Edith Cowan University
School of Science

CSI6203 Scripting Languages

Module 3

Conditions

Contents

Learning Objectives

After finishing this module, you should be able to:

- Understand and execute scripts that require decision making
- Write if statements to control program flow
- Write case statements to control program flow
- Test the output of a script

Decision making

- Scripts can make decisions to run certain commands only when a specific condition is met

- Conditional statements add intelligence to the scripts and allow them to be more robust and reliable

Command-line Lists

Command-line Lists

- Command-line lists are some of the simplest forms of decision making

- They behave in a similar way to boolean operators in programming languages
  - && (AND)
  - || (OR)

Command-line Lists

- The && (AND) operator can be used to link multiple commands together
- The second command will only execute if the first command succeeds

```
$ ./findCandy && ./eatCandy
```

Command-line Lists

- The || (OR) operator can also be used to link multiple commands together

- The second command will only execute if the first command fails

```
$ ./findCandy || ./beSad
```

Exit Status

- Whether a command or script succeeds or fails is determined by its exit status
- If the script ends in `exit 0` then it is considered a success
- Any other exit status is considered a failure

Exit Status

- The exit status of a script can also be read by checking the value of the $? variable

```
$ ./findCandy
$ echo $?
0
```

test

test

- The `test` command can be used to evaluate a true or false expression

- `test` will succeed (return an `exit` status of 0) if the provided expression is true
- `test` will fail (return an `exit` status of 1) if the provided expression is false

test

- The `test` command can be used to evaluate a true or false expression

- `test` will succeed (return an `exit` status of 0) if the provided expression is true
- `test` will fail (return an `exit` status of 1) if the provided expression is false

test

- Command-line list expressions using test:

```
#!/bin/bash
read -p "What is your name? " name
test $name = 'Rob' && echo "Hello Rob" && exit 0
echo "Your name isn't Rob"; exit 1
```

```
What is your name?
boris
Your name isn’t Rob
```

test

- There is also a shortcut for the `test` command
- The [command can be used with the same effect

test

- Command-line list expressions using [ ]:

```
#!/bin/bash
read -p "What is your name? " name
[ $name = 'Rob' ] && echo "Hello Rob" && exit 0
echo "Your name isn't Rob"; exit 1
```

```
What is your name?
Boris
Your name isn’t Rob
```

test booleans

- AND OR NOT boolean expressions can be used in the test statement to allow for more complex decision making

- -a (AND)
- -o (OR)
- ! (NOT)

test

- When would the following statement succeed?

```
[ "$name" = 'Frank' -a "$USER" = 'root' -o ! "$1" = 'apples' ]
```

- When the "name" variable is "Frank"
- and the current user is "root"
- or the first command-line argument is "apples"

Numeric comparisons

- test can also be used to test conditions about numbers

- -eq (equal to)
- -gt (greater than)
- -lt (less than)
- -ge (greater than or equal to)
- -le (less than or equal to)

test

- numeric comparisons using [ ]:

```
#!/bin/bash
[ $1 -gt 5 ] && echo "$1 is greater than five"
exit 0
```

```
$ ./script 20
20 is greater than five
```

File Test Conditions

- Various information about files can also be checked using test

Condition	Meaning
-e	File or directory exists
-d	Is a directory
-f	Is a normal file
-r	Is readable
-w	Is writeable
-x	Is executable
-nt	Is newer than
-ot	Is older than

test

- File comparisons using [ ]:

```
#!/bin/bash
[ assignment1.txt -nt backup.txt ] && cp assignment1.txt backup.txt
exit 0
```

```
$ ./script 20
20 is greater than five
```

Common errors

- Be careful of spacing!
  - Because [ ] is a command, there must be spaces between it and the conditions within it
- Use double quotes around variables!
  - Otherwise variables that contain spaces will be misinterpreted as multiple commands

if statements

If statements

- As decision making logic grows more complex, it can be easier and more manageable to use the “`if`” control structure instead of using && and ||
- If statements execute commands when an expression succeeds (similar to &&)

test

- Command-line list expressions using [ ]:

```
#!/bin/bash
read -p "What is your name? " name
[ $name = 'Rob' ] && echo "Hello Rob" && exit 0
echo "Your name isn't Rob"; exit 1
```

```
What is your name?
Boris
Your name isn’t Rob
```

test

- If statement expressions using [ ]:

```
#!/bin/bash
read -p "What is your name? " name
if [ $name = 'Rob' ]; then
    echo "Hello Rob"
    exit 0
else
    echo "Your name isn't Rob";
    exit 1
fi
```

```
What is your name?
Boris
Your name isn’t Rob
```

If statements

- Although Command-line lists are useful for short, simple logic, If statements are often more readable.
- The code within the “if” block will only execute if the condition evaluates to true
- The code within the “else” block will only execute if the condition evaluates to false

If statements

- Else is not required in if statements and the following two examples are effectively the same

```
#!/bin/bash
[ assignment1.txt -nt backup.txt ] && cp assignment1.txt backup.txt
exit 0
```

```
#!/bin/bash
if [ assignment1.txt -nt backup.txt ]; then
    cp assignment1.txt backup.txt
fi
exit 0
```

Combining tests in if statements

- test statements can be combined using command-line lists
- This allows for more complex logic in if statements

```
#!/bin/bash
if [ $1 = 'backup' ] && [ assignment1.txt -nt backup.txt ]; then
    cp assignment1.txt backup.txt
fi
exit 0
```

Elif

- if statements can be further extended by providing multiple branching paths

```
#!/bin/bash
if [ $1 = 'backup' ] && [ assignment1.txt -nt backup.txt ]; then
    cp assignment1.txt backup.txt
elif [ $1 = 'restore' ]; then
    cp backup.txt assignment1.txt
fi
exit 0
```

case statements

Case statements

- Another way to handle branching paths in scripts is to use the “`case`” statement
- This works in a similar way to the “`switch`” or “`select`” statements in other languages

Case statements

```
#!/bin/bash
case $1 in
    "HD")
        echo "High Distinction";;
    "D")
        echo "Distinction";;
    "CR")
        echo "Credit";;
    "C")
        echo "Pass";;
    "N")
        echo "Fail";;
    *)
        echo "Unknown Grade";;
esac
exit 0
```

Case statements

- The case statement expands the provided expression and then tries to match each case in turn
  - eg. `case $name in`
- When it finds a match in one of the cases, it will execute everything inside until it reaches a ;;
  - eg. `"Leisa")`
- If no cases match the expression, the default case will be executed
  - `*)`

Summary

- Terms to review and know include:
  - Decision making
  - Command-line Lists
  - Booleans
  - If statements
  - The test statement and [ ]
  - Case statements

References and Further Reading

- Ebrahim, M. and Mallet, A. (2018) Mastering Linux Based Scripting (2nd Ed) Chapter 3, pp 53-74

- http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-6.html
- http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_03.html
- http://wiki.bash-hackers.org/commands/classictest
-
-