Lab 7
Cut command.
Positional Parameters.
Functions in Bash.

Linux Commands

cut = cuts parts of a line by byte position, character, and field.

- cut OPTIONS file

Options:

-c : cuts characters

-f : cuts fields

-d: specify the field separator

Notes:

1). If the delimiter is present, but the number of fields in the line is less than the requested field number, an empty line will be printed.
2). If there is no separator character in the line, the whole line will be printed.

Example

>cat **cutFile**

a11 a12 a13 a14 a15

b21 b22 b23 b24 b25

c31 c32 c33 c34 c35

>cut **-c1-3,5,8-10** cutFile

a11a a1

b21b b2

c31c c3

>cut **-d" " -f2,4** cutFile

a12 a14

b22 b24

c32 c34

>cat **cutFile2**

a11:a12:a13

b21:b22

c31 c32 c33

>cat cutFile2|**cut -d":" -f1**

a11

b21

c31 c32 c33

>cat cutFile2|**cut -d":" -f3**

a13

c31 c32 c33

Bash

1.Positional Parameters

When we ran some command of program by

> program_name word1 word2 word3

we can read each item on the command line because the positional parameters contain the following:

- $0 contains "program_name"
- $1 contains "word1"
- $2 contain "word2"
- $3 contains "word3"
- $# contains number of parameters (not incliding the program_name)
- $@ contains all the parameters (not incliding the program_name)

Example PosPar.sh:

```
#!/bin/bash
#WAY1
echo "Positional Parameters"
echo The program name  '$0 = ' $0
echo The first parameter  '$1 = ' $1
echo The second parameter '$2 = ' $2
echo The third parameter'$3 = ' $3
echo The total number of parameters is $#

#########################################################
#WAY2
i=1
for  PARAM in $@
do
	echo "The parameter number $i is $PARAM"
	i=$((i+1))
done

#########################################################
#WAY3

NUM_OF_PARAM=$#
for (( i=1; i<=NUM_OF_PARAM; i++ ))
do
	echo "name=$0"
	echo "The parameter number $i is $1 "
	 shift
done
```

Note: shift command shifts all the arguments to the left.

Runing:

./PosPar.sh fir sec third

2.Functions

Defining function:

```
function func_name () {
        commands
}
```

```
func_name () {
        commands
}
```

Calling function:

func_name

Local Variables:

Can be defined in the function by:

local var_name

Input Parameters

Can be passed using **positional parameters**

Return value

Can be passed using **command substitution**

Notes:

- The **let** command performs various arithmetic, bitwise and logical operations. The built-in command works only with integers.

- Useful math operators: +, -, *, /, ++, --, +=, -=.

Example FuncExample.sh:

```
#!/bin/bash
##########################################
#Call function
function hello1 () {
	echo "Hello!"
}
hello1  #Call function
##########################################
# Function with one input parameter
function hello2 () {
	echo "Hello $1"
}
hello2 Tomer  #Call function with parameter: Tomer is $1 parameter of hello2
		#Output is Hello Tomer

##########################################
# Function with number of input parameters
hello3 () {
for NAME in $@
do
	echo "Hello $NAME"
done
}

hello3 Miri Natali Yasmin
```

```
##########################################
# Function with output value
sum () {
result=0
for NUM in $@
do
      let result+=$NUM       # Also possible result=$((result+NUM))
done

echo $result
}

n=`sum 1 2 3`  # Run sum function with 3 specific input parameters
echo "n=$n"

n=`sum $@`   #Run sum function with input parameters of FuncExample.sh
echo "new n=$n"

##########################################
# Function with local parameters
surprise () {
local a=surprise_a
b=surprise_b
}
a=original_a
b=original_b
echo $a $b
surprise
echo $a $b
```