CS Essentials

Session 3: Bash Essentials 2

Oxford University CompSoc

Loops

Loops

Execute certain sequences of commands until a condition is met

While Loops

While Loops

Structure:

```
while [ condition ]
do
    commands
done
```

While Loops: Example

While Loops: Example

Execute commands a specific number of times:

```
var=3
while [ $var -le 9 ]
do
    echo $var
    (( var ++ ))
done
```

While Loops: Example

Execute commands a specific number of times:

```
var=3
while [ $var -le 9 ]
do
    echo $var
    (( var ++ ))
done
```

NOTE: *-le means less than or equal to*

Until Loops

Until Loops

Structure:

```
until [ condition ]
do
    commands
done
```

Until Loops

Structure:

```
until [ condition ]
do
    commands
done
```

NOTE: *any while loop can be replaces by an until loop by negating the condition*

Until Loops: Example

Until Loops: Example

```
var=3
until [ $var -gt 9 ]
do
    echo $var
    (( var ++ ))
done
```

Until Loops: Example

```
var=3
until [ $var -gt 9 ]
do
    echo $var
    (( var ++ ))
done
```

NOTE: *-gt means less than or equal to*

For Loops

For Loops

For loops are different, as we execute the sequence of commands for every element in a list.
Structure:

```
for var in list
do
    commands
done
```

For Loops: Example

For Loops: Example

```
friends='Amy Tom Lisa Matt'
for person in $friends
do
    echo $person
done
```

For Loops: Example

```
friends='Amy Tom Lisa Matt'
for person in $friends
do
    echo $person
done
```

NOTE: *Variables set with singles quotes are considered list of strings.*

Aside: Ranges

Aside: Ranges

A list can be represented as a range.

Aside: Ranges

A list can be represented as a range.

{1..100} will include every number from 1 to 100.

Aside: Ranges

A list can be represented as a range.

$\{1..100\}$ will include every number from 1 to 100.

We could also define a step: $\{1..100..2\}$ will include all odd numbers from 1 to 100.

Ranges: Example

Ranges: Example

```
for number in {1..100..2}
do
    echo $number
done
```

Equivalent definitions

Equivalent definitions

Exercise: Transform the previous *while* loop into a *for* loop.

For Loops: Continued

For Loops: Continued

break and *continue* are useful:

For Loops: Continued

break and *continue* are useful:
break terminates the iterations over the loop;

For Loops: Continued

break and *continue* are useful:
break terminates the iterations over the loop;

```
numbers='4 6 22 10 3 8'
for number in $numbers
do
    if [ (( $number % 2 -ne 0 )) ]
    then
        echo Found an odd number, $number!
        break
    fi
done
```

For Loops: Continued

For Loops: Continued

break and *continue* are useful:

For Loops: Continued

break and *continue* are useful:
continue jumps over an iteration of the loop.

For Loops: Continued

break and *continue* are useful:
continue jumps over an iteration of the loop.

```
numbers='4 6 22 10 3 8'
for number in $numbers
do
    if [ (( $number % 2 -ne 0 )) ]
    then
        continue
    fi
    echo Found an odd number, $number!
done
```

For Loops: Example

For Loops: Example

```
files=$(ls)
for file in $files
do
```

For Loops: Example

```
files=$(ls)
for file in $files
do
    if [ $file = test ]
    then
        echo There exists a file called "test"
```

For Loops: Example

```
files=$(ls)
for file in $files
do
    if [ $file = test ]
    then
        echo There exists a file called "test"
        if [ -d $file ]
        then
            echo and it is a directory.
        else
            echo and it is not a directory.
        fi
        break
    fi
done
```

Practice

Write a script to delete every file in a directory.

Practice

Write a script to delete every file in a directory.

NOTE: *there is also a select loop; look it up if you are interested*

Functions

Functions

What is a function?

Functions

What is a function?
Naming sequences of commands.

Functions

What is a function?
Naming sequences of commands.
Structure:

```
function name_of_the_function {
    some commands
    ...
}
```

Functions

Functions

Why use them?

Functions

Why use them?

organize code nicely

Functions

Why use them?

organize code nicely

easier to maintain

Functions: Example

Functions: Example

```
function aboutMe {
    echo $USER
    echo $PATH
}

aboutMe
```

Functions: Arguments

Functions: Arguments

$0 is the name of the script/function

Functions: Arguments

$0 is the name of the script/function
$1.. work in the same way as for the scripts

Functions: Arguments

$0 is the name of the script/function
$1.. work in the same way as for the scripts

```
function printDate {
     echo "Today's date is $1"
}

printDate $(date +%Y-%m-%d)
```

Functions

Functions

Final exercise:

Functions

Final exercise:
Write a backup script.

Thank you!