Creating Conditional Statements

Andrew Mallett
LINUX AUTHOR AND CONSULTANT

@theurbanpenguin www.theurbanpenguin.com

Module Overview

- Simple AND / OR constructs
- Creating IF statements
- Arithmetic evaluations
- Comparing strings in evaluations
- Using regular expressions as part of a test
- Testing file attributes
- Using case statements

Conditional Expressions

The power of scripts really depends on the script being able to adapt to different conditions. There is no point trying to open a file that does not exist or create a user that already exists.

```
$ echo hello || echo bye
hello

$ echo hello && echo bye
hello
goodbye
```

Simple AND / OR Statements

The double vertical bar represents an OR statement. The second command only runs if the first fails

The AND is the double ampersand and the second command executes only if the first command succeeds.

```
$ declare -i days=30

$ if [ $days -lt 1 ] ;then  echo "Enter correct value"; fi

$ declare -i days=Monday

$ if [ $days -lt 1 ] ;then  echo "Enter correct value"; fi
Enter correct value
```

Simple IF

An if statement in the BASH or ZSH has at least one condition to test followed by one or more actions. The statement is terminated with the fi keyword.

```
$ declare -i days=31

$ if [ $days -lt 1 ] || [ $days -gt 30 ] ;
then  echo "Enter correct value"; fi
Enter correct value
```

Extending the Test with AND / OR

When we looked at the days variable before we said that we could easily test for incorrect values; those values being less than 0 or greater than 30.

```
$ declare -i days=31

$ if (( days < 1 || days > 30 )) ;
then  echo "Enter correct value"; fi
Enter correct value
```

Arithmetic Evaluation

Older shells are POSIX compliant and use the previous evaluation. Newer BASH and ZSH shells support advanced syntax for arithmetic evaluation, allowing simple notation and combing the OR within the test. The dollar can be omitted for variable name.

```
$ declare -i days=30

$ if (( days < 1 || days > 30 )) ;
then  echo "Enter correct value"; else echo "All good"; fi
All good
```

ELSE

Adding and ELSE test allows us to action both correct and incorrect input.

```
$ declare -i days

$ read days
Monday

$ if (( days < 1 )); then  echo "Enter numeric value";
elif (( days > 30 )) ; then echo "Too high" ;
else echo "The value is $days" ; fi
Enter numeric value
```

ELIF

If more than one condition needs to be test we can add elif statements. We can also use the command read to populate the variable, this is useful to gain input from a script.

Demo

We start be looking at simple conditional statements

```
$ declare -l user_name

$ read user_name
Bob

$ [ $user_name == 'bob' ] && echo "user is bob"
user is bob
```

Testing String Values

We can use the if statement as we have previously seen, but we can also use a simple test with AND / OR . NOTE: You can use either = or == to test string equality . I prefer == to differentiate from the assignment operator and to fit in with != and =~ both of which use two symbols.

```
$ declare -l browser

$ read browser
Firefox

$ [[ $browser == *fox ]] && echo "The browser's Firefox"
The browser's Firefox
```

Testing Partial String Values

Using the [[syntax in advanced shells we can test for partial values. The dollar is still required for the variable.

```
$ declare -l test_var

$ read test_var
color

$ [[ $test_var =~ colou?r ]] && echo "${BASH_REMATCH[0]}"
color
```

Testing Regular Expression

Using the [[syntax in advanced shells we use the match operator =~ to test for regular expressions. The result is also stored in the array BASH_REMATCH

Demo

Testing string values

```
$ type test [ [[
test is a shell builtin
[ is a shell builtin
[[ is a shell keyword
```

The Test Command

There is a command test. The [is a synonym for test and [[is the advanced test that should be used in precedence to [

```
$ test -f /etc/hosts && echo YES
YES

$ [[ -f /etc/hosts ]] && YES
YES
```

Testing File Attributes

We can test to see if a file is of a specific type. Here we check for a regular file.

-f

To test for a regular file

```
[[ -f /etc/hosts ]]
```

-d

Tests for a directory

```
[[ -d /etc ]]
```

Tests for symbolic link

[[-L /etc/localtime]]

-e

For the existence of a file no matter the type

```
[[ -e /etc/nologin ]]
```

-r

Tests for the read permission, r = read, w = write, x=execute

```
[[ -r /etc/hosts ]]
```

-k

Tests for the sticky bit

[[-k /tmp]]

-S

Tests for the SUID bit, use g for the GUID bit

```
[[ -s /bin/passwd]]
```

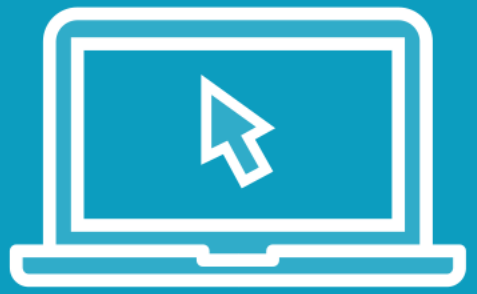
Demo

Working with the test command and file tests

Putting It All Together

```
#!/bin/bash
declare -l DIR
echo -n "Enter the name of the directory to create: "
read DIR
if [[ -e $DIR ]]
 then
  echo "A file exists with the name $DIR"
  exit 1
 else
  if [[ -w $PWD ]]
   then
    mkdir $DIR
   else
     echo "You cannot create $DIR within $PWD"
     exit 2
  fi
fi
```

Demo



Creating the demo script

```
case $USER in

  tux )
    echo "You are cool"
    ;;

  root )
    echo "You are the boss"
    ;;

esac
```

Case Statement

The case statement is more efficient the many elif statements being able to read the test condition just once. Start with case and terminates with esac. Each block ends with ;;

Pluralsight's Four Seasons

```
#!/bin/bash
declare -l month=$(date +%b)
case $month in
  dec | jan | feb )
    echo "Winter";;

  mar | apr | may )
    echo "Spring";;

  jun | jul | aug )
    echo "Summer";;

  sep | oct | nov )
    echo "Autumn";;
esac
```

Demo

Using the case statement

Summary

Simple AND && - simple OR ||

if < cond > ; then < action > ; fi

((days < 1))

((days < 1 || days > 30))

[[$month == jan]]

[[$month == j*]]

[[$month != jan]]

[[$month =~ y$]]

[[-s /bin/passwd]]

case $VAR in; <val1>); <action>;; esac

Next up:
Building Effective Functions