14.3.5 Bash Scripting Logic Facts

In addition to the basic commands, scripts can contain scripting logic constructs to control the flow of the script.

This lesson covers the following topics:

- Branching constructs
- Looping constructs
- Exit codes

Branching Constructs

There are two branching constructs that perform different actions based on specific conditions or user input.

Construct	Description	Examples
if/then/else/fi	Use the **if/then/elif/else/fi** construct evaluate conditions and branch based on the results. ▪ The **if** statement defines the condition to be evaluated. ▪ The **then** statement specifies the commands to perform if the condition evaluates to *true*. ▪ The **elif** statement is run if the condition in the if statement evaluates to false. It also supplies another condition and specifies the commands to perform in the **elif** condition evaluates to *true*. ▪ The **else** statement specifies the commands to perform if the condition evaluates to *false*. ▪ Operands include: ▪ = (equal to) ▪ **!=** (not equal to) ▪ > (greater than) ▪ < (less than) ▪ The **if** command requires spaces between the conditions and the operand. ▪ The **if** command also requires spaces between the conditions and the brackets ('[' and ']'). ▪ The construct must be closed with the **fi** command. The **test** command can be used within an **if/then/else/fi** construct to evaluate whether a condition evaluates to *true* or *false*. The **test** command options include: ▪ **-d** tests whether a directory exists. ▪ **-e** tests whether a file exists. ▪ **-f** tests whether a regular file exists. ▪ **-G** tests whether the specified file exists and is owned by a specific group. ▪ **-h** or **-L** tests whether the specified file exists and if it is a symbolic link. ▪ **-O** tests whether the specified file exists and if it is owned by a specific user. ▪ **-r** tests whether the specified file exists and if the read permission is granted. ▪ **-w** tests whether the specified file exists and if the write permission is granted. ▪ **-x** tests whether the specified file exists and if the execute permission is granted. Operands for **test** include: ▪ = tests whether strings are equivalent. ▪ **!=** tests whether strings are not equivalent. ▪ **-o** is used to specify that either of the options can be equivalent. ▪ **-eq** tests whether integers are equivalent. ▪ **-ne** tests whether integers are not equivalent. ▪ **-gt** tests whether the first integer is greater than the second. ▪ **-lt** tests whether the first integer is less than the second.	**#!/bin/bash** **echo "What is your name?"** **read NAME** **if [$NAME = "George"]** **then** **echo "That's my name, too."** **else** **echo "Hello" $NAME "I'm George."** **fi** **exit 0** The script takes the user input and evaluates it to determine whether the user types *George*. If the user types *George*, the script responds with *That's my name, too.* Otherwise, the script responds with the statement under the *else* command. The if statement could also be written using the test command. For example, **if test $NAME = "George"** **test -f ~/myfile.txt** determines whether a file named myfile.txt exists in the user's home directory. **test -d ~/binin** determines whether a directory named bin exists in the user's home directory. **test $name = "George Washington"** determines whether the value of the $name variable is George Washington. **test $name -o $name2 = "George Washington"** determines whether the value of either the $name or the name2 variable is George Washington. **test $num1 -le $num2** determines whether the value of the variable num1 is less than the value of the variable num2.
case/esac	Use the **case/esac** construct to branch a script when the condition being	**#!/bin/bashsh**

	evaluated has several possible outcomes. ▪ The case construct can have an unlimited number of possible options. ▪ When the script evaluates an option as being true, all remaining options are skipped. ▪ Each option can execute several lines of commands. ▪ Close each case with two semi-colons. ▪ Case structures must be closed using **esac** (case spelled backwards.)	**echo "What is your favorite season?"** **read season** **case $season in** **spring)** **echo "The thing I like best about spring is the flowers."** **;;** **summer)** **echo "I wish I could go swimming, but being a computer, that might not work out so well."** **;;** **fall)** **echo "Fall leaves, crisp morning air...what's not to like?"** **;;** **autumn)** **echo "Fall leaves, crisp morning air...what's not to like?"** **;;** **winter)** **echo "Skiing looks fun, but snowstorms interfere with my reception of the neighbor's Wi-Fi."** **;;** ***)** **echo $SEASON "is not listed in my database as being a season. Choose: spring, summer, fall, autumn, or winter."** **;;** **esac** **exit 0** The script asks the user about season preferences and has a response for each common answer. The last option is a catch-all for any answer other than those specified in the script.

Looping Constructs

There are three looping constructs that repeatedly run a set of commands according to a specified set of conditions.

Construct	Description	Examples
while/do/done	The **while/do/done** construct implements a **while** loop. A **while** loop continuously executes all commands between the **do** and **done** statements while a specific condition evaluates to *true*. The **while** loop is useful for repeating an action until a specified condition is met. ▪ A **while** loop requires do and done statements. ▪ A **while** loop can create infinite loops if the condition never evaluates to *false*.	**#!/bin/bash** **declare -i num** **echo "I'm thinking of a number between 1 and 100."** **# Give num a value to prevent errors.** **num=0** **while test $num -ne 23** **do** **echo "What is your guess?"** **read num** **if test $num -lt 23** **then** **echo "The number is higher."** **fi** **if test $num -gt 23** **then** **echo "The number is lower."** **fi** **done** **echo "You guessed it."** **exit 0** This script uses a **while** loop to keep the user guessing numbers until they get the answer. As long as the number is not 23, the if statements keep giving the user clues as to whether the number is higher or lower. As soon as the user types 23, the **while** loop exits and the final

		statement displays telling the user that the guess is correct.
until/do/done	The **until/do/done** construct implements an until loop. An **until** loop is nearly identical to a **while** loop, but evaluates the condition using the opposite logic: ▪ The e **while** loop executes commands while a condition is true. ▪ The **until** loop executes commands until a condition is true.	**#!/bin/bash** **declare -i num** **echo "I'm thinking of a number between 1 and 100."t;** **# Give num a value to prevent errors.** **num=0** **until test $num -eq 23** **dodo** **echo "What is your guess?"** **read num** **if test $num -lt 23** **thenen** **echo "The number is higher."** **fi** **if test $num -gt 23** **thenen** **echo "The number is lower."** **fi** **done** **echo "You guessed it."** **exit 0**
for/do/done	The **for/do/done** construct implements a **for** loop. A **for** loop executes a set of commands a set number of times. A **for** loop: ▪ Is useful when a specific action needs to be done a set number of times. ▪ Executes all commands between the do and done statements. These are required. ▪ Can be used with a list of items with one action being done for each item in the list. The **seq** command can be used to create a sequence of numbers for use in a for loop: ▪ When using only one number in the command, **seq** starts at 1 and counts to the specified number. ▪ When two numbers are given in the command, **seq** begins with the first number and counts up to the second number. ▪ When three numbers are given in the command, **seq** starts at the first number and counts in increments of the second number up to the third number.	**#!/bin/bash** **declare -i num** **echo "Multiplication quiz!!!";** **echo "Which set of multiplication tables do you want to drill?"** **read num** **echo "OK. We'll work on 0X"$num "through 12X"$num"."** **for looper in $(seq 0 12)** **do** **echo "What is" $num "X" $looper"?"** **read answer** **if test $ANSWE -eq $[$num*$looper]** **then** **echo "That is correct!"** **else** **echo "That's not it. The correct answer is" $[$num*$looper]"."** **fi** **done** **exit 0** The script uses a **for** loop to ask the correct number of multiplication questions. The loop starts at 0 and increments from 0 to 12. This ensures that the correct number of questions are asked. **seq 10** counts from 1 to 10. **seq 5 15** starts at 5 and counts to 15. **seq 5 5 100** starts at 5 and counts to 100 in increments of 5. (for example 5, 10, 15, 20...) **seq 10 -1 -10** starts at 10 and counts down to -10. Other methods used to create a **for** loop include: ▪ **for looper in 1 2 3 4 5** counts from 1 to 5. ▪ **for looper in {1..5}** counts from 1 to 5. ▪ **for looper in {0..10..2}** counts from

		0 to 10 in increments of 2.

Exit Codes

Every command terminates with an exit code. The **$?** special variable contains the exit code from the last executed command. In addition the **exit** command will both exit a script or routine and set an exit code.

The following examples illustrate how the **$?** special variable and the **exit** command can be used in scripts.

Example	Description
#!/bin/bash **echo "Enter the file to find:"** **read filename** **find $filename** **result=$?** **case $result in** **0)** **echo "The $filename file was found"** **;;** **1)** **echo "The $filename file was not found"** **;;** ***)** **echo "Unexpected error code: $result"** **;;** **esac** **exit 0**	The **find** command returns an exit code of 0 when a given filename is found. If it doesn't find the file, it returns with an exit code of 1. The exit code is captured in the **result=$?** command and used in the case structure.
#!/bin/bash **if ! [$(id -u) = 0]; then** **echo "The $0 script must be run as root"** **exit 101** **elif ! [-e /etc/aconffile.conf]; then** **echo "The configuration file /etc/aconffile.conf is missing"** **exit 102** **else** **echo "Performing normal processing"** **# commands to be performed by root ...** **fi** **exit 0**	The results of the **id -u** command is zero (0) if the root account is being used. If root is not being used the script exits with a code of 101. The **elif** statement checks if a configuration file exits. If not, the script exits with a code of 102. Otherwise (**else**), normal processing commands are run and the script exits with a code of zero (0).