Systems Programming & Scripting

Lecture 14 - Shell Scripting: Control Structures, Functions

Control Structures

- Shell scripting supports creating more complex programs using control structures.
- Decision:
  - if statement
- Iteration (looping):
  - while/until statement
  - for statement

The if statement

if command
then
    statement
else
    statement
fi

- Similar in behaviour to the if statement in other programming languages with few differences:
  - The command is executed which could be a script. Based on the return value, the statement after the *then* or *else* get executed.

The if statement (cont'd)

- Return value 0 is true and any other value is considered to be false.

```
if script1; then
   ls –l
else
  echo “script1 returned false”
fi
```

The test Command

- The if statement is used to run executable.
- To use the if statement to only compare two values, it is required to run a program to do that.
- The *test* program is provided by the operating system for this purpose.
- The *test* program is invoked by running *[*, a symbolic link to /bin/test.

Example

```
#! /bin/sh

FIRST_ARGUMENT=“$1”
if [ x$FIRST_ARGUMENT = “xHans” ]; then
  echo “Hello Hans, good to see you”
else
  echo “Hello World”
fi
```

Example Explained

- Reading from standard input
  - Sending a message to standard output based on input.
- The *test* program is called to compare input with “Hans”
  - Note spaces after *[* and before *].*
  - Note spaces before and after =.
  - Those spaces are required.

Example Explained (cont'd)

- Two arguments in the comparison statement are proceeded by an *x*
  - To take into account when FIRST_ARGUMENT has no value.
  - Could also use double quote marks to solve the empty variable problem.

Example

```
#!/bin/bash
# copy a file, creating a backup if the target
file exists

if [ $# -lt 2 ]
then
  echo "Usage: $0 <fromfile> <tofile>"
  exit 1
fi

if [ -f $2 ]
 then mv $2 $2.bak
fi

cp $1 $2
```

Example Explained

- The first conditional checks whether enough arguments are supplied.
- The second conditional checks whether the target file already exists.
- If so, it is moved to a .bak file
- Then the source file is copied to the target file.

Comparison Operators

- is equal to: if ["$a" -eq "$b"]
- Is not equal to: if ["$a" -ne "$b"]
- Is greater than: if ["$a" -gt "$b"]
- Is greater than or equal to: if ["$a" -ge "$b"]
- Other integer operators: -lt, -le
- String operators: =, ==, !=, <, >, -z (string is null)

The expr Command

- Performs string comparison and integer mathematics.
- Takes a number of arguments, each representing a token from the expression to be evaluated.
- (2*5) + 7

  expr ‘(‘ ‘2’ ‘*’ ‘5’ ‘)’ ‘+’ ‘7’
- To compare two string alphabetically

  expr “hello” ‘<‘ “why”

The while Statement

while command; do

statement

done

- The while statement block is between while and done.
- Similar to the if statement, the command (or script) is executed. The statement will be executed as long as the command evaluates to true.
- The *test* (bracket) command is commonly used as with the if statement.

Example

```
while [ “$*” != “” ]
do
  echo “value is $1”
  shift
done
```

Source: http://lowfatlinux.com/linux-script-looping.html

- Prints arguments passed to the script while the input argument string is not null.
- The *shift* command moves the values stored in the command line variables to the left one position.

The until Statement

until command

do

statement

done

- Differ from the while statement in that its body is executed as long as the condition is false.

Example

```
count=1
until [ “$*” = “” ]
do
  echo “value number $count $1 ”
  shift
  count=$[ $count + 1 ]
  # count= ‘expr $count + 1’
done
```

Source: http://lowfatlinux.com/linux-script-looping.html

The for Statement

for item in list

do

statement

done

- Similar behaviour to *foreach* statement in other programming languages.
  - Iterates through each item in a list
  - Typically statement mentions $item

Example

```
for item in “$*”
do
  echo “value is $item”
done
```

Source: http://lowfatlinux.com/linux-script-looping.html

- In each iteration, the value of *item* is assigned the nth item in the list.
- The loop terminates when all items in the list are processed.

Functions

functionname () {

statement

}

- Functions in shell-scripts are very restricted
- The () indicates that this is a function, but it is always empty.
- Arguments are referred to in the body of the function as $1, $2 etc
- A function may return an integer value as the return code

A Larger Example

```
#!/bin/bash
# From: Advanced Bash-Scripting Guide
# http://tldp.org/LDP/abs/html/
# Arabic number to Roman numeral conversion
# Usage: roman number-to-convert

LIMIT=200
E_ARG_ERR=65
E_OUT_OF_RANGE=66

if [ -z "$1" ]
then
  echo "Usage: `basename $0` number-to-convert"
  exit $E_ARG_ERR
fi

num=$1
if [ "$num" -gt $LIMIT ]
then
  echo "Out of range!"
  exit $E_OUT_OF_RANGE
fi
```

A Larger Example (cont'd)

```
to_roman ()   # Must declare function before
              # first call to it.
{
number=$1
factor=$2
rchar=$3
let "remainder = number - factor"
while [ "$remainder" -ge 0 ]
do
  echo -n $rchar
  let "number -= factor"
  let "remainder = number - factor"
done

return $number
}
```

A Larger Example (cont'd)

```
to_roman $num 100 C
num=$?
to_roman $num 90 LXXXX
num=$?
to_roman $num 50 L
num=$?
to_roman $num 40 XL
num=$?
to_roman $num 10 X
num=$?
to_roman $num 9 IX
num=$?
to_roman $num 5 V
num=$?
to_roman $num 4 IV
num=$?
to_roman $num 1 I
echo
exit
```

Further Reading

Arnold Robbins, “Classic Shell Scripting: Hidden Commands that Unlock the Power of Unix”, O’Reilly, 2005.

My web page on programming languages:
http://www.macs.hw.ac.uk/~hwloidl/prg-lang.htm

Mendel Cooper “Advanced Bash Scripting Guide”,
http://tldp.org/LDP/abs/html/index.html

“Bash Reference Manual”,
http://www.gnu.org/software/bash/manual/bashre

On-line Unix and Shell tutorials
http://www.cyberciti.biz/tips/linux-unix-commands-chea

Exercises

- In the roman numeral example, extend the range of numbers that can be covered.
- In the line count example, count the number of lines of files that have been changed within the last n days, or files owned by yourself.
- Work through the exercises in Chapters 10-12, 23, 26 of the Advanced Bash-Scripting Guide.