Secure Scripting

**assessment questions**

# Unit 1. The Basics

Learning Objective 1.1

In which of the following situations would scripting be more appropriate than writing a compiled program?

1. You need to carry out a computation that requires high performance and must be as quick as possible.
2. You need to carry out an administrative task repeatedly, and the task does not require maximum efficiency.
3. Your task requires the use of several specialized libraries (modules) that have interfaces designed for programs written in C (Java).
4. Your computational task requires use of a low-level resource.

Answer: b. Compiled programs are faster than scripts, so a is wrong. Calling specialized libraries in a given programming language is best done using that language (or a compatible one), so c is wrong. Shell scripts work at the user level, so low-level resources must be accessed using other programs, so d is wrong. Scripts are typically used for administrative tasks that must be repeated and for which speed is not critical, so b is correct.

Learning Objective 1.1 (also 1.2, 1.3, and 1.5)

Write a script called “frame.sh” that takes a filename as an argument and prints the filename followed by a colon, then the contents of the file beginning on the next line. If no file is named, print an error message and exit.

Answer: See the file saved as “frame.sh” in the directory 21.SeS\_AssessmentAnswerFiles. Note that the error message in line 4 may be worded differently.

Learning Objective 1.2

What does the line

#! /bin/sh

mean?

1. The script is named /bin/sh .
2. Nothing. Because the first character is “#”, this line is a comment.
3. When the script is run, it is to be interpreted by the program /bin/sh.
4. It should be the last line in the script, and it causes the script to exit cleanly.

Answer: c. See Slide 5 in the Unit 1 presentation.

Learning Objective 1.2

Write a shell script that prints “It is now” followed by the date and time.

*Hint*: Use the command *date* to print the time.

Answer: See the file saved as “prdate.sh” in the directory 21.SeS\_AssessmentAnswerFiles.

Learning Objective 1.3

A user types the command line

scriptus 3 abc gleep

What do the references $0, $3, and $# refer to?

1. scriptus, gleep, 3
2. scriptus 3 abc gleep, 3, 4
3. (the empty string), abc, 3
4. gleep, 3, 4

Answer: $0 refers to the command name, in this case, “scriptus”. $3 refers to argument 3, which is “gleep”. $# refers to the number of arguments, which is three. So the answer is a.

Learning Objective 1.3

The following script is called “xyzzy.sh”:

#! /bin/sh

echo $1 - $2

What does

sh xyzzy.sh 5 3

print?

1. 5 – 3
2. 2
3. xyzzy.sh – 5
4. sh – xyzzy.sh

Answer: In this command, argument 1 is 5 and argument 2 is 3. So c and d are wrong. *Echo* prints the text it is given, and neither it nor the shell script do arithmetic, so a is correct.

Learning Objective 1.3

The following script is called “hello.sh”:

#! /bin/sh

cat ‘$1’ “$2” $3

What command does

sh hello.sh A “B C” D E

execute?

1. cat A B C D E
2. cat $1 B C D
3. cat $1 A B C
4. cat A B C D

Answer: In this command, argument 1 is A, argument 2 is B C (because of the double quotes), argument 3 is D, and argument 4 is E. The single quotes around the $1 in the cat command cause that to be interpreted as the string $1 by the shell. Therefore the cat command is:

cat $1 B C D

So the answer is b.

Learning Objective 1.4 (also 1.2, 1.3, and 1.5)

Write a script called “choix.sh” that prints “hello” if it is called with an argument of 1, “goodbye” if it is called with an argument of 2, and “oops…bad argument” if it is called with any other argument.

Answer: See the file saved as “choix.sh” in the directory 21.SeS\_AssessmentAnswerFiles.

Learning Objective 1.4

If this is in a script that is run with three arguments, what will be printed?

if [ $# = 2 ]

then

echo “There are 2 arguments”

elif [ $# = 3 ]

then

echo “There are 3 arguments”

else

echo “I don’t know how many arguments you gave me”

fi

1. There are 2 arguments
2. There are 3 arguments
3. I don’t know how many arguments you gave me
4. None of the above

Answer: As there are three arguments, $# is 3. Because the [ … ] has =, the comparison is a string comparison, and the number for $#, 3, is converted to the string “3”. This matches the “3” on the other side of the “=” sign, so the condition of the first “elif” is true. Thus, the answer is b.

Learning Objective 1.5 (also 1.2, 1.3, and 1.4)

Write a script called “check.sh” that checks to see if a file named “MasterList” exists and is readable in the current working directory. If it exists, say so; if not, give an error message and exit.

Answer: See the file saved as “check.sh” in the directory 21.SeS\_AssessmentAnswerFiles.

Learning Objective 1.5 (also 1.2, 1.3, and 1.4)

Write a script that prints its argument. If none is given, print “No arguments given”; if one is given but is of zero length, print “The argument is empty.”

Answer: See the file saved as “prarg.sh” in the directory 21.SeS\_AssessmentAnswerFiles. Note the script prints argument 1 if there is more than one argument and argument 1 is not of zero length. Students can handle this case any way they like.

# Unit 2. Advanced Control

Learning Objective 2.1

When is it appropriate to combine two scripts into one that performs the same function?

1. When you need to save disk space, and combining the files containing the scripts will be smaller than the two files containing the separate scripts
2. When the functions that the scripts perform are tightly coupled
3. When the scripts use the same control structures
4. When the output of one script is used as the input to the other script

**Answer: Saving disk space is not a good reason to combine scripts that do two different things, so a is wrong. Similarly, two scripts may do two different things using similar control structures, so c is wrong. As the output of one script can be used as input to another script (either using files or pipes), d is not a good reason. But if two scripts work together to perform a single operation, then it is appropriate to combine them, so b is correct.**

Learning Objective 2.1 (also 2.6)

Here are two scripts that perform basic arithmetic operations. Please merge them into one.

|  |  |
| --- | --- |
| mult.sh:  #! /bin/sh  expr $1 '\*' $2  exit 0 | intdiv.sh:  #! /bin/sh  expr $1 / $2  exit 0 |

The new script should take three arguments, the third one being "-m" to multiply and "-d" to divide. So, to multiply 6 and 3, your script would be given the arguments "6 3 -m"; to divide 6 by 3, the arguments would be "6 3 -d". You can assume the first two arguments are integers but cannot assume the third argument is "-m" or "-d". So don't forget to add error-checking!

**Answer: See the file saved as “new2.sh” in the directory** 21.SeS\_AssessmentAnswerFiles.

Learning Objective 2.2

Write a shell script that tests a list of filenames to see if the files exist. For each one that does not exist, it should write a message saying "file *name* does not exist" (where *name* is the filename). The list of files will be given on the command line.

**Answer: See the file saved as “filetest.sh” in the directory** 21.SeS\_AssessmentAnswerFiles.

Learning Objective 2.2 (also 2.3)

What files does the loop

for word in hello goodbye "auf weidersehen" adieu

do

ls $word

done

print the attributes of? Assume all the files exist.

***Answer*: On the third iteration through the loop, the value of the variable “word” is the string “auf weidersehen” (without the quotes). So the *ls* command is:**

**ls auf weidersehen**

**(again, without the quotes). As neither file exists, *ls* prints two error messages. So the output is:**

**hello**

**goodbye**

**ls: cannot access 'auf': No such file or directory**

**ls: cannot access 'weidersehen': No such file or directory**

**adieu**

Learning Objective 2.3

What does the following script print, assuming there is a command called "date" that prints the date?

#! /bin/sh

VAR= date

echo $VAR

exit 0

1. A single line with the word "date" on it
2. Nothing; the space after the "=" sign gives an error
3. The date followed by a blank line
4. The date

**Answer: c. There is a blank between the “=” and the word “date” in line 2, so VAR is set to an empty string, not date. Thus, a and b are both wrong as “date” is interpreted as a command and executed. The line with echo prints a blank line. So c is right and d is wrong.**

Learning Objective 2.3

The following script is called “plugh.sh”:

#! /bin/sh

A=1

B=3

C="+"

echo $A '$C' $B

What does it print?

1. 1 + 3
2. $A $C $B
3. 1 $C 3
4. 1 "+" 3

**Answer: c. $A and $B interpolate the value of the variables A and B, so b is wrong. ‘$C’ is $C, not the value of the variable C, because of the single quotes, so a and d are wrong.**

Learning Objective 2.4

Which of the following tests whether the numeric value of the variable X is between 5 and 100 (not inclusive), or exactly 500?

1. if [ ( 5 -lt $X -a $X -lt 100 ) -o $X -eq 500 ]
2. if [ 5 -lt $X -o $X -lt 100 -o $X -eq 500 ]
3. if [ 5 -lt $X -a $X -lt 100 -a $X -eq 500 ]
4. if [ 5 -lt $X -a [ $X -lt 100 -o $X -eq 500 ] ]

**Answer: a. b is true if the value of X is greater than 5, or less than 100, or equal to 500; c is true if the value of X is between 5 and 100 and equal to 500 (an impossibility); and produces an error, as “[“ is a command and not a grouping operator like “(“.**

Learning Objective 2.4

Write a Boolean expression that is true when the program "id -nu" prints the word "root" or when the program "whoami" prints the word "thomas".

**Answer: [ “`id –nu`” = “root” –o “`whoami`” = “Thomas” ]**

Learning Objective 2.5

What is the purpose of an option in a script?

**Answer: An option changes the way a script works but only for that particular execution of the script. See Slide 40 in the Unit 2 presentation, 08.SeS\_Unit2\_AdvancedControl\_Presentation.**

Learning Objective 2.5

The option "-x" requires special processing at the end of the script, *after* all the other arguments have been processed and the script has performed all operations using them. Which of the following is the best way to cause the script to do the special processing?

1. Save the arguments in a variable, and then go through the value of that variable at the end of the script looking for the "-x" option.
2. When the arguments are being processed, if one is "-x", set a variable to indicate that option was encountered. Then, just before the special processing would be done, check the value of that variable to determine if the "-x" option is set.
3. Recheck the arguments directly just before the special processing would be done, to determine if the "-x" option is present.
4. When the option "-x" is encountered, create a file named "special$$". Then, just before the special processing would be done, see if that file exits. If it does, do the special processing. Then delete that file.

**Answer: a and c both require rescanning all the options, which is unnecessary. d requires a file to be created, which again is unnecessary. b is by far the best, because it requires only one pass through the argument list.**

Learning Objective 2.6 (also 2.4 and 2.5)

The script "doarith.sh" takes as arguments a number, a "+" or "-" sign, and another number. Write a Boolean expression that checks the second argument to be sure it is either a "+" or "-", and prints an appropriate error message if not.

**Answer: Here is the relevant if statement.**

**if [ "$2" != '+' -a "$2" != '-' ]**

**then**

**echo "$2 is neither + nor -" 1>&2**

**fi**

**You can also do this in a single line (which most students will not do):**

**[ "$2" != '+' -a "$2" != '-' ] && echo "$2 is neither + nor -" 1>&2**

Learning Objective 2.6

The following script produces an error for some argument(s).

#! /bin/sh

for str in "$@"

do

expr $str + 1

done

exit 0

Determine what that (those) argument(s) are.

**Answer: The obvious ones are any non-integers. A much less obvious one is an integer beginning with “+”; students will discover this by trying it out. Amusingly enough, integers beginning with “-“ (i.e., negative integers) do not produce an error!**

# Unit 3. Advanced Scripting

Learning Objective 3.1

How do you read data from a file one line at a time?

1. Use a *while* loop, and make the command (condition) in the *while* be the command *read*; then give *read* the name of the variable to hold the line.
2. Use a *for* loop, and after the “in” put the filename.
3. Give the *read* command with the filename as an argument.
4. Use a *for* loop, and after the “in” put the *read* command followed by the filename.

**Answer: a. In a *for* statement, what follows the “in” is a list of values that the variable will take, so b is wrong. The argument to *read* is the variable in which the input line is to be stored, so c is wrong. For d, the variable in the *for* loop will be set to the string value “read” for the first iteration, and the filename for the second, so d is wrong.**

Learning Objective 3.1

A file has seven fields in it, separated by colons. What should IFS be set to, in order to enable the script to read the line as seven separate fields?

**Answer: The colon (“:”) character. See Slides 11 and 12 in 14.SeS\_Unit3\_AdvancedScripting\_Presentation.**

Learning Objective 3.2

When should you use a *while* loop instead of a *for* loop?

**Answer: Use a for loop when you know the number of times that the loop is to be executed (it’s the number of values in the for statement). Use a while statement when you do not (as, for example, when reading lines from a file).**

Learning Objective 3.2

When reading a file using redirection and a while loop, the redirection is placed after the loop and not after the while statement. Why?

**Answer: The contents of the file are to be applied to the entire loop, one iteration per line. If you put the redirection after the while statement but before the “do” line, it will be read once, and you will do the loop repeatedly only for the first line—really, an infinite loop. See Slide 8 in 14.SeS\_Unit3\_AdvancedScripting\_Presentation.**

Learning Objective 3.3

What commands or constructs are used to do arithmetic in a shell script?

**Answer: There are several possible correct answers. The one used most in Unit 3 has been the expr command, but as Slide 20 in 14.SeS\_Unit3\_AdvancedScripting\_Presentation points out, the let construct could be used also. Another one for the Bourne shell is the double parentheses: “((“ and “))”. You have to do an assignment to a variable in them and then print the value of the variable using echo. We did not discuss this option in Unit 3.**

Learning Objective 3.3

The following script is called “avast.sh”:

#! /bin/sh

A=1

B=3

C="+"

Which of the following constructs adds the values of A and B and assigns that value to SUM?

1. SUM=`[ $A $C $B ]`
2. SUM=`expr $A $C $B`
3. SUM=( $A + $B )
4. SUM=`sed $A + $B`

**Answer: b. Of these, the only one that does arithmetic is the *expr* command. So the correct answer must be b.**

Learning Objective 3.4

What is the value of the variable X after the following line is executed?

X=`echo “abcde abcde 123” | sed ‘s/abcde/xxxxx/’`

**Answer: The point of this exercise is that the *sed* substitution command (“s/abcde/xxxxx/”) matches only the first occurrence of “abcde”, and replaces only that one with “xxxxx”. So the value of X is**

**xxxxx abcde 123**

Learning Objective 3.4

What value is stored in WEIRD after the following is executed?

WEIRD=0

WEIRD=”$WEIRD `expr 9 + 21`”

WEIRD=”$WEIRD `echo $WEIRD | sed ‘s/ /X/’`”

**Answer: The first line sets WEIRD to 0. The second line concatenates the value of 9 + 21 to the current value of WEIRD, separated by a blank, so the new value of WEIRD is**

**0 30**

**The third line concatenates to the current value of WEIRD, followed by a blank, the result of changing the blank in the value of WEIRD to “X”. So the final value of WEIRD is**

**0 30 0X30**

Learning Objective 3.5

What is the value of the variable X after the following line is executed?

X=`echo “abcde abcde 123” | sed ‘s/\(.\*\)23/\1/’`

**Answer: The first pattern in the *sed* expression (“\(.\*\)23”) matches the longest string that ends in “23”. The “\1” means to repeat whatever is matched by the pattern “\(.\*\)” in the first part of the *sed* expression. As *echo* prints its argument, and that is the input to *sed*, the “\1” is “abcde abcde 1”, and that is what is printed. So the output is**

**abcde abcde 1**

Learning Objective 3.5

Which of the following does the pattern “\(.\*\) \1” match?

1. abcde abc
2. abcde edcba
3. abcde abcde
4. 1234 5687

**Answer: The “\1” matches what the pattern in the first “\( \)” matches. There is a blank space following the “\)”, so the pattern in the first “\( \)” ends in a blank. As “.\*”, which matches 0 or more characters, is between “\(“ and “\(“, the pattern “\(.\*\) “ matches the longest string that ends in a blank. So “\(.\*\) \1” matches some pattern, a blank, and then that first pattern repeated once. As c is the only string that this describes, the correct answer is c.**