CSNB113: System Administration

-

9th Topic: Shell Scripts – Programming II

Parameter: $# and $?

It is possible to ***pass arguments*** when ***calling*** a shell program.

Sometimes, we need - or expect – a specific number of arguments. If not, the program wouldn't work. There is a convenient way to check the number of arguments; with $#:

```
#!/bin/bash
if [ $# -ne 1 ]; then
    echo "Usage - $0  single_argument"
    exit 9
fi
```

This program introduced yet another feature: `exit`

'exit' allows us to intercept if the previous program ran successfully, if it went wrong, and where it went wrong. The number following the word 'exit' is **passed to the shell**, and can be used for another program, or just displayed.

In the example above, whenever you have a number of arguments **not equal to 1**, **9** will be passed to the calling routine.

```
$ ./exito AD
$ echo $?
0
$ ./exito
Usage - ./exito  single_argument
$ echo $?
9
$
```

List Constructors - 0

A program will **always** output an exit code, if you write it or not.

With exit codes of your choice, you can control what happens with the next command. There are many uses for system administration: when one command executes correctly, the subsequent one can be run as well. When the first command does not run, the next command does not need to run. Example: creating a user account. If it fails, there is no need to set a password for it!

```
#!/bin/bash
if [ $# -ne 1 ]; then
    echo "Usage - $0  single_argument"
    exit 9
fi

$ ./exito && date
Usage - ./exito  single_argument
$ ./exito AD && date
Fri Jan 21 16:04:55 MYT 2011
$
```

Huuh, what happened here??

List Constructors - I

The && are actually **list constructors**. You can use them to concatenate a list of commands, and at the same time do a **compound comparison**.

This means, instead of using the semicolon between commands, which is a list constructor as we saw in the last lecture, we can use a combined function that constructs the list, PLUS stops execution when it compares the exit codes:

```
#!/bin/bash
if [ $# -ne 1 ]; then
    echo "Usage - $0  single_argument"
    exit 9
fi

$ ./exito ; date
Usage - ./exito  single_argument
Fri Jan 21 16:21:05 MYT 2011
$ ./exito && date
Usage - ./exito  single_argument
$
```

Understand??

Default exit codes

There are some default exit codes that the programs produce; irrespective if the programmer has inserted them or not. The programmer may insert additional, or other, codes if (s)he wanted.

Exit Value	Exit Status
0 (Zero)	Success
Non-zero	Failure
2	Incorrect usage
127	Command Not found
126	Not an executable

```
$ abcdefgh
abcdefgh: command not found
$ echo $?
127
$ date ; echo $?
Fri Jan 21 17:36:40 MYT 2011
0
$
```

Windows, too!

Windows also creates exit codes. It is a tad more difficult to intercept them, though:

```
:begin
@echo off
%1
if not errorlevel 1 goto end
echo ERROR !
:end
echo %errorlevel%
@echo on
```

This is a very basic ***wrapper*** for any Windows program. Name it `demo.bat` and pass the program name foo as parameter:

```
C:\........\> demo.bat foo
```

and it will return an 'exit' or 'return' value. If foo is not valid, it will return 9009; e.g.

Error handling

System Administration is a lot about working with files and directories. If they don't exist, or you mistype them, the resulting action is undefined.

Scripts, that is programs, need proper **error-handling**!

Check this example of script errhand, that contains some basic error handling:

```
$ ./errhand
Usage: ./errhand filename
$ ./errhand uwe one
Usage: ./errhand filename
$ ./errhand uwe
Sorry, file uwe does not exist
$ ./errhand testo
#!/bin/bash
TEST="no"
while [ "$TEST" != "yes" ];
do
        read TEST
done
$
```

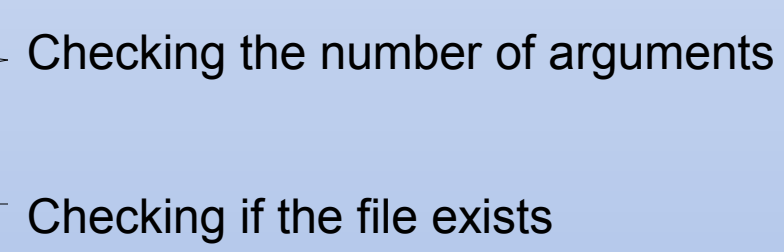
Error handling

System Administration is a lot about working with files and directories. If they don't exist, or you mistype them, the resulting action is undefined.

Scripts, that is programs, need proper **error-handling**!

Check this example of script errhand, that contains some basic error handling:

```
$ ./errhand
Usage: ./errhand filename
$ ./errhand uwe one
Usage: ./errhand filename
$ ./errhand uwe
Sorry, file uwe does not exist
$ ./errhand testo
#!/bin/bash
TEST="no"
while [ "$TEST" != "yes" ];
do
        read TEST
done
$
```



Error handling - code

Check this example of script `errhand`, that contains some basic error handling:

```
#!/bin/bash

if [ $# -ne 1 ]; then
    echo "Usage: $0 filename"
    exit 1
fi

if [ -f $1 ]; then
    cat $1
else
    echo "Sorry, file $1 does not exist"
fi
```

The first had been done some slides before. The second part tests the file: '-f' stands for 'file exists'.

There are many other possible checks for files and directories.

File and Directory checks

Operator	Tests Whether
`-e`	File exists
`-f`	File is a *regular* file
`-d`	File is a *directory*
`-s`	File is not zero size
`-r`	File has *read* permission
`-w`	File has *write* permission
`-x`	File has *execute* permission
`-O`	You own the file
`-G`	*Group id* of file same as yours

Handling files with scripts - 0

How to read file names, handle file names? How to check files?

This is the sample directory, consisting of 7 files:

```
$ ls -l
-rw-r--r-- 1 admini wheel    10240 2011-01-10 18:14 demo.tar
-rw-r--r-- 1 admini wheel        0 2011-01-24 14:31 empty_file
-rwxr-xr-x 1 admini wheel      164 2011-01-24 11:40 errhand
drwxr-xr-x 2 admini wheel     4096 2011-01-10 18:18 etc
-rwxr-xr-x 1 admini wheel      215 2011-01-24 14:29 file_handler
-rw-r--r-- 1 admini wheel       77 2011-01-17 16:21 testo
-rw-r--r-- 1 admini wheel        0 2011-01-10 18:05 test.txt
```

There are 7 files, there is one directory, there are two files without content.

The task of the script is, to identify the files with content, the empty files, as well as any file that actually is a directory.

Therefore, we create a '`do … done`' **loop**, looping through all files (file names) that we have to obtain first.

Handling files with scripts - I

This is sample code to show how to get the list of files into our program (*globbing* anyone?):

```
#!/bin/bash
# first demo of handling file by file

for FILE in *
do
if [ -s $FILE ]; then
        echo "$FILE has content"
else
        echo "$FILE is empty"
fi
# checking if it is a directory
if [ -d $FILE ]; then
        echo "$FILE is a directory"
fi
done
$
```

Once the files (file names) are known, we can do some ***test***s on the files

Handling files with scripts - II

Here is the output of the program, run in the directory and on the files as above:

```
$ ./file_handler
demo.tar has content
empty_file is empty
errhand has content
etc has content
etc is a directory
file_handler has content
testo has content
test.txt is empty
$
```

This works, but is not really nice:

- `etc` shows twice
- we don't see how many files there are in that directory altogether
- we don't know about executable files, the user cannot select what (s)he wants to see

(and more)

References

- http://www.google.com.my/
  search for 'shell script'
-