

Shell Script Constructs

The *sh-bang statement* specifies which shell will run the script. This should be the first statement in the script

```
#!/bin/bash      # a bash script
```

```
#!/bin/tcsh     # a tcsh script
```

```
#!/bin/zsh      # a zsh script
```

Comments are encouraged. The *pound sign* (#) denotes a comment or remark. Comments are not processed by the script. They exist for your interest and edification.

Positional arguments are the parameters (or arguments) that are passed to the script from the command line.

Example of a script named *viewVars*

```
#!/bin/bash
# script to display arguments on the command line
echo $1
echo $2
```

Here's the call:

```
$ viewVars 24 "sam"    # note that sam is a literal string
24
sam
$
```

Q: How does the shell know where one argument ends and the next one begins?

A: Because of an environment variable called the IFS (Internal Field Separator). The IFS defaults to a whitespace character...change it at your own peril...

Special Shell Variables

Variable/Parameter	Contents
\$0	The command name
\$n	Value of the n_{th} command line arg
\$*	All command line arguments
\$@	All command line arguments
\$#	Count of all command line args
\$\$	PID of current process
\$_	PID of most recent background task
\$?	Exit status of last task executed

Selection Statements

Simple Selection

```
If [ some condition is true ]  
then  
    do the true thing  
    do the true thing ...  
fi
```

Example

```
#!/bin/bash
# display two args given on command line
# display usage message if incorrect number of vars
```

```
If [ $# -eq 0 -o $# -lt 2 ] # if number of args is 0
Then                          # or number of args < 2
    echo "Usage: dispVars var1 var2" 1>&2
    exit 1
fi
```

```
If [ $# -eq 2 ]
then
    echo "The value of var1 is $1"
    echo "The value of var2 is $2"
    exit 0
fi
```

Selection with an Option

```
If [ some condition is true ]
then
    do the true thing
    do the true thing ...
else
    do false thing
fi
```

Example script

```
If [ $# -ne 2 ]
then
    echo "Usage: dispVars var1 var2" 1>&2
    exit 1
else
    echo "The value of var1 is $1"
    echo "The value of var2 is $2"
    exit 0
fi
```

example runs...

```
$ dispVars 3 # only one arg
Usage: dispVars var1 var2
$
```

```
$ dispVars 3 9
The value of var1 is 3
The value of var2 is 9
$
```

Extended if statement

Syntax:

```
If [ true ]  
then  
    do this  
elif [ true ]  
then  
    do this  
elif [ true ]  
then  
    do this  
elif [ true ]  
    do this  
else  
    do this  
fi
```

Notes:

If the first condition fails, the second condition is tried.
If the second condition fails, the third condition is tried.
So on and so on
If all conditions fail, the commands following the trailing
else are executed

If, on the other hand, when the first true condition is
met, the commands for that condition will be executed
and then control goes to the next programming
statement after the fi

Example

```
#!/bin/bash
# script to determine a letter grade from a test score

if [ $# -ne 1 ]
then
    echo "Usage: grade score" 1>&2
    exit 1
fi

if [ $score -ge 90 ]
then
    echo " Grade: A"
elif [ $score -ge 80 ]
then
    echo " Grade: B"
elif [ $score -ge 70 ]
then
    echo " Grade: C"
elif [ $score -ge 60 ]
then
    echo " Grade: D"
else
    echo " Grade: F"
fi

echo "Processing Completed"
exit 0
```

for ... in loops

Syntax:

Type 1:

```
for loop-index in argument-list  
do  
    commands
```

done

Notes:

The structure assigns the first value in the argument list to the and executes commands between do and done.

Example:

```
$ cat myForIn  
for fruit in apples oranges pears  
do  
    echo “ $fruit”  
done  
echo “task complete”
```

run it...

```
$ myForIn  
apples  
oranges  
pears  
task complete
```

For Loop (type 2)

Syntax:

```
for loop-index  
do  
    commands  
  
done
```

Notes:

The loop-index automatically takes on the value of each of the command line arguments, one at a time.

Example:

```
$ cat myFor  
for arg  
do  
    echo “ $arg”  
done
```

run it...

```
$ myFor red green blue  
red  
green  
blue
```