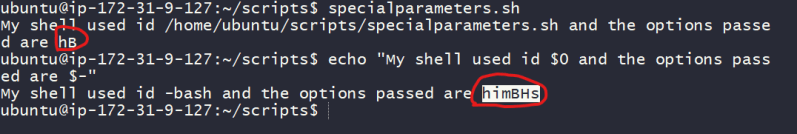
**Special parameters**

* $0 $1 …
* $# : count of number of arguments passed
* $-:

Create a simple script and execute

* #!/bin/bash
* echo "My shell used id $0 and the options passed are $-"
  + Directly execute the following statement in terminal
* echo "My shell used id $0 and the options passed are $-"
  + The following results will be shown 
  + The options set are as follows
    - h: This is short for hash all
    - i: Shows its and interactive
    - m: This is short for monitor
    - B: This allows brace expansion. “`mkdir dir{1,2}
    - H: This allows history expansion of running commands

**Setting default values to parameters**

* Its often a good practice to assign default values for non-critical parameters. See the below example and execute this

#!/bin/bash

#####################################################################

# Author: Shaik Khaja Ibrahim

# Version: v1.0.0

# Date: 02-Sep-2020

# Description: This script demonstrates default values for

# positional parameters

# Usage: ./defaultparamvalues.sh <name> <course>

#####################################################################

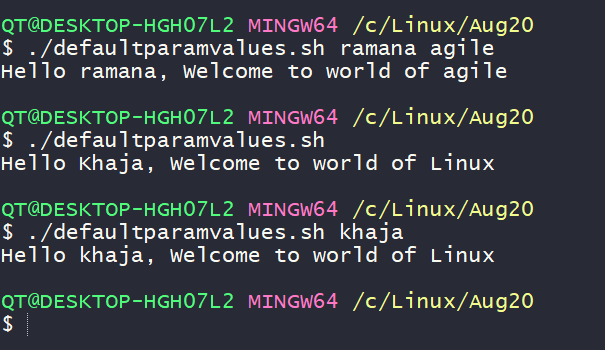
name=$1

course=$2

[ -z $name ] && name="Khaja"

[ -z $course ] && course="Linux"

echo "Hello ${name}, Welcome to world of ${course} "



* There is an alternative syntax to assign default value to parameter

${parameter-default}

* lets apply this syntax to shell script

#!/bin/bash

#####################################################################

# Author: Shaik Khaja Ibrahim

# Version: v1.0.0

# Date: 02-Sep-2020

# Description: This script demonstrates default values for

# positional parameters

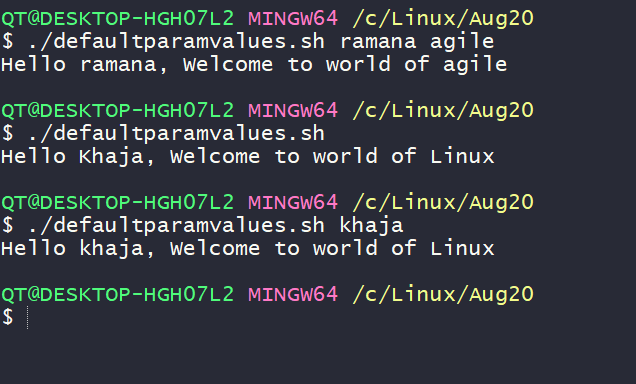
# Usage: ./defaultparamvalues.sh <name> <course>

#####################################################################

name=${1-"Khaja"}

course=${2-"Linux"}

echo "Hello ${name}, Welcome to world of ${course} "



* If the parameter is declared and it has a null value

${parameter:-default}

**Looping constructs in bash scripting**

* Lets try to understand
  + *for* loops
  + The internal field separator
  + Counting directories & files
  + Nested Loops
  + Redirecting loop output
  + *while* and *until* loops
* Sample for script

#!/bin/bash

# printing multiple courses

echo "DevOps"

echo "AWS"

echo "Azure"

echo "Linux"

echo "Windows"

echo "Now using for"

# with for loop

for course in DevOps AWS Azure Linux Windows ; do

echo "${course}"

done

# other kind of for loop

echo "Now using for which is c-styled"

courses=(DevOps AWS Azure Linux Windows)

for (( index=0; index<5; index++ )) do

echo "${courses[$index]}"

done

**Internal Field Separator**

* By default the IFS value has one of (space, newline or tab)
* Lets assume you want o iterate over

Hello,

This is Linux

I'm fun to work with

* Now if we write to script to iterate and print over this content as shown below

#!/bin/bash

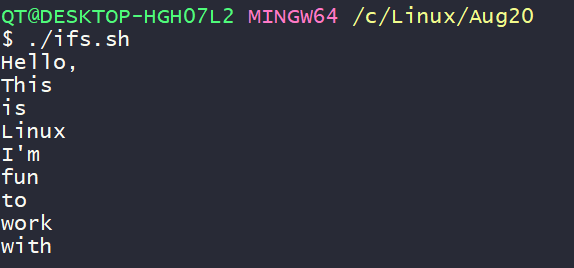
file="text.txt"

for item in $(cat $file)

do

echo "$item"

done



* What should be done to this script to iterate over lines. Now add IFS=$’\n’ to the shell script and this will do the trick

#!/bin/bash

file="text.txt"

IFS=$'\n'

for item in $(cat $file)

do

echo "$item"

done

