.

2. 

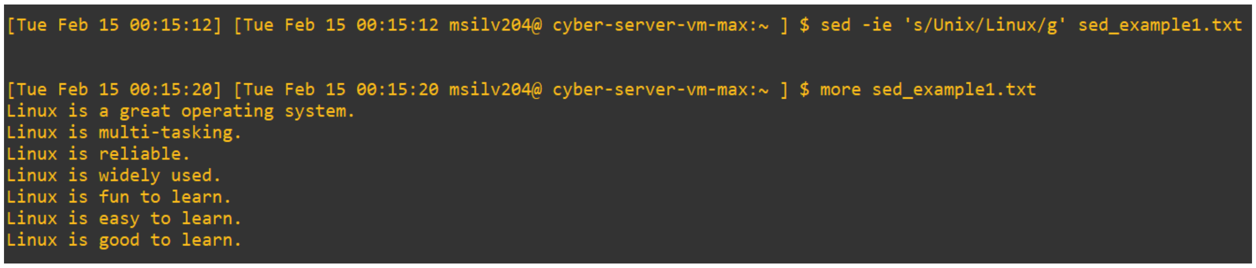
```
[Mon Feb 14 23:56:48] [Mon Feb 14 23:56:48 msilv204@ cyber-server-vm-max:~ ] $ grep reliable grep_example2.txt
Unix is reliable.
```

Only 1 line in the file displays because only 1 line contains the *reliable* string.

3. 

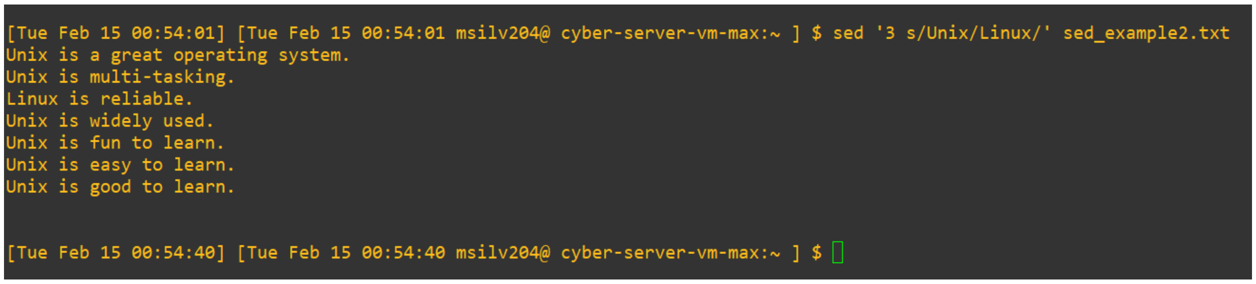
```
[Tue Feb 15 00:00:41] [Tue Feb 15 00:00:41 msilv204@ cyber-server-vm-max:~ ] $ grep learn grep_example3.txt
Unix is fun to learn.
Unix is easy to learn.
Unix is good to learn.
```

Only 3 out of the 7 total lines in the file display because the *learn* string occurs in only the last 3 lines of the file.

4. 

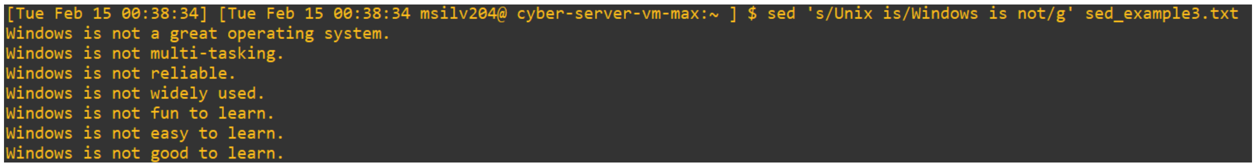
```
[Tue Feb 15 00:15:12] [Tue Feb 15 00:15:12 msilv204@ cyber-server-vm-max:~ ] $ sed -ie 's/Unix/Linux/g' sed_example1.txt


[Tue Feb 15 00:15:20] [Tue Feb 15 00:15:20 msilv204@ cyber-server-vm-max:~ ] $ more sed_example1.txt
Linux is a great operating system.
Linux is multi-tasking.
Linux is reliable.
Linux is widely used.
Linux is fun to learn.
Linux is easy to learn.
Linux is good to learn.
```

All lines in the file change because all lines contained the string that the *sed* command was targeting.

5. 

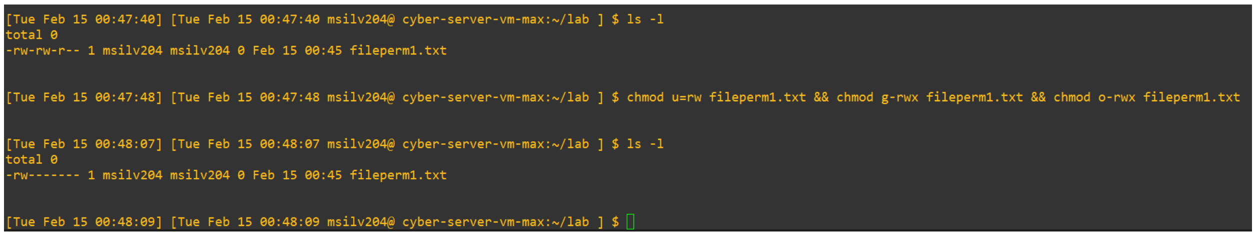
```
[Tue Feb 15 00:54:01] [Tue Feb 15 00:54:01 msilv204@ cyber-server-vm-max:~ ] $ sed '3 s/Unix/Linux/' sed_example2.txt
Unix is a great operating system.
Unix is multi-tasking.
Linux is reliable.
Unix is widely used.
Unix is fun to learn.
Unix is easy to learn.
Unix is good to learn.


[Tue Feb 15 00:54:40] [Tue Feb 15 00:54:40 msilv204@ cyber-server-vm-max:~ ] $
```

Only the third line in the file changes because the third line contains the third occurrence of the *Unix* string.

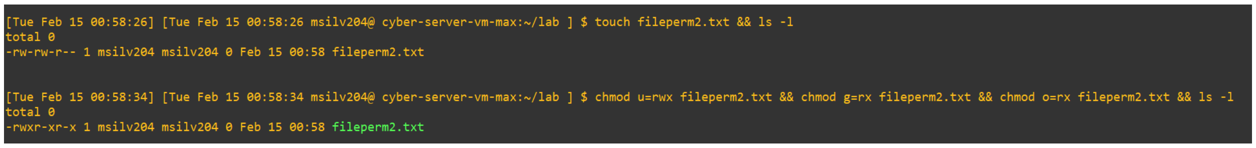
6. 

```
[Tue Feb 15 00:38:34] [Tue Feb 15 00:38:34 msilv204@ cyber-server-vm-max:~ ] $ sed 's/Unix is/Windows is not/g' sed_example3.txt
Windows is not a great operating system.
Windows is not multi-tasking.
Windows is not reliable.
Windows is not widely used.
Windows is not fun to learn.
Windows is not easy to learn.
Windows is not good to learn.
```

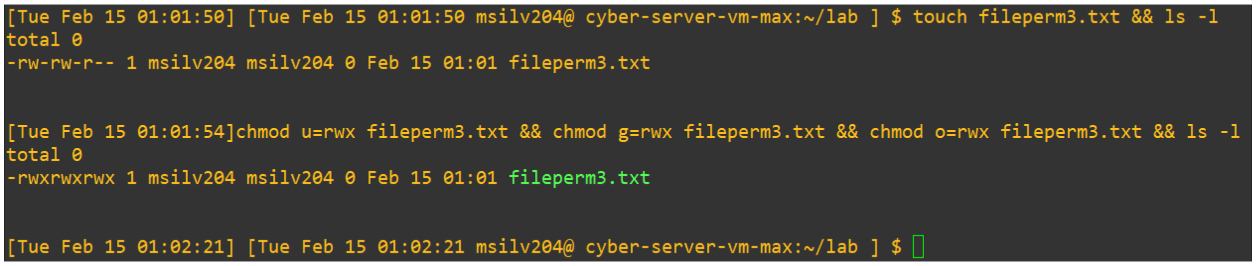
All lines in the file change because all lines contained the string that the *sed* command was targeting.

7. 

```
[Tue Feb 15 00:47:40] [Tue Feb 15 00:47:40 msilv204@ cyber-server-vm-max:~/lab ] $ ls -l
total 0
-rw-rw-r-- 1 msilv204 msilv204 0 Feb 15 00:45 fileperm1.txt

[Tue Feb 15 00:47:48] [Tue Feb 15 00:47:48 msilv204@ cyber-server-vm-max:~/lab ] $ chmod u=rw fileperm1.txt && chmod g-rwx fileperm1.txt && chmod o-rwx fileperm1.txt

[Tue Feb 15 00:48:07] [Tue Feb 15 00:48:07 msilv204@ cyber-server-vm-max:~/lab ] $ ls -l
total 0
-rw------- 1 msilv204 msilv204 0 Feb 15 00:45 fileperm1.txt

[Tue Feb 15 00:48:09] [Tue Feb 15 00:48:09 msilv204@ cyber-server-vm-max:~/lab ] $
```

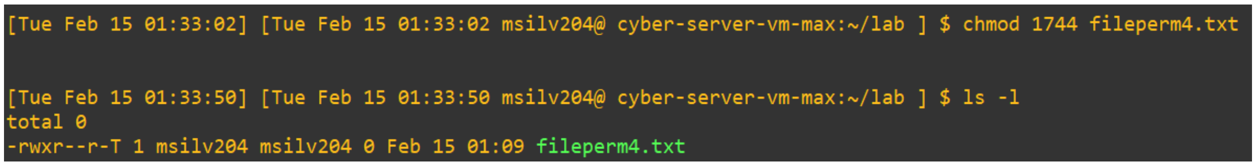
8. 

```
[Tue Feb 15 00:58:26] [Tue Feb 15 00:58:26 msilv204@ cyber-server-vm-max:~/lab ] $ touch fileperm2.txt && ls -l
total 0
-rw-rw-r-- 1 msilv204 msilv204 0 Feb 15 00:58 fileperm2.txt

[Tue Feb 15 00:58:34] [Tue Feb 15 00:58:34 msilv204@ cyber-server-vm-max:~/lab ] $ chmod u=rwx fileperm2.txt && chmod g=rx fileperm2.txt && chmod o=rx fileperm2.txt && ls -l
total 0
-rwxr-xr-x 1 msilv204 msilv204 0 Feb 15 00:58 fileperm2.txt
```

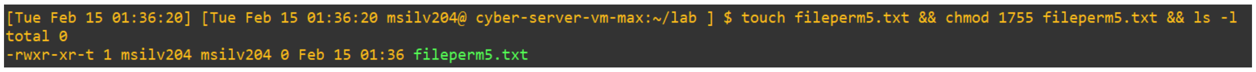
9. 

```
[Tue Feb 15 01:01:50] [Tue Feb 15 01:01:50 msilv204@ cyber-server-vm-max:~/lab ] $ touch fileperm3.txt && ls -l
total 0
-rw-rw-r-- 1 msilv204 msilv204 0 Feb 15 01:01 fileperm3.txt

[Tue Feb 15 01:01:54]chmod u=rwx fileperm3.txt && chmod g=rwx fileperm3.txt && chmod o=rwx fileperm3.txt && ls -l
total 0
-rwxrwxrwx 1 msilv204 msilv204 0 Feb 15 01:01 fileperm3.txt

[Tue Feb 15 01:02:21] [Tue Feb 15 01:02:21 msilv204@ cyber-server-vm-max:~/lab ] $
```

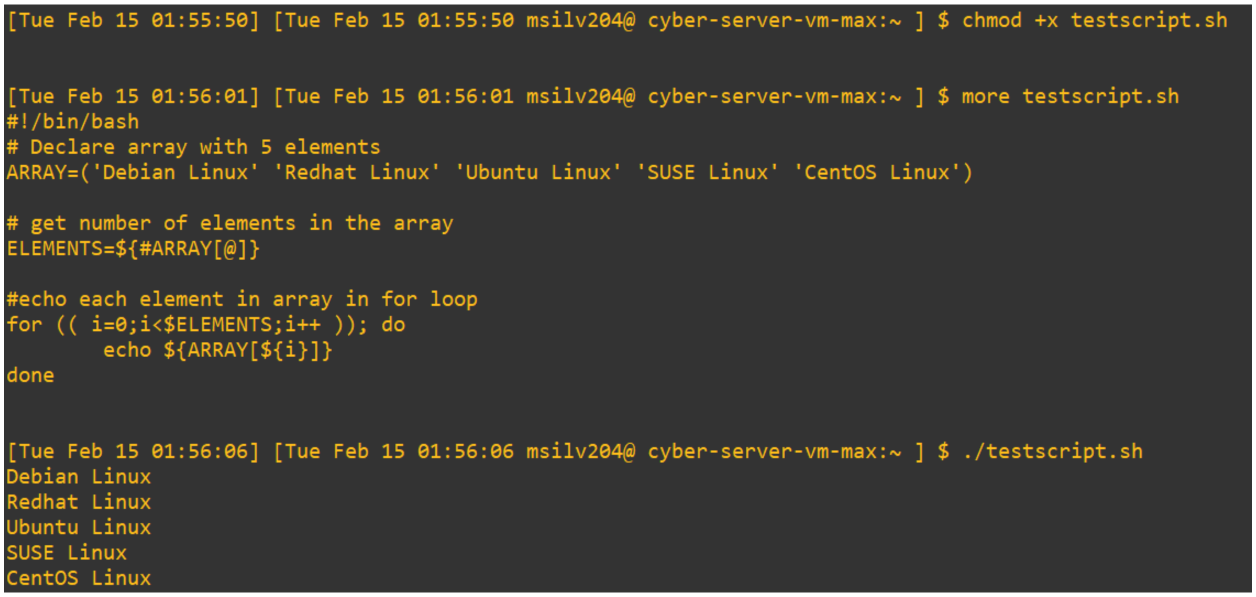
Having a file like *fileperm3.txt* that is wide-open for everyone is usually bad practice since the information in that file is now accessible and editable for all users – this is poor access control, dangerously blurring the lines of acceptable use. This is a breach of data integrity and confidentiality (if sensitive data was entered into this file).

10. 

```
[Tue Feb 15 01:33:02] [Tue Feb 15 01:33:02 msilv204@ cyber-server-vm-max:~/lab ] $ chmod 1744 fileperm4.txt

[Tue Feb 15 01:33:50] [Tue Feb 15 01:33:50 msilv204@ cyber-server-vm-max:~/lab ] $ ls -l
total 0
-rwxr--r-T 1 msilv204 msilv204 0 Feb 15 01:09 fileperm4.txt
```

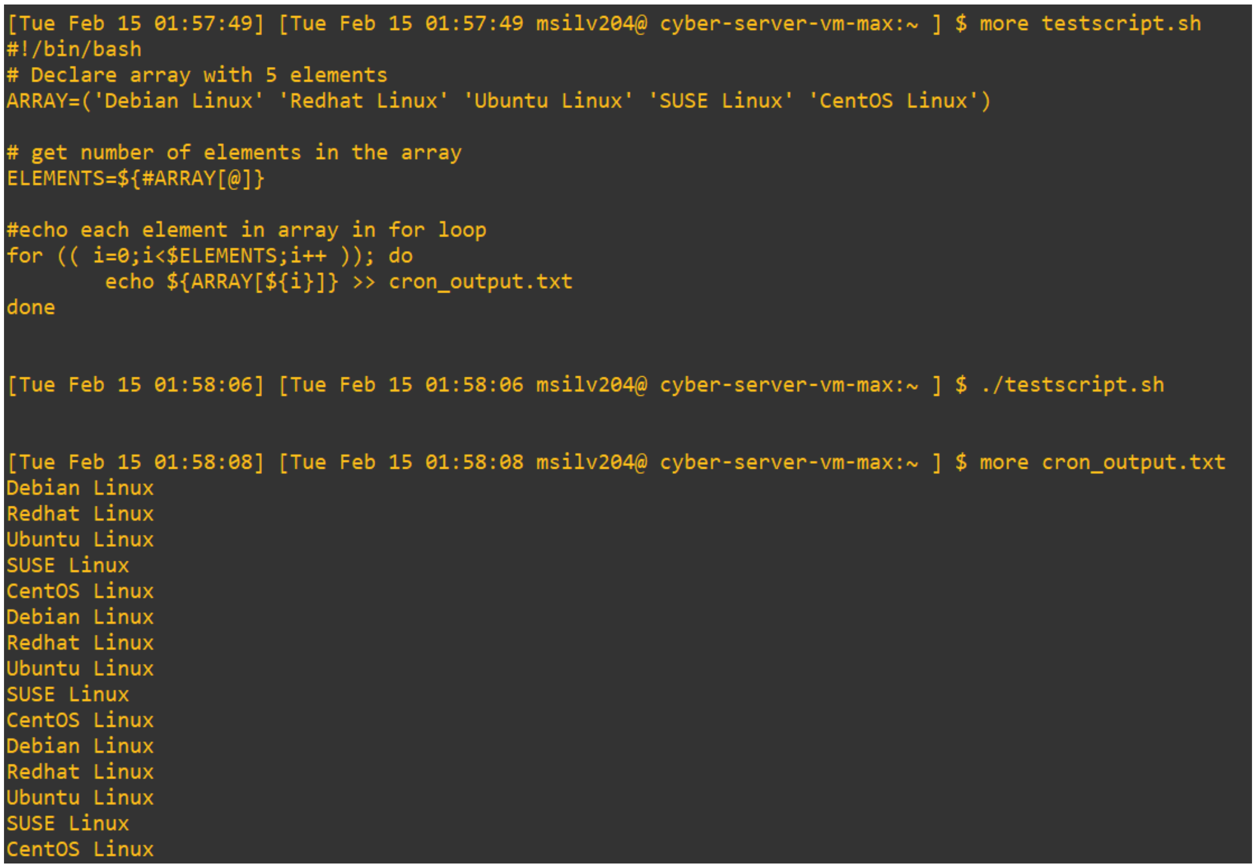
11. 

```
[Tue Feb 15 01:36:20] [Tue Feb 15 01:36:20 msilv204@ cyber-server-vm-max:~/lab ] $ touch fileperm5.txt && chmod 1755 fileperm5.txt && ls -l
total 0
-rwxr-xr-t 1 msilv204 msilv204 0 Feb 15 01:36 fileperm5.txt
```

12. 

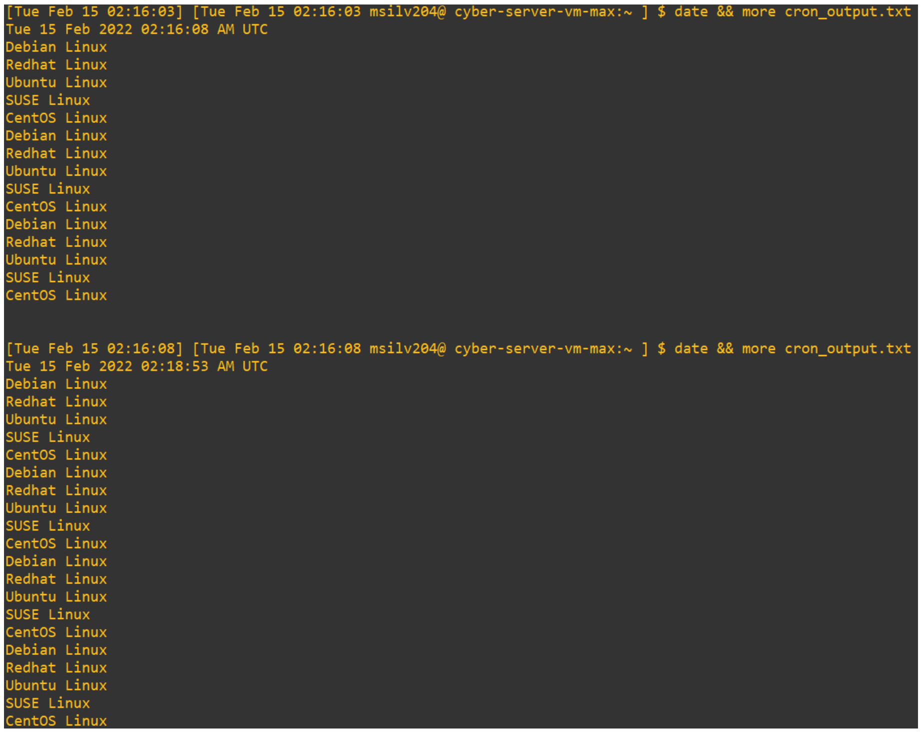
```
[Tue Feb 15 01:55:50] [Tue Feb 15 01:55:50 msilv204@ cyber-server-vm-max:~ ] $ chmod +x testscript.sh


[Tue Feb 15 01:56:01] [Tue Feb 15 01:56:01 msilv204@ cyber-server-vm-max:~ ] $ more testscript.sh
#!/bin/bash
# Declare array with 5 elements
ARRAY=('Debian Linux' 'Redhat Linux' 'Ubuntu Linux' 'SUSE Linux' 'CentOS Linux')

# get number of elements in the array
ELEMENTS=${#ARRAY[@]}

#echo each element in array in for loop
for (( i=0;i<$ELEMENTS;i++ )); do
        echo ${ARRAY[${i}]}
done


[Tue Feb 15 01:56:06] [Tue Feb 15 01:56:06 msilv204@ cyber-server-vm-max:~ ] $ ./testscript.sh
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
```

The script executes correctly after giving the script execute permissions.

13. 

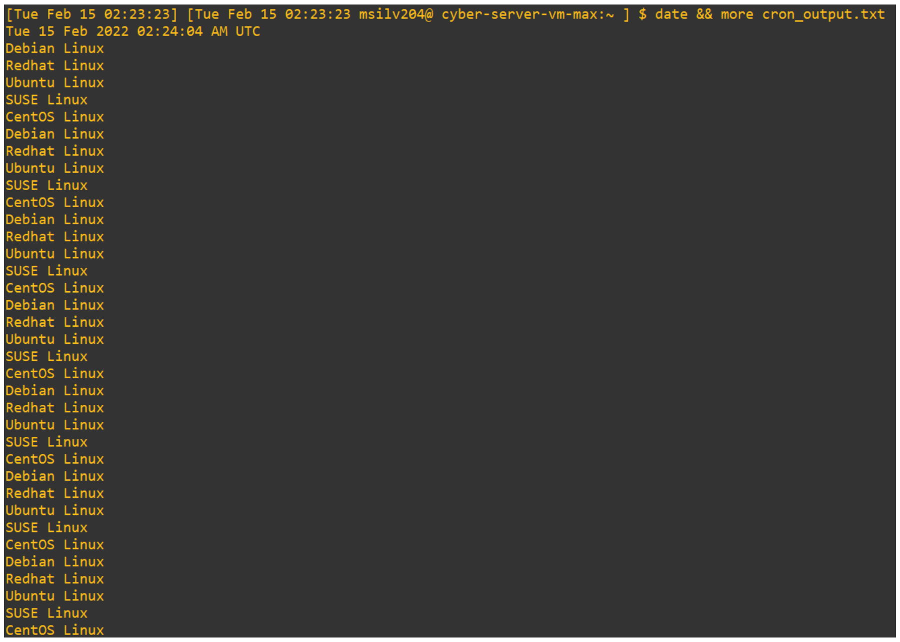
```
[Tue Feb 15 01:57:49] [Tue Feb 15 01:57:49 msilv204@ cyber-server-vm-max:~ ] $ more testscript.sh
#!/bin/bash
# Declare array with 5 elements
ARRAY=('Debian Linux' 'Redhat Linux' 'Ubuntu Linux' 'SUSE Linux' 'CentOS Linux')

# get number of elements in the array
ELEMENTS=${#ARRAY[@]}

#echo each element in array in for loop
for (( i=0;i<$ELEMENTS;i++ )); do
        echo ${ARRAY[${i}]} >> cron_output.txt
done


[Tue Feb 15 01:58:06] [Tue Feb 15 01:58:06 msilv204@ cyber-server-vm-max:~ ] $ ./testscript.sh


[Tue Feb 15 01:58:08] [Tue Feb 15 01:58:08 msilv204@ cyber-server-vm-max:~ ] $ more cron_output.txt
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
```

The script executes successfully and the data ends up in the file following the continuation arrows.

14. 

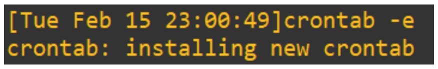
```
[Tue Feb 15 02:16:03] [Tue Feb 15 02:16:03 msilv204@ cyber-server-vm-max:~ ] $ date && more cron_output.txt
Tue 15 Feb 2022 02:16:08 AM UTC
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux

[Tue Feb 15 02:16:08] [Tue Feb 15 02:16:08 msilv204@ cyber-server-vm-max:~ ] $ date && more cron_output.txt
Tue 15 Feb 2022 02:18:53 AM UTC
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
```

The *cron_output.txt* file now contains more data – the *cron* job automatically executes the script every 2 minutes, so the script outputs another array into the text file.

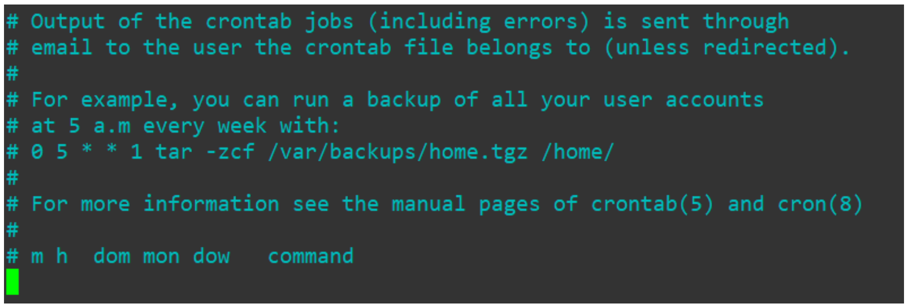
15. 

```
[Tue Feb 15 02:23:23] [Tue Feb 15 02:23:23 msilv204@ cyber-server-vm-max:~ ] $ date && more cron_output.txt
Tue 15 Feb 2022 02:24:04 AM UTC
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
Debian Linux
Redhat Linux
Ubuntu Linux
SUSE Linux
CentOS Linux
```

As time passes, the file continues to grow with data, since the *cron* job continuously executes the script on the specified interval, and the script always feeds data into the file.

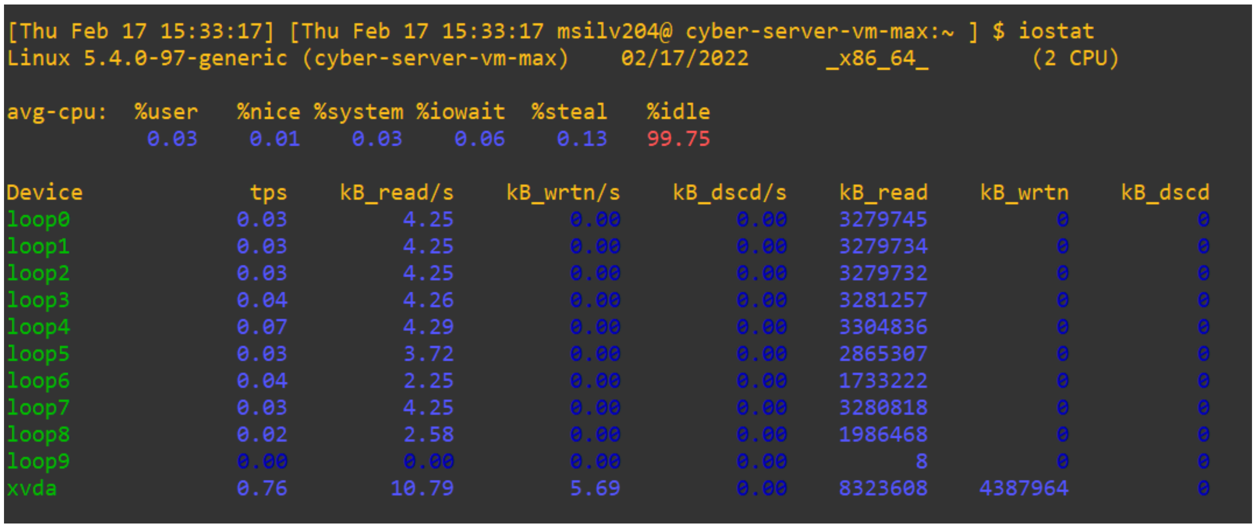


```
[Tue Feb 15 23:00:49]crontab -e
crontab: installing new crontab
```

16. 

```
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
```

The *crontab* entry can be removed with the *crontab -e* command, which opens up a text editor that allows the user to remove any jobs they may have added earlier.

17. 

```
[Thu Feb 17 15:33:17] [Thu Feb 17 15:33:17 msilv204@ cyber-server-vm-max:~ ] $ iostat
Linux 5.4.0-97-generic (cyber-server-vm-max)    02/17/2022      _x86_64_        (2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.03    0.01    0.03    0.06    0.13   99.75

Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
loop0             0.03         4.25         0.00         0.00    3279745          0          0
loop1             0.03         4.25         0.00         0.00    3279734          0          0
loop2             0.03         4.25         0.00         0.00    3279732          0          0
loop3             0.04         4.26         0.00         0.00    3281257          0          0
loop4             0.07         4.29         0.00         0.00    3304836          0          0
loop5             0.03         3.72         0.00         0.00    2865307          0          0
loop6             0.04         2.25         0.00         0.00    1733222          0          0
loop7             0.03         4.25         0.00         0.00    3280818          0          0
loop8             0.02         2.58         0.00         0.00    1986468          0          0
loop9             0.00         0.00         0.00         0.00          8          0          0
xvda              0.76        10.79         5.69         0.00    8323608    4387964          0
```

The *iostat* command generates the CPU Utilization report and the Device Utilization report, and the Network File System report.

18. As a system administrator, I feel like the most useful commands for debugging server issues are *netstat -a* and *top*. The *netstat* command will invoke a network statistics tool that can display incoming/outgoing network connections, protocol statistics, and routing tables, which is useful for measuring network traffic and solving outages, slowdowns, or bottleneck issues on a network. The *top* command provides a real-time view of system processes. Together, these commands can solve hardware and network issues for servers.