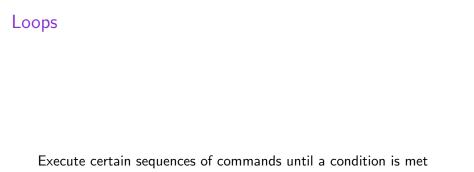
CS Essentials Session 3: Bash Essentials 2



Loops



While Loops

While Loops

Structure:

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

While Loops: Example

While Loops: Example

Execute commands a specific number of times:

While Loops: Example

Execute commands a specific number of times:

NOTE: -le means less than or equal to

Until Loops

Until Loops

Structure:

Until Loops

Structure:

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

Until Loops: Example

Until Loops: Example

Until Loops: Example

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.

A list can be represented as a range.

A list can be represented as a range.

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

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.

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

Equivalent definitions

Equivalent definitions

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

break and continue are useful:

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

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
```

break and continue are useful:

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

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
```

Ranges: Example 2

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

Ranges: Example 2

```
\label{files} \begin{array}{l} \mbox{files} = \mbox{\$(ls)} \\ \mbox{for file in \$files} \\ \mbox{do} \\ \mbox{if [\$file = test]} \\ \mbox{then} \\ \mbox{echo There exists a file called "test"} \end{array}
```

Ranges: Example 2

```
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

What is a function?

What is a function?

Naming sequences of commands.

What is a function?

```
Naming sequences of commands. Structure:
```

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

Why use them?

Why use them?

organize code nicely

Why use them?

organize code nicely easier to maintain

Functions: Example

Structure:

Functions: Example

Structure:

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

\$0 is the name of the script/function

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

```
$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'
```

Final exercise:

Final exercise:

Write a backup script.

Thank you!