COMP 2718: Shell Scripts: Part 4

By: Dr. Andrew Vardy

Outline

- More Tricks with Arguments
- `shift` - Shifting In Arguments
- Passing Arguments Along

More Tricks with Arguments

We have seen how scripts can use the parameters $1, $2 and so on to access command-line arguments. For example:

```
----------------------------------------
echo "First 4 args:" $1 $2 $3 $4
echo "\$0:" $0
echo "Number of args:" $#
----------------------------------------
```

This script introduces two new special variables. $0 is set to the pathname of the script itself. $# is the number of arguments.

```
$ echo_args.sh alpha beta gamma delta epsilon
First 4 args: alpha beta gamma delta
$0: ./echo_args.sh
Number of args: 5
```

`shift` - Shifting In Arguments

If many arguments are passed in, it is sometimes useful to step through them one-by-one. This can be done by using `shift` which moves the arguments "downwards". For example, if a script is called as follows:

```
$ script_name alpha beta gamma 12
```

Initially the parameters are as follows:

```
$1: alpha   $2: beta   $3: gamma   $4: 12
$#: 4
```

After `shift` is called we have the following:

```
$1: beta    $2: gamma  $3: 12      $4:
$#: 3
```

```
--------------------------------------------------
echo "First 4 args:" $1 $2 $3 $4
echo "Number of args:" $#
shift
echo -e "\nFirst 4 args:" $1 $2 $3 $4
echo "Number of args:" $#
--------------------------------------------------
```

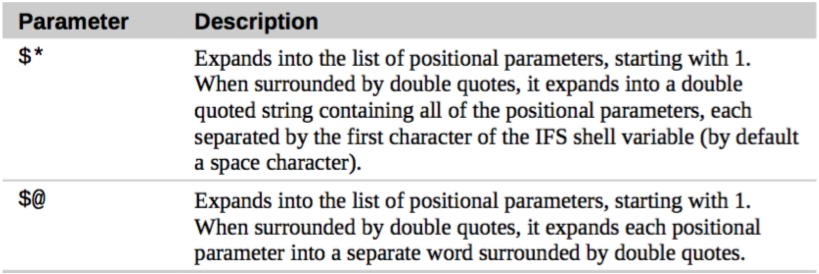
```
$ ./shift.sh alpha beta gamma delta epsilon
First 4 args: alpha beta gamma delta
Number of args: 5

First 4 args: beta gamma delta epsilon
Number of args: 4
```

`shift` can be useful for processing command-line options.

```
# Process options for the fictional command:
#   fic-options.sh [-r] [-i] [-w word]
while (($# > 0)); do
    option=$1
    if [ "$option" == "-r" ]; then
        echo "-r option"
    fi
    if [ "$option" == "-i" ]; then
        echo "-i option"
    fi
    if [ "$option" == "-w" ]; then
        # Requires an additional word
        shift
        echo "-w option with word =" $1
    fi
    shift
done
```

Passing Arguments Along

Sometimes it is useful to pass along the whole list of arguments to another program, or perhaps a function. The special parameter `$@` can be used for this. There is another special parameter `$!` that achieves a similar result:



Parameter	Description
$*	Expands into the list of positional parameters, starting with 1. When surrounded by double quotes, it expands into a double quoted string containing all of the positional parameters, each separated by the first character of the IFS shell variable (by default a space character).
$@	Expands into the list of positional parameters, starting with 1. When surrounded by double quotes, it expands each positional parameter into a separate word surrounded by double quotes.

In general, you are advised to use `"$@"` as it preserves individual arguments with embedded spaces.

The following example script provides a wrapper around any operation (in this case `ping`). It creates and adds a message to a logfile every time it is executed. Meanwhile, it passes along all command-line arguments to `ping`.

```
-----------------------------------------------------------
# Generic shell wrapper that performs an operation
# and creates a log file describing it.

# Set the following two variables.
OPERATION=ping
LOGFILE=logfile.txt

# Log it.
echo "$(date) + $OPERATION "$@"" >> $LOGFILE
# Now, do it.
exec $OPERATION "$@"
-----------------------------------------------------------
```