*CS1101: Lecture 10*

# Shell Scripts – Control Structure

Dr. Barry O'Sullivan
b.osullivan@cs.ucc.ie


Course Homepage
http://www.cs.ucc.ie/~osullb/cs1101



## Lecture Outline

- Lecture Outline
- Control Structures
  - The `if` Statement
  - Example: `if` Statement
  - The `test` Command
  - Using `elif` and `else`
  - The `case` Statement
  - Example: `case` Statement
  - `for` Loops
  - `while` Loops
  - `until` Loops
  - `until` Loops
- Taken from: Anderson – Just Enough UNIX





## Control Structures

- Normally, the shell processes the commands in a script sequentially, one after another in the order they are written in the file.
- Often, however, you will want to change the way that commands are processed.
- You may want to choose to run one command or another, depending on the circumstances; or you may want to run a command more than once.
- To alter the normal sequential execution of commands, the shell offers a variety of control structures.





## Control Structures

- There are two types of **selection structures**, which allow a choice between alternative commands:
  - if/then/elif/else/fi
  - case
- There are three types of **repetition or iteration structures** for carrying out commands more than once:
  - for
  - while
  - until

## The `if` Statement

- The `if` statement lets you choose whether to run a particular command (or group of commands), depending on some condition.
- The simplest version of this structure has the general form

```
if conditional expression
then
    command(s)
fi
```

- When the shell encounters a structure such as this, it first checks to see whether the conditional expression is true.
- If so, the shell runs any commands that it finds between the `then` and the `fi` (which is just `if` spelled backwards).



- If the conditional expression is not true, the shell skips the commands between `then` and `fi`.





## Example: `if` Statement

- Here is an example of a shell script that uses a simple `if` statement:

```
#!/bin/sh
set ‘date‘
if test $1 = Fri
then
    echo "Thank goodness it’s Friday!"
fi
```





## The `test` Command

- Here we have used the `test` command in our conditional expression.
- The expression

```
test $1 = Fri
```

  checks to see if the parameter $1 contains `Fri`; if it does, the `test` command reports that the condition is true, and the message is printed.

- The `test` command can carry out a variety of tests; refer to some documentation for details.

## Using `elif` and `else`

- We can make the selection structures much more elaborate by combining the `if` statement with the `elif` ("else if") and `else` statements.
- Here is a simple example:

```
#!/bin/sh
set ‘date‘
if test $1 = Fri
then
    echo "Thank goodness it’s Friday!"
elif test $1 = Sat 11 test $1 = Sun
then
    echo "You should not be here working
    echo "Log off and go home."
else
    echo "It is not yet the weekend."
    echo "Get to work!"
fi
```



## The `case` Statement

- The shell provides another selection structure that may run faster than the `if` statement on some UNIX systems.
- This is the `case` statement, and it has the following general form:

```
case word in
    patternl) command(s) ;;
    pattern2) command(s) ;;
    ...
    patternN) command(s) ;;
esac
```

- The `case` statement compares `word` with `pattern1`; if they match, the shell runs the command(s) on the first line.
- Otherwise, the shell checks the remaining patterns, one by one, until it finds one that





matches the word; it then runs the command(s) on that line.

- * is the *default* case





## Example: `case` Statement

- Here is a simple shell script that uses the case statement:

```
#!/bin/sh
set ‘date‘

case $1 in

Fri) echo "Thank goodness it’s Friday!";

Sat | Sun) echo "You should not be here
           echo "Log off and go home!";;

*) echo "It is not yet the weekend.";
   echo "Get to work!";;

esac
```

## for Loops

- Sometimes we want to run a command (or group of commands) over and over.
- This is called iteration, repetition, or looping.
- The most commonly used shell repetition structure is the `for` loop, which has the general form:

```
for variable in list
do
    command(s)
done
```

- Here is a simple example:

```
#!/bin/sh
for host in $*
do
    ping $host
done
```



## while Loops

- The general form of the `while` loop is:

```
while condition
do
    command(s)
done
```

- As long as the condition is true, the commands between the `do` and the `done` are executed.
- Example

```
#!/bin/sh
# Print a message ten times
count=10
while test $count -gt 0
do
    echo $*
    count=‘expr $count - 1‘
done
```





## until Loops

- Another kind of iteration structure is the `until` loop.
- It has the general form:

```
until condition
do
    command(s)
done
```

- This loop continues to execute the command(s) between the `do` and `done` until the `condition` is true.





## until Loops

- We can rewrite the previous script using an `until` loop instead of the `while` loop:

```
#!/bin/sh
# Print a message ten times
count=10
until test $count -eq 0
do
    echo $*
    count=‘expr $count - 1‘
done
```