Shell scripting or programming mostly consists of the features which today's modern programming languages offer. We have driven next set of interview questions.

1. **How do you terminate a shell script if statement?**

With fi, which is "if" spelled backwards.
Details:
The shell script example below uses an if statement to check if a file assigned to the variable myfile exists and is a regular file:

```
#!/bin/ksh

myfile=$1

if [ -f $myfile ]

then

echo "$myfile exists"

fi

exit 0
```

2. **What UNIX operating system command would you use to display the shell's environment variables?**

Running the "env" command will display the shell environment variables.
Details:
Sample env command output:

```
# env

HISTFILE=/home/lfl/.history

PATH=/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin

SHELL=/bin/ksh
```

```
HOSTNAME=livefirelabs.com

USER=lfl

MAIL=/var/spool/mail/lfl

HOME=/home/lfl

HISTSIZE=1000
```

It would also be good to understand the purpose of the common shell environment variables that are listed in the env command output.

3. **What code would you use in a shell script to determine if a directory exists?**

The UNIX test command with the -d option can be used to determine if a directory exists.
Details:
The following test command expression would be used to verify the existence of a specified directory, which is stored in the variable $mydir:

```
if [ -d $mydir ]

then

command(s)

fi
```

If the value stored in the variable mydir exists and is a directory file, the command(s) located between then and fi will be executed.

You can consult the test command's man page ("$ man test") to see what test command options are available for use.

4. **How do you access command line arguments from within a shell script?**

Arguments passed from the command line to a shell script can be accessed within the shell script by using a $ (dollar sign) immediately followed with the argument's numeric position on the command line.

Details:
For example, $1 would be used within a script to access the first argument passed from the

command line, $2 the second, $3 the third and so on. ***Bonus:*** $0 contains the name of the script itself.

5. **How would you use AWK to extract the sixth field from a line of text containing colon (:) delimited fields that is stored in a variable called passwd_line?**

```
echo $passwd_line | awk -F: '{ print $6 }'
```

Details:
Consider this line of text stored in the variable $passwd..

```
# echo $passwd.
mail:x:8:12:mail:/var/spool/mail:
```

(***Background:*** The lines in the system passwd file are delimited (separated) by a colon (:)…$passwd_line contains a single line from the passwd file.)

The output of the echo command is piped to AWK. The -f option for the awk command informs awk of what the field separator is (colon in this example), and print $6 instructs awk to print the 6th field in the line.

6. **What does 2>&1 mean and when is it typically used?**

The 2>&1 is typically used when running a command with its standard output redirected to a file. For example, consider:

```
command > file 2>&1
```

Anything that is sent to command's standard output will be redirected to "file" in this example.

The 2 (from 2>&1) is the UNIX file descriptor used by standard error (stderr). Therefore, 2>&1 causes the shell to send anything headed to standard error to the same place messages to standard output (1) are sent…which is "file" in the above example.

To make this a little clearer, the > in between "command" and "file" in the example is equivalent to 1>.

7. **Within a UNIX shell scripting loop construct, what is the difference between the break and continue?**

Using break within a shell scripting loop construct will cause the entire loop to terminate. A continue will cause the current iteration to terminate, but the loop will continue on the next iteration.

8. **What is the significance of $#?**

$# shows the count of the arguments passed to the script.

9. **What is the difference between $* and $@?**

$@ treats each quoted arguments as separate arguments but $* will consider the entire set of positional parameters as a single string.

10. **Given a file, replace all occurrence of word "ABC" with "DEF" from 5th line till end in only those lines that contains word "MNO"**

```
sed –n '5,$p' file1|sed '/MNO/s/ABC/DEF/'
```

11. **Explain about "s" permission bit in a file?**

"s" bit is called "set user id" (SUID) bit.

"s" bit on a file causes the process to have the privileges of the owner of the file during the instance of the program.

For example, executing "passwd" command to change current password causes the user to writes its new password to shadow file even though it has "root" as its owner.

12. **How can any user find out all information about a specific user like his default shell, real-life name, default directory, when and how long he has been using the system?**

finger "loginName" ...where loginName is the login name of the

user whose information is expected.

13. **What is the difference between $$ and $!?**

$$ gives the process id of the currently executing process whereas $! Shows the process id of the process that recently went into the background.

14. **What are zombie processes?**

These are the processes which have died but whose exit status is still not picked by the parent process. These processes even if not functional still have its process id entry in the process table.

15. **What will happen to my current process when I execute a command using exec?**

"exec" overlays the newly forked process on the current process; so when I execute the command using exec, the command gets executed on the current shell without creating any new processes.

E.g., Executing **"exec ls"** on command prompt will execute ls and once ls exits, the process will shut down

16. **What is the difference between grep and egrep?**

egrep is Extended grep that supports added grep features like "+" (1 or more occurrence of a previous character),"?"(0 or 1 occurrence of a previous character) and "|" (alternate matching)

17. **Write a command sequence to find all the files modified in less than 2 days and print the record count of each.**

```
find . -mtime -2 -exec wc -l {} \;
```

18. **What are "c" and "b" permission fields of a file?**

“c “ and “b” permission fields are generally associated with a device file. It specifies whether a file is a special character file or a block special file.

19. How will you abort a shell script before it is successfully executed?

We need to use ‘exit’ command to fulfil the above described situation. A ‘exit’ command when forced to output any value other than 0 (zero), the script will throw an error and will abort. The value 0 (zero) under Unix environment shell scripting represents successful execution. Hence putting ‘exit -1’, without quotes before script termination will abort the script.

For example, create a following shell script as ‘**test.sh**‘.

```
#!/bin/bash

echo "Hello"

exit -1

echo "foxutech"
```

Save the file and execute it.

# sh test.sh

```
Hello

exit.sh: 3: exit: Illegal number: -1
```

From the above script, it is clear that the execution went well before exit **-1** command.

20. How will you check the length of a line from a text file?

‘sed’ command is used to find or check the length of a line from a text file.

A ‘sed –n ‘n p’ file.txt‘, where ‘n‘ represents the line number and ‘p‘ print out the pattern space (to the standard output). This command is usually only used in conjunction with the -n command-line option. So, how to get the length count? Obviously! we need to pipeline the output with ‘wc‘ command.

```
# sed –n 'n p' file.txt | wc –c
```

To get the length of line number ‘5’ in the text file ‘foxutech.txt‘ we need to run.

```
# sed -n '5 p' foxutech.txt | wc -c
```

21. **If any of a group of staffs working for a company abc. The company ask you to create a directory 'dir_abc, such that any member of the group can create a file or access a file under it, but no one can delete the file, except the one created it. what will you do?**

An interesting scenario to work upon. Well in the above said scenario we need to implement the below steps which is as easy as cake walk.

```
# mkdir dir_abc

# chmod g+wx dir_abc

# chmod +t dir_abc
```

The first line of command create a directory (dir_abc). The second line of command above allow group (g) to have permission to 'write' and 'execute' and the last line of the above command – The '+t' in the end of the permissions is called the 'sticky bit'. It replaces the 'x' and indicates that in this directory, files can only be deleted by their owners, the owner of the directory or the root superuser.

22. **Can you tell me the various stages of a Linux process; it passes through?**

A Linux process normally goes through four major stages in its processing life.

Here are the 4 stages of Linux process.

Waiting: Linux Process waiting for a resource.

Running: A Linux process is currently being executed.

Stopped: A Linux Process is stopped after successful execution or after receiving kill signal.

Zombie: A Process is said to be 'Zombie' if it has stopped but still active in process table.

23. **What is the use of cut command in Linux?**

A 'cut' is a very useful Linux command which proves to be helpful when we need to cut certain specific part of a file and print it on standard output, for better manipulation when the field of the file and file itself is too heavy.

For example, extract first 10 columns of a text file 'txt_foxutech'.

```
# cut -c1-10 txt_foxutech
```

To extract 2nd, 5th and 7th column of the same text file.

```
# cut -d;-f2 -f5 -f7 txt_foxutech
```

24. **What is inode? Brief about it.**

A 'inode' is a 'data-structure', which is used for file identification on Linux. Each file on a Unix System has a separate 'inode' and an 'Unique' inode Number.

25. **How to pass argument to a script?**

```
./script argument
```

Example: Script will show filename

```
./show.sh file1.txt

cat show.sh

#!/bin/bash

cat $1
```