*CS1101: Lecture 9*

# The Shell as a Programming Language

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


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



## Lecture Outline

- Counting Arguments
- Using `read`
- The `set` Command
- Arithmetic using `expr`
- Control Structures
  - The `if` Statement
  - Example: `if` Statement
  - The `test` Command
  - Using `elif` and `else`
- Taken from: Anderson – Just Enough UNIX





## Counting Arguments

- The parameter $# contains the number of arguments that the user typed.
- We can modify the script `echo.args` once again to use this parameter:

```
#!/bin/sh
# Illustrate the use of positional param
echo You typed $# arguments: $*
```

- Suppose we were then to type the command line

```
$ echo.args To be or not to be
```

- The computer would respond with

```
You typed 6 arguments: To be or not to b
```





## Using `read`

- The positional parameters are useful for capturing command-line arguments but they have a limitation: once the script begins running, the positional parameters cannot be used for obtaining more input from the standard input.
- For this you have to use the `read` statement.
- Let's modify the previous program to make use of `read`:

```
#!/bin/sh
# Illustrate the use of positional param
# variables and the read command.
echo 'What is your name?'
read name
echo "Well, $name, you typed $# argument
echo "$*"
```

- In this script, `name` is a user-defined variable.



## The `set` Command

- The positional parameters are sometimes called *read-only variables*, because the shell gets their values for you when you type arguments to the script.
- However, you can also set their values using the set command.
- To illustrate this, consider the following shell script, which we will assume is in the file `setdate`:

```
#!/bin/sh
# Demonstrate the set command
set ‘date‘
echo "Time: $4 $5"
echo "Day: $1"
echo "Date: $3 $2 $6"
```





## The `set` Command

- Assuming that `setdate` has been made executable with the `chmod` command, we can run the script by typing `setdate` as a command name
- The output will look something like this:

```
Time: 10:56:08 EST
Day: Fri
Date: 10 Aug 2001
```

- What happened?





## The `set` Command

- The backquotes in the `set ‘date‘` command run the UNIX `date` command, which produces output something like this:

```
Fri Aug 10 10:56:08 EST 2001
```

- This does not appear on the screen.
- Instead, the `set` command catches the output in the positional parameters $1 through $6:
  - $1 contains Fri
  - $2 contains Aug
  - $3 contains 10
  - $4 contains 10:56:08
  - $5 contains EST
  - $6 contains 2001

## Arithmetic using expr

- The shell is not intended for numerical work – if you have to do a lot of calculations, you should consider C, FORTRAN, or Java.
- Nevertheless, the `expr` utility may be used to perform simple arithmetic operations on integers
- `expr` is not a shell command, but rather a separate UNIX utility; however, it is most often used in shell scripts.
- To use it in a shell script, you simply surround the expression with backquotes.



## Arithmetic using expr

- For example, here is a simple script called `add` that adds two numbers typed as arguments:

```
#!/bin/sh
# Add two numbers
sum=‘expr $1 + $2‘
echo $sum
```

- Example of its use:

```
$ add 4 3
7
$
```

- The expr command only works on integers (i.e., whole numbers).
- It can perform addition (+), subtraction (-), multiplication (*), integer division (/) and integer remainder (%).





## 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
```