Edith Cowan University
School of Science

CSI6203 Scripting Languages

Module 5

Loops

Contents

- Repetition in scripts
- Iteration
- For Loops
- C-Style for loops
- Nested loops
- while and until

Learning Objectives

After finishing this module, you should be able to:

- Understand and execute scripts that require iteration
- Write scripts that iterate through content

For loops

for loops

- for is a shell keyword used to control iteration

```
#!/bin/bash
names="Joe Jessie John Alyssa"
for name in $names; do
    echo "the person's name is $name"
done
exit 0
```

- Iteration allows code to be repeated for each item within a list of items

for loops

- In a for loop, we read each item in the list from left to right
- If the list is a string of text, the items are separated by spaces by default
- Each value in the list is assigned to the variable on the left one at a time

```
for animal in "cow dog cat"; do
…
done
```

for loops

- Sometimes, we don’t want our lists to be separated by spaces

```
for phrase in "first one" "second one" "third one"; do
```

- This works fine if our list is a literal one but what if it’s coming from somewhere else?

```
for phrase in $(cat phrases.txt); do
```

for loops

- Sometimes, we don’t want our lists to be separated by spaces

```
for phrase in "first one" "second one" "third one"; do
```

- This works fine if our list is a literal one but what if it’s coming from somewhere else?

```
for phrase in $(cat phrases.txt); do
```

IFS

- The Internal Field Separator (IFS) variable is used by the system to tell where one item in a list ends and the next one starts
- By default, this is a space so that structures such as for loops will count through each word in a list
- By setting this to something else, we can make it split each item in the list based on eg. newlines

IFS

With the IFS set to a newline, each phrase will be iterated through, one line at a time instead of one word at a time

```
IFS=$'\n'
for phrase in $(cat phrases.txt); do
```

for loops and directories

For loops are also often used to iterate through the contents of files and directories

```
#!/bin/bash
for file in /home/jane/homework/* do
    if [ -d $file ]; then
        echo "$file is a folder"
    elif [ -f $file ]; then
        echo "$file is a file"
    fi
done
exit 0
```

C-style for loops

- Bash also supports C style for loops that count for a specific number of times.
- The C style for loop sets an initial value, a guard and an increment within the loop.
- This is very similar to for loops in other programming languages such as java, C# and C++

C-style for loops

- The “i” variable keeps count of the loop repeating

```
#!/bin/bash
for((i=0; i<20; i++))
do
    echo "this has been repeated $i times"
done
exit 0
```

C-style for loops

- The initial value starts at 0

```
#!/bin/bash
for((i=0; i<20; i++))
do
    echo "this has been repeated $i times"
done
exit 0
```

C-style for loops

- The loop will continue to repeat while i is less than 20

```
#!/bin/bash
for((i=0; i<20; i++))
do
    echo "this has been repeated $i times"
done
exit 0
```

C-style for loops

- Each time the loop repeats, i will go up by one

```
#!/bin/bash
for((i=0; i<20; i++))
do
    echo "this has been repeated $i times"
done
exit 0
```

Loop Structure

Nested loops

- Loops can be placed inside each other.
- The entire inner loop will be repeated by the outer loop

```
#!/bin/bash
for((i=0; i<3; i++))
do
    echo "outer loop $i"
    for((j=0; j<3; j++))
    do
        echo " inner loop $j"
    done
done
exit 0
```

Nested loops

- Loops can be placed inside each other.
- The entire inner loop will be repeated by the outer loop

```
outer loop 0
  inner loop 0
  inner loop 1
  inner loop 2
outer loop 1
  inner loop 0
  inner loop 1
  inner loop 2
outer loop 2
  inner loop 0
  inner loop 1
  inner loop 2
```

Extra loop controls

- The loop controls `break` and `continue` can be use to change the behaviour of loops
- These are primarily useful for error handling or to skip unwanted items
- The `break` statement exits a loop early
- The `continue` statement skips to the next iteration

break

```
#!/bin/bash
for file in *; do
    if [ -d "$file" ]; then
        echo "There is a directory here"
        break
    fi
done
exit 0
```

continue

```
#!/bin/bash
for file in *; do
    [ -r "$file" ] || continue
    cat "$file"
done
exit 0
```

While/Until loops

While loops

- For loops are useful when we know exactly how many times we want commands to repeat
- Either we are repeating for each item in a list
- Or we are repeating a specific number of times

While loops

- In many cases, we would rather keep looping until a certain condition is met
- Repeat while the user has not chosen to exit
- Repeat until a correct value is entered
- Repeat while there is still additional information being written

While loops

```
#!/bin/bash
read -p "please type a number greater than 5 " number
while(( $number < 5 )); do
    echo "that number is not greater than 5!"
    read -p "please type a number greater than 5" number
done
echo "thank you!"
exit 0
```

Until loops

- While loops can also be written as “until” loops
- Functionally, the operate the same but with the condition reversed

While loops

```
#!/bin/bash
read -p "please type a number greater than 5 " number
until(( $number >= 5 )); do
    echo "that number is not greater than 5!"
    read -p "please type a number greater than 5" number
done
echo "thank you!"
exit 0
```

Infinite Loops

- Beware of infinite loops!
- There is nothing in bash that stops you from creating loops that cannot finish.

- These can be created by using a guard that:
  - Has a boolean expression that can never be false
  - Has a boolean expression that can be false but doesn't reach that case
  - Has an error that causes the loop to not execute the statements within

Infinite Loops

- Has a boolean expression that can never be false

```
#!/bin/bash
until(( 2 >= 5 )); do
    echo "uh oh!"
done
exit 0
```

Infinite Loops

- Has a boolean expression that can never be false

uh oh!

uh oh!

uh oh!

uh oh!

uh oh!

uh oh!……………………………………

Infinite Loops

- Has a boolean expression that can be false but doesn’t reach that case

```
#!/bin/bash
x=1
echo "I'm counting in twos!"
until(( x == 10 )); do
    echo $x
    x=$(($x+2))
done
exit 0
```

Infinite Loops

- Has a boolean expression that can be false but doesn't reach that case

```
I'm counting in 2s!
1
3
5
7
9
11
13
15...........................
```

Infinite Loops

- Has a boolean expression that can be false but doesn’t reach that case

```
#!/bin/bash
read -p "please type a number between 1 and 10" number
until(( $number >= 5 )); do
    echo "that number is not greater than 5!"
done
echo "thank you!"
exit 0
```

Infinite Loops

- Has a boolean expression that can be false but doesn’t reach that case

```
#!/bin/bash
read -p "please type a number between 1 and 10" number
until(( $number >= 5 )); do
    echo "that number is not greater than 5!“
    read -p "please type a number between 1 and 10" number
done
echo "thank you!"
exit 0
```

Summary

- Terms to review and know include:
  - Iteration
  - For Loops
  - Lists
  - IFS
  - C-Style for loops
  - Nested loops
  - while
  - until

References and Further Reading

- Ebrahim, M. and Mallet, A. (2018) Mastering Linux Based Scripting (2nd Ed) Chapter 6, pp 102-120
- http://tldp.org/LDP/abs/html/internalvariables.html
- http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-7.html
- http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_02.html
- http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_03.html
- http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_05.html
-
-