Chapter 3c. scripting parameters

3c.1. script parameters

A **bash** shell script can have parameters. The numbering you see in the script below continues if you have more parameters. You also have special parameters containing the number of parameters, a string of all of them, and also the process id, and the last return code. The man page of **bash** has a full list.

```
#!/bin/bash

echo The first argument is $1
echo The second argument is $2
echo The third argument is $3

echo \$ $$  PID of the script
echo \# $#  count arguments
echo \? $?  last return code
echo \* $*  all the arguments
```

Below is the output of the script above in action.

```
[clim@sop scripts]$ ./pars one two three
The first argument is one

The second argument is two
The third argument is three

$ 5610 PID of the script

# 3 count arguments

? 0 last return code

* one two three all the arguments
```

Once more the same script, but with only two parameters.

```
[clim@sop scripts]$ ./pars 1 2
The first argument is 1

The second argument is 2
The third argument is

$ 5612 PID of the script

# 2 count arguments

? 0 last return code

* 1 2 all the arguments
[clim@sop scripts]$
```

Here is another example, where we use **$0**. The **$0** parameter contains the name of the script.

```
clim@sop~$ cat myname echo
this script is called $0
clim@sop~$ ./myname

this script is called ./myname
clim@sop~$ mv myname test42
clim@sop~$ ./test42

this script is called ./test42
```

3c.2. shift through parameters

The **shift** statement can parse all **parameters** one by one. This is a sample script.

```
stud1000@srvUbuntu$ cat shift.ksh

#!/bin/ksh


if [ "$#" == "0" ]

 then

  echo You have to give at least one parameter.
  exit 1

fi


while (( $# ))
 do

  echo You gave me $1
  shift

 done
```

Below is some sample output of the script above.

```
stud1000@srvUbuntu$ ./shift.ksh
one You gave me one

stud1000@srvUbuntu$ ./shift.ksh one two three 1201 "33
42" You gave me one

You gave me two
You gave me three
You gave me 1201
You gave me 33 42

stud1000@srvUbuntu$ ./shift.ksh

You have to give at least one parameter.
```

3c.3. runtime input

You can ask the user for input with the **read** command in a script.

```
#!/bin/bash

echo -n Enter a number:
read number
```

3c.4. sourcing a config file

The **source** (as seen in the shell chapters) can be used to source a configuration file.

Below a sample configuration file for an application.

```
[clim@sop scripts]$ cat myApp.conf

# The config file of myApp


# Enter the path here
myAppPath=/var/myApp


# Enter the number of quines here
quines=5
```

And here an application that uses this file.

```
[clim@sop scripts]$ cat myApp.bash

#!/bin/bash

#

# Welcome to the myApp application

#
```

The running application can use the values inside the sourced configuration file.

```
[clim@sop scripts]$ ./myApp.bash
There are 5 quines

[clim@sop scripts]$
```

3c.5. get script options with getopts

The **getopts** function allows you to parse options given to a command. The following script allows for any combination of the options a, f and z.

```
stud1000@srvUbuntu$ cat options.ksh

#!/bin/ksh


while getopts ":afz" option;
do

 case $option in
  a)

   echo received -a

   ;;

  f)

   echo received -f

   ;;

  z)

   echo received -z

   ;;

  *)

   echo "invalid option -$OPTARG"

   ;;

 esac
done
```

This is sample output from the script above. First we use correct options, then we enter twice an invalid option.

```
stud1000@srvUbuntu$ ./options.ksh
stud1000@srvUbuntu$ ./options.ksh
-af received -a

received -f

stud1000@srvUbuntu$ ./options.ksh -
zfg received -z

received -f
invalid option -g

stud1000@srvUbuntu$ ./options.ksh -a -b
-z received -a

invalid option -b
received -z
```

You can also check for options that need an argument, as this example shows.

```
stud1000@srvUbuntu$ cat argoptions.ksh

#!/bin/ksh



while getopts ":af:z" option;
do

 case $option in
  a)

   echo received -a

   ;;

  f)

   echo received -f with $OPTARG

   ;;

  z)

   echo received -z

   ;;

  :)

   echo "option -$OPTARG needs an argument"

   ;;

  *)

   echo "invalid option -$OPTARG"

   ;;

 esac
done
```

This is sample output from the script above.

```
stud1000@srvUbuntu$ ./argoptions.ksh -a -f hello
-z received -a

received -f with hello
received -z

stud1000@srvUbuntu$ ./argoptions.ksh -zaf
42 received -z

received -a
received -f with 42

stud1000@srvUbuntu$ ./argoptions.ksh
-zf received -z

option -f needs an argument
```

3c.6. get shell options with shopt

You can toggle the values of variables controlling optional shell behaviour with the **shopt** built-in shell command. The example below first verifies whether the cdspell option is set;

it is not. The next shopt command sets the value, and the third shopt command verifies that the option really is set. You can now use minor spelling mistakes in the cd command. The man page of bash has a complete list of options.

```
clim@sop:~$ shopt -q cdspell ; echo $? 1

clim@sop:~$ shopt -s cdspell clim@sop:~$
shopt -q cdspell ; echo $? 0

clim@sop:~$ cd /Etc

/etc
```

3c.7. practice: parameters and options

3c.1.1. Write a script that receives four parameters, and outputs them in reverse order.

3c.2.1. Write a script that receives two parameters (two filenames) and outputs whether those files exist.

3c.3.1. Write a script that asks for a filename. Verify existence of the file, then verify that you own the file, and whether it is writable. If not, then make it writable.

3c.4.1. Make a configuration file for the previous script. Put a logging switch in the config file, logging means writing detailed output of everything the script does to a log file in /tmp.