# CSCI 274 - Intro to Linux OS

## Week 7 - Exit status, Command Line Arguments, and Conditional Controls

Max Gawason (maxgawason@mines.edu) [A]
Rocco Marchitto (rmarchitto@mines.edu) [B/C/D]

# Overview

1. Exit Status
2. Command Line Arguments
3. Conditional Commands
    a. if-then-fi
    b. test
    c. false/true

# Exit Status

On Linux systems, programs can pass a value to their parent process while terminating. This value is referred to as an exit code or exit status. So, any script or command executed will have an exit code. The exit status value varies from 0 to 255.

You can use command exit status in the shell script to display an error message or take some sort of action based on the exit code.

**Exit Status**

| | |
|---|---|
| **0** | **Successful execution of command** |
| 1 | command fails because of an error during expansion or redirection, the exit status is greater than zero. |
| 2 | Incorrect command usage |
| 126 | Command found but not executable |
| 127 | command not found |

# Exit codes in command line

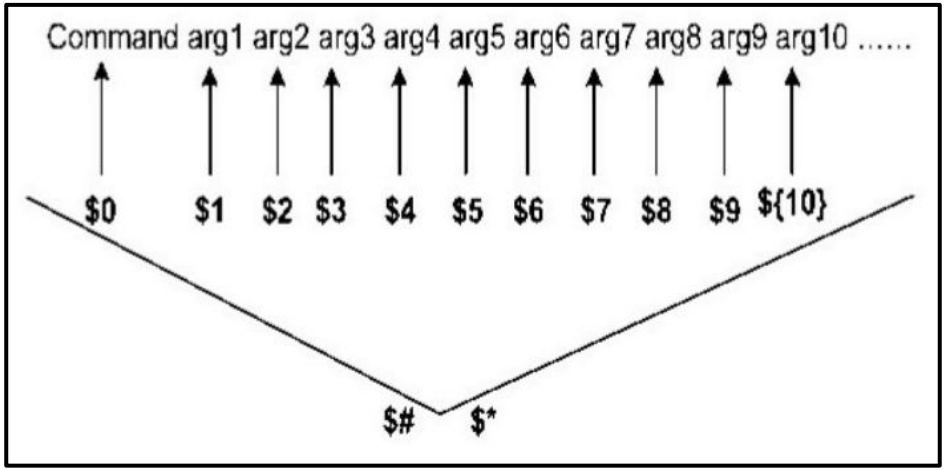
```
echo -e "Successful execution"
echo -e "====================="
echo "hello world"
# Exit status returns 0, because the above command is a success.
echo "Exit status" $?

echo -e "Incorrect usage"
echo -e "====================="
ls --option
# Incorrect usage, so exit status will be 2.
echo "Exit status" $?
```

You can use **$?** to find out the exit status of a Linux command. Execute **echo $?** command to check the status of executed command.

# Command Line Arguments

Command line arguments (also known as positional parameters) are the arguments specified at the command prompt with a command or script to be executed. The locations at the command prompt of the arguments as well as the location of the command, or the script itself, are stored in corresponding variables. These variables are special shell variables.



# Command Line Arguments ...

| Variable | Description |
|---|---|
| $0 | The filename of the current script. |
| $n | These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on). |
| $# | The number of arguments supplied to a script. |
| $* | All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2. |
| $@ | All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2. |
| $$ | The process number of the current shell. For shell scripts, this is the process ID under which they are executing. |
| $! | The process number of the last background command. |

# if-then-fi

If statements (and, closely related, case statements) allow us to make decisions in our Bash scripts. They allow us to decide whether or not to run a piece of code based upon conditions that we may set. There are total **5** conditional statements which can be used in bash programming:

- if statement
- if-else statement
- if..elif..else..fi statement (Else If ladder)
- if..then..else..if..then..fi..fi..(Nested if)
- switch statement

# Continue ...

- if statement

```
if [ expression ]
then
   statement
fi
```

- if-else statement

```
if [ expression ]
then
   statement1
else
   statement2
fi
```

# Continue ...

- if..elif..else..fi statement

```
if [ expression1 ]
then
   statement1
   statement2
   .
elif [ expression2 ]
then
   statement3
   statement4
   .
else
   statement5
fi
```

# Continue ...

- if..then..else..if..then..fi..fi..

```
if [ expression1 ]
then
   statement1
   statement2
   .
else
   if [ expression2 ]
   then
      statement3
      .
   fi
fi
```

# Continue ...

- switch statement

```
case  in

   Pattern 1) Statement 1;;

   Pattern n) Statement n;;

esac
```

```
CARS="bmw"

#Pass the variable in string
case "$CARS" in
    #case 1
    "mercedes") echo "Headquarters - Affalterbach, Germany" ;;

    #case 2
    "audi") echo "Headquarters - Ingolstadt, Germany" ;;

    #case 3
    "bmw") echo "Headquarters - Chennai, Tamil Nadu, India" ;;
esac
```

# Test

The square brackets ( [ ] ) in the if statement above are actually a reference to the command **test**. This means that all of the operators that **test** allows may be used here as well. Look up the man page for **test** to see all of the possible operators (there are quite a few) but some of the more common ones are:

| Operator | Description |
|---|---|
| **! EXPRESSION** | The EXPRESSION is false. |
| **-n STRING** | The length of STRING is greater than zero. |
| **-z STRING** | The lengh of STRING is zero (ie it is empty). |
| **STRING1 = STRING2** | STRING1 is equal to STRING2 |
| **STRING1 != STRING2** | STRING1 is not equal to STRING2 |
| **INTEGER1 -eq INTEGER2** | INTEGER1 is numerically equal to INTEGER2 |
| **INTEGER1 -gt INTEGER2** | INTEGER1 is numerically greater than INTEGER2 |
| **INTEGER1 -lt INTEGER2** | INTEGER1 is numerically less than INTEGER2 |
| **-d FILE** | FILE exists and is a directory. |
| **-e FILE** | FILE exists. |
| **-r FILE** | FILE exists and the read permission is granted. |
| **-s FILE** | FILE exists and it's size is greater than zero (ie. it is not empty). |
| **-w FILE** | FILE exists and the write permission is granted. |
| **-x FILE** | FILE exists and the execute permission is granted. |

# Boolean Operations

Sometimes we only want to do something if multiple conditions are met. Other times we would like to perform the action if one of several condition is met. We can accommodate these with boolean operators.

| Variable | Description |
| --- | --- |
| && | and |
| \|\| | or |

# True

The **true** command's sole purpose is to return a successful exit status. It is useful when you want part of a command or conditional expression always to be true. When provided no arguments, true returns successfully. When provided any number of arguments, true returns successfully.

The **true** command always returns 0, representing "true" or "success".

# False

**false** command is used to return an exit status code (“1” by default) that indicates failure. It is useful when the user wants a conditional expression or an argument to always be unsuccessful. When no argument is passed to the false command, it fails with no output and exit status as 1.

We can use the **false** command in if statement when we want to execute a statement/command if the condition becomes false.