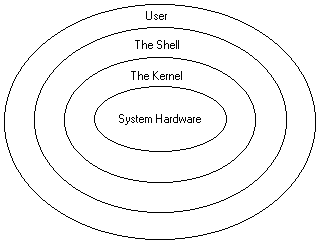
**20 UNIX Shell Scripting Interview Questions...and Answers!!! - Part I**

As someone who has written a large number of both simple and complex UNIX shell scripts during the past 15 years, the following are some basic shell scripting interview questions that I would ask if I needed to hire someone to write new shell scripts or maintain preexisting scripts for me.  
  
These type of questions would help me to determine If the candidate for the position has spent any time actually writing shell scripts, either on the job or as part of a shell scripting course or training class, or has just read about them in books or online.  
  
Let's get started...  
  
**(1) What is a UNIX shell?**  
The UNIX shell is a program that serves as the interface between the user and the UNIX operating system. It is not part of the kernel, but communicates directly with the kernel. The shell translates the commands you type in to a format which the computer can understand. It is essentially a command line interpreter.  
  
**Details:**  
Visual representation of the UNIX operating system environment:



List of commonly used UNIX shells:  
  
· The Bourne Shell (sh)  
· The C Shell (csh or tsch)  
· The Bourne Again Shell (bash)  
· The Korn Shell (ksh)  
  
  
**(2) What needs to be done before you can run a shell script from the command line prompt?**  
You need to make the shell script executable using the UNIX chmod command.  
  
**Details:**  
This chmod command makes the shell script file "example1" executable for the user (owner) only:

$ chmod u+x example1

this syntax makes it executable for all (everyone):

$ chmod a+x example1

You can optionally use octal notation to set UNIX permissions using the chmod command (e.g., $ chmod 755 example1). This topic is beyond the scope of this article, but you can find more information by entering "unix file permissions chmod numeric notation" in your favorite search engine.  
  
  
**(3) 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

(See shell scripting interview question #6 below if you do not know what $1 in this example means.)  
  
  
**(4) 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.  
  
  
**(5) 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.  
  
  
**(6) 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.  
  
  
**(7) 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\_line -

$ echo $passwd\_line  
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.  
  
  
**(8) 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>.  
  
**Before you continue reading... Has this article been helpful to you? Would it benefit others? If you answered "yes" to either question, kindly share the page.** 



**MORE READERS = MORE FUTURE ARTICLES  
Thank you for sharing.**   
  
**(9) What are some ways to debug a shell script problem?**  
Although this is somewhat dependent on what the problem is, there are a few commonly used methods for debugging shell script problems.   
  
One method, which is frequently used across all programming languages, is to insert some debug statements within the shell script to output information to help pinpoint where and why the problem is being introduced.   
  
Another method specific to shell scripting is using "set -x" to enable debugging.   
  
**Details:**  
Consider the following shell script example (notice that "set -x" is commented out at this time):

#!/bin/ksh  
#set -x  
i=1  
while [ $i -lt 6 ]  
do  
print "in loop iteration: $i"  
((i+=1))  
done  
exit

This script will produce the following output:

$ ./script1  
in loop iteration: 1  
in loop iteration: 2  
in loop iteration: 3  
in loop iteration: 4  
in loop iteration: 5  
$

If we uncomment (remove the #) from the "set -x" line in the script, this is what is displayed when the script runs:

$ ./script1  
+ i=1  
+ [ 1 -lt 6 ]  
+ print in loop iteration: 1  
in loop iteration: 1  
+ let i+=1  
+ [ 2 -lt 6 ]  
+ print in loop iteration: 2  
in loop iteration: 2  
+ let i+=1  
+ [ 3 -lt 6 ]  
+ print in loop iteration: 3  
in loop iteration: 3  
+ let i+=1  
+ [ 4 -lt 6 ]  
+ print in loop iteration: 4  
in loop iteration: 4  
+ let i+=1  
+ [ 5 -lt 6 ]  
+ print in loop iteration: 5  
in loop iteration: 5  
+ let i+=1  
+ [ 6 -lt 6 ]  
+ exit  
$

In addition to displaying the intended output ("in loop iteration" lines), enabling debugging with "set -x" also shows each line of execution preceded by a plus sign (+).  
  
Although this can become unwieldly for larger shell scripts, it should be obvious how useful this can be when debugging a shell script to identify the root cause of a problem.  
  
  
**(10) 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.  
  
  
**Do you need to practice writing and executing UNIX shell scripts (on a REAL SERVER) before the interview?**With a highly concentrated effort, either of these online courses can be completed within 24 hours...

[UNIX and Linux Operating System Fundamentals](http://www.livefirelabs.com/UNIX_Linux_Fund-tip-4.htm) contains a very good "Introduction to UNIX Shell Scripting" module, and should be taken if you are new to the UNIX and Linux operating system environments or need a refresher on key concepts.  
  
[UNIX Shell Scripting](http://www.livefirelabs.com/Korn_Shell_Scripting-tip-4.htm) is a good option if you are already comfortable with UNIX or Linux and just need to sharpen your knowledge about shell scripting and the UNIX shell in general.

...or you can optionally select a subset of modules from the course to focus on prior to the interview and then return to work on the remaining topics as your schedule permits.  
  
Both courses include access to an Internet Lab system for completing the course's hands-on exercises, which are used to re-enforce the key concepts presented in the course. Any questions you may have while taking the course are answered by an experienced UNIX technologist.  
  
Thanks for reading, and best of luck with your shell scripting interview questions and your interview in general!!!