

Εισαγωγή στα Λειτουργικά Συστήματα



SET ΔΙΑΦΑΝΕΙΩΝ 13

SHELL SCRIPTS: TEST, IF

ΑΝΤΩΝΗΣ ΣΙΔΗΡΟΠΟΥΛΟΣ

Διεργασίες: Γονείς και Παιδιά

2

- Κάθε διεργασία κάνοντας `exit` επιστρέφει έναν κωδικό εξόδου.
 - Είτε με `exit(0)`, `exit(1)` etc....
 - Είτε με `return(κωδικός)` από την συνάρτηση `main()`
 - Γι' αυτό ο τύπος της `main` στη γλώσσα C είναι:
`int main(int argc, char **argv)`
- Όταν μια διεργασία κάνει `exit` (ή τερματίζεται για οποιοδήποτε λόγο) τότε πηγαίνει το αντίστοιχο `signal` στην γονική διεργασία (`SIGCHLD`)

Διεργασίες: Γονείς και Παιδιά

3

- Η γονική διεργασία πρέπει να κάνει “collect” το “παιδί” που τερματίστηκε.
- Μέχρι ο γονέας να κάνει “collect”, η τερματισμένη διεργασία παραμένει στον πίνακα διεργασιών με status: Zombie.
- Το “collect” γίνεται με τις system calls:
`wait` , `waitpid` (είναι υλοποιημένες στη C, αλλά υπάρχουν αντίστοιχες συναρτήσεις σε όλες τις γλώσσες προγραμματισμού)
- Η `wait` επιστρέφει στον γονέα τον “exit code” του παιδιού που τερματίστηκε.

Διεργασίες:Κωδικός εξόδου

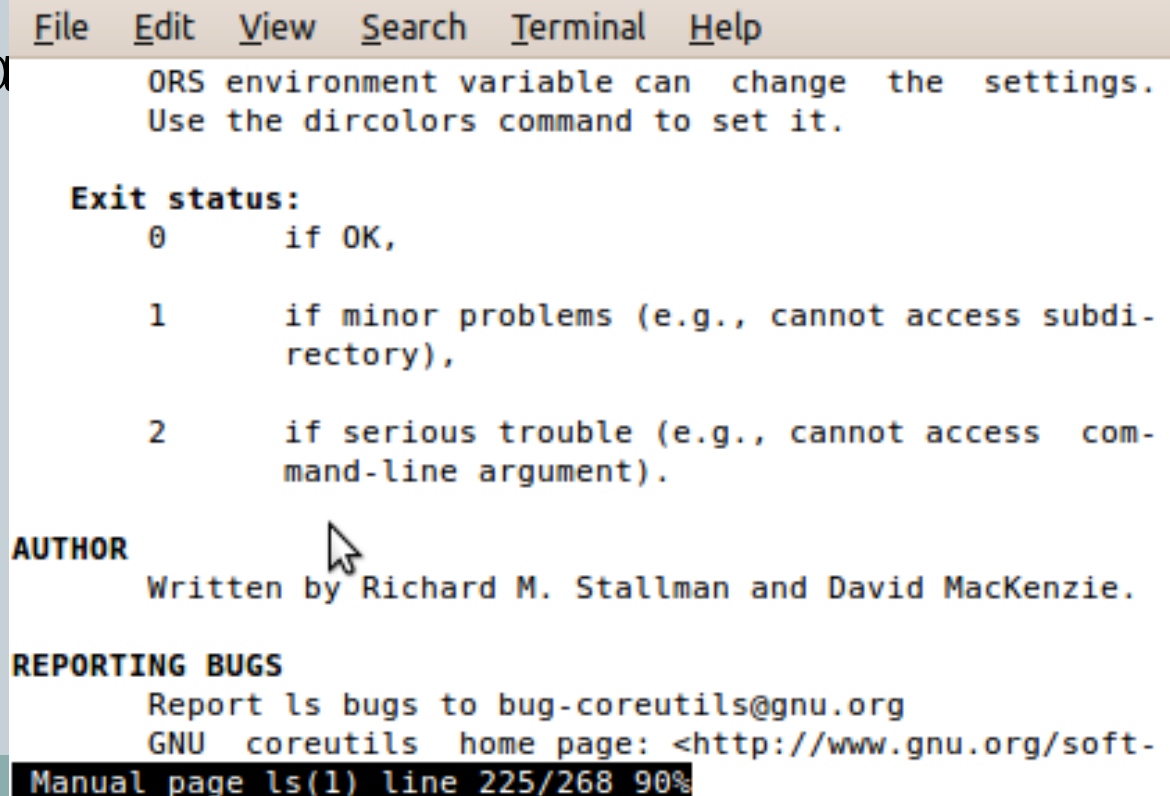
4

- Υπάρχει η σύμβαση, όταν η εκτέλεση του προγράμματος ήταν πετυχημένη (δεν συνέβη κάποιο σφάλμα) η διεργασία επιστρέφει 0. Διαφορετικά επιστρέφει έναν ακέραιο που εκφράζει το σφάλμα.
- Ο κωδικός εξόδου (exit code) συνήθως αναφέρεται και ως "error code" ή "exit status".
- Όταν η γονική διεργασία είναι το shell, χρησιμοποιεί το "exit code" του παιδιού που τερματίστηκε για να "καταλάβει" αν η διεργασία εκτελέστηκε σωστά.

Διεργασίες:Κωδικός εξόδου

5

- Στο manual page κάθε εντολής περιγράφεται κάθε περίπτωση και ποιους κωδικούς σφάλματος επιστρέφει η εντολή.
- Βλέπουμε δίπλα από το manual της εντολής ls.



The screenshot shows a terminal window with a menu bar containing 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The main content is the manual page for 'ls'. It includes a note about the 'ORS' environment variable, an 'Exit status:' section with three entries (0 for OK, 1 for minor problems, 2 for serious trouble), an 'AUTHOR' section crediting Richard M. Stallman and David MacKenzie, and a 'REPORTING BUGS' section with contact information for bug-coreutils@gnu.org and the GNU coreutils home page. The status bar at the bottom indicates 'Manual page ls(1) line 225/268 90%'. A mouse cursor is visible over the 'AUTHOR' section.

```
File Edit View Search Terminal Help
ORS environment variable can change the settings.
Use the dircolors command to set it.

Exit status:
0      if OK,

1      if minor problems (e.g., cannot access subdi-
       rectory),

2      if serious trouble (e.g., cannot access com-
       mand-line argument).

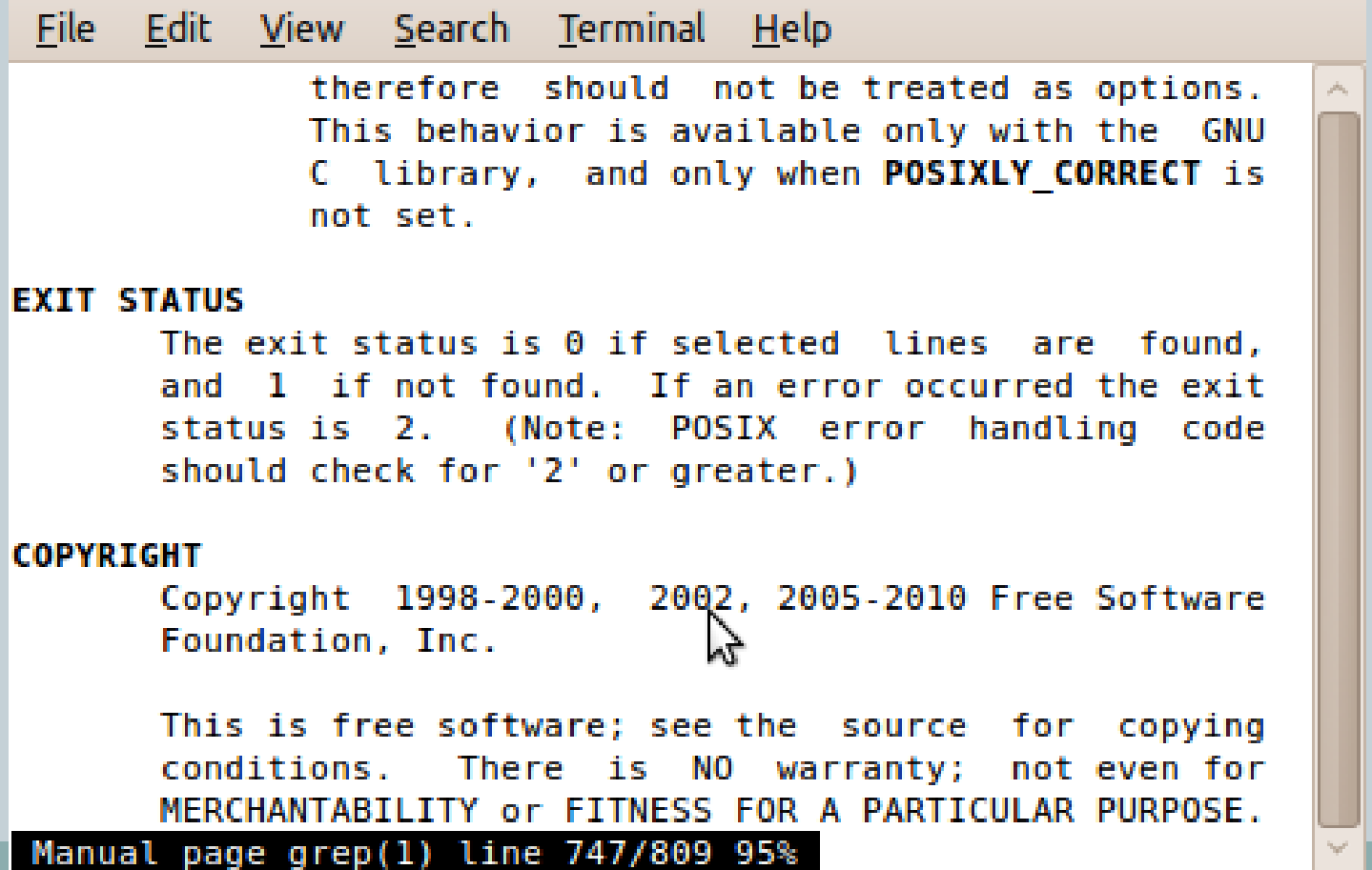
AUTHOR
       Written by Richard M. Stallman and David MacKenzie.

REPORTING BUGS
       Report ls bugs to bug-coreutils@gnu.org
       GNU coreutils home page: <http://www.gnu.org/soft-
Manual page ls(1) line 225/268 90%
```

Διεργασίες:Κωδικός εξόδου

6

- Βλέπουμε απόσπασμα από το manual της εντολής `grep`.



The screenshot shows a terminal window with a menu bar containing `File`, `Edit`, `View`, `Search`, `Terminal`, and `Help`. The main content area displays text from the `grep` manual page. The text is in a monospaced font with some words highlighted in blue. A mouse cursor is visible over the word `2002` in the copyright section. At the bottom, a status bar shows `Manual page grep(1) line 747/809 95%`.

```
File Edit View Search Terminal Help

therefore should not be treated as options.
This behavior is available only with the GNU
C library, and only when POSIXLY_CORRECT is
not set.

EXIT STATUS

The exit status is 0 if selected lines are found,
and 1 if not found. If an error occurred the exit
status is 2. (Note: POSIX error handling code
should check for '2' or greater.)

COPYRIGHT

Copyright 1998-2000, 2002, 2005-2010 Free Software
Foundation, Inc.

This is free software; see the source for copying
conditions. There is NO warranty; not even for
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Manual page grep(1) line 747/809 95%
```

shell & exit codes

7

- Το shell διαβάζει τον κωδικό που επέστρεψε η κάθε εντολή και τον αποθηκεύει στην μεταβλητή \$?.
- Το \$? περιέχει κάθε φορά το exit code της τελευταίας εντολής που εκτελέστηκε από το shell.

```
skyblue_asidirop_216_$ls -l /tmp > /dev/null
skyblue_asidirop_217_$echo $?
0
skyblue_asidirop_218_$ls -l /tmdddp > /dev/null
ls: /tmdddp: Δεν υπάρχει τέτοιο αρχείο ή κατάλογος
skyblue_asidirop_219_$echo $?
2
skyblue_asidirop_220_$echo $?
0
skyblue_asidirop_221_$
```

shell & exit codes

8

File Edit View Search Terminal Help

```
asidirop@dellpc:~$ man grep
asidirop@dellpc:~$ ls -l /tmp > /dev/null
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$ ls -l /pppp > /dev/null
ls: cannot access /pppp: No such file or directory
asidirop@dellpc:~$ echo $?
2
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$
```

περιέχει τον
κωδικό εξόδου
της ls

περιέχει τον κωδικό
εξόδου της
προηγούμενης echo

shell & exit codes

9

- Για κάθε εντολή το shell προσπαθεί να βρεί τον κωδικό εξόδου.
- Μπορούμε να ομαδοποιήσουμε εντολές μέσα σε μια λογική συνθήκη, και το shell υπολογίζει τον κωδικό εξόδου για την ομάδα εντολών.
- Σε αυτόν τον συνδυασμό εντολών μπορούμε να χρησιμοποιήσουμε:
 - Λογικό OR (||)
 - Λογικό AND (&&)
 - Λογικό NOT (!)

shell & exit codes

10

- Σύνταξη:

- εντολή1 ΛΤ εντολή2 ΛΤ εντολή3 ...
- (ΛΤ = Λογικός τελεστής)
- ! εντολή

Και οι 2 εντολές επέστρεψαν true. Το αποτέλεσμα της AND είναι true (ο)

```
asidirop@dellpc:~$ ls -l out-0013.pdf && stat out-0013.pdf
-rw-r----- 1 asidirop asidirop 849886 2010-01-10 19:25 out-0013.pdf
  File: `out-0013.pdf'
  Size: 849886          Blocks: 1672          IO Block: 4096    regular file
Device: 808h/2056d    Inode: 4401283       Links: 1
Access: (0640/-rw-r-----)  Uid: ( 1000/asidirop)   Gid: (
1000/asidirop)
Access: 2012-02-22 10:48:54.000000000 +0200
Modify: 2010-01-10 19:25:59.000000000 +0200
Change: 2010-04-16 21:42:21.000000000 +0300
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$
```

shell & exit codes

11

- `true && false = false`

Η 2^η εντολή
επέστρεψε false (1).
Το αποτέλεσμα της
AND είναι false (1)

```
asidirop@dellpc:~$ ls -l out-0013.pdf && stat xzy
-rw-r----- 1 asidirop asidirop 849886 2010-01-10 19:25 out-0013.pdf
stat: cannot stat `xzy': No such file or directory
asidirop@dellpc:~$ echo $?
1
asidirop@dellpc:~$
```

shell & exit codes

12

- **false && false = false**
- **false && true = false**
- **false && anything = false**
- Στην περίπτωση που το 1^ο σκέλος μιας AND είναι false, τότε οτιδήποτε και να είναι το 2^ο, το αποτέλεσμα της λογικής έκφρασης είναι false. Άρα δεν χρειάζεται να υπολογιστεί (να ληφθεί υπόψη) το 2^ο σκέλος.
- Αν η 1^η εντολή δώσει λάθος, τότε δεν εκτελείται η επόμενη (με &&)

Η 1^η εντολή επέστρεψε false (2). Το αποτέλεσμα της AND είναι false (2)

```
asidirop@dellpc:~$ ls -l zxy && stat out-0013.pdf
ls: cannot access zxy: No such file or directory
asidirop@dellpc:~$ echo $?
2
asidirop@dellpc:~$
```

shell & exit codes

13

- Το && χρησιμοποιείται όταν θέλουμε να εκτελέσουμε μια σειρά εντολών στις οποίες θέλουμε να προχωράμε στην επόμενη μόνο αν η προηγούμενη εκτελέστηκε σωστά.

δεν έχει νόημα να εκτελεστεί η 2^η εντολή αν η 1^η δώσει σφάλμα.

```
asidirop@dellpc:~$ file='os.txt'
asidirop@dellpc:~$ ls -l "$file" && wc -l "$file"
-rw-r--r-- 1 asidirop asidirop 5879 2009-11-03 21:19 os.txt
135 os.txt
asidirop@dellpc:~$ file='os2.txt'
asidirop@dellpc:~$ ls -l "$file" && wc -l "$file"
ls: cannot access os2.txt: No such file or directory
asidirop@dellpc:~$
```

shell & exit codes

14

- Το && χρησιμοποιείται όταν θέλουμε να εκτελέσουμε μια σειρά εντολών στις οποίες θέλουμε να προχωράμε στην επόμενη μόνο αν η προηγούμενη εκτελέστηκε σωστά.

```
asidirop@dellpc:~$ ls -l "$file" && wc -l "$file" && echo "OK all"
-rw-r--r-- 1 asidirop asidirop 5879 2009-11-03 21:19 os.txt
135 os.txt
OK all
asidirop@dellpc:~$ file='os2.txt'
asidirop@dellpc:~$ ls -l "$file" && wc -l "$file" && echo "OK all"
ls: cannot access os2.txt: No such file or directory
asidirop@dellpc:~$ echo $?
2
asidirop@dellpc:~$
```

shell & exit codes

15

- Το `||` χρησιμοποιείται όταν θέλουμε να εκτελέσουμε μια μόνο εντολή από μια σειρά εντολών.

Η 1^η εντολή έδωσε false, η 2^η εντολή έδωσε true, το αποτέλεσμα είναι true.

```
asidirop@dellpc:~$ ls out-*
out-0011.pdf  out-0013.pdf  out-0013.txt
asidirop@dellpc:~$ ls -l out-0011.txt || ls -l out-0011.pdf
ls: cannot access out-0011.txt: No such file or directory
-rw-r----- 1 asidirop asidirop 1974962 2009-12-13 12:59 out-0011.pdf
asidirop@dellpc:~$ echo $?
0
```

shell & exit codes

16

- `false || false = false`
 - `false || true = true`
 - `true || false = true`
 - `true || true = true`
- } `true || any = true`

Η 1^η εντολή έδωσε true, το αποτέλεσμα της OR είναι true.

```
asidirop@dellpc:~$ ls out-*
out-0011.pdf  out-0013.pdf  out-0013.txt
asidirop@dellpc:~$ ls -l out-0013.txt || ls -l out-0013.pdf
-rw-r--r-- 1 asidirop asidirop 0 2012-05-16 10:39 out-0013.txt
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$
```


shell & exit codes

17

- Αν η 1^η εντολή εκτελεστεί σωστά, τότε δεν εκτελούνται οι υπόλοιπες.
- αν η 1^η εντολή δεν εκτελεστεί σωστά, τότε εκτελείται και η επόμενη.

```
asidirop@dellpc:~$ file='os.txt'
asidirop@dellpc:~$ wc -l "$file" || echo "File $file NOT
FOUND"
135 os.txt
asidirop@dellpc:~$ file='os2.txt'
asidirop@dellpc:~$ wc -l "$file" || echo "File $file NOT
FOUND"
wc: os2.txt: No such file or directory
File os2.txt NOT FOUND
asidirop@dellpc:~$
```

shell & exit codes

18

- Το ! είναι το λογικό NOT.
- ΠΡΟΣΟΧΗ: **πρέπει να υπάρχει κενό μετά το !.**
πχ αν γράψουμε !wc σημαίνει επανάληψη της τελευταίας εντολής που ξεκινούσε με wc.

```
asidirop@dellpc:~$ file='os2.txt'
asidirop@dellpc:~$
asidirop@dellpc:~$ ! wc -l "$file" && echo "File $file NOT
FOUND"
wc: os2.txt: No such file or directory
File os2.txt NOT FOUND
asidirop@dellpc:~$ file='os.txt'
asidirop@dellpc:~$ ! wc -l "$file" && echo "File $file NOT
FOUND"
135 os.txt
asidirop@dellpc:~$
```

shell & exit codes με διασωλήνωση

19

- Στην περίπτωση της διασωλήνωσης δεν υπολογίζεται το συνολικό "exit status" όλων των εντολών, αλλά στο \$? αποθηκεύεται μόνο της τελευταίας.

```
asidirop@dellpc:~$ ls /tt| wc
ls: cannot access /tt: No such file or directory
0          0          0
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$ ls | wc -ppp
wc: invalid option -- 'p'
Try `wc --help' for more information.
asidirop@dellpc:~$ echo $?
1
asidirop@dellpc:~$
```

Η εντολή if

20

- **Σύνταξη:**

```
if entolh-elegxou ; then
    Commands...
elif entolh-elegxou ; then
    Commands...
elif entolh-elegxou ; then
    Commands...
else
    Commands...
fi
```

- Τα τμήματα 'elif' μπορούν να είναι όσα επιθυμούμε, όπως επίσης και κανένα.
- Το τμήμα 'else' μπορεί να είναι το πολύ ένα ή κανένα.
- Υποχρεωτικά το block της if τερματίζει-ολοκληρώνεται με το fi.

Η εντολή if

21

- Μπορούμε να συντάξουμε μια εντολή if στην γραμμή εντολών (όμως πρακτικά είναι λίγο δύσκολος ο χειρισμός εντολής με πολλές γραμμές)
- Στο παρακάτω, εκτελείται η εντολή `ls "$dir"`, και εφόσον εκτελείται σωστά (επιστρέφει true/0) μπαίνουμε και στο block "then".

```
asidirop@dellpc:~$ dir="/tmp"
asidirop@dellpc:~$ if ls "$dir" ; then
> echo "OK"
> fi
file1          file2
file3          file4
OK
```

Η εντολή if

22

- Στο παρακάτω, εκτελείται η εντολή `ls "$dir"`, και εφόσον εκτελείται σωστά (επιστρέφει true/0) μπαίνουμε και στο block "then".
- Το block "else" προφανώς δεν εκτελείται.

```
asidirop@dellpc:~$ dir="/tmp"
asidirop@dellpc:~$ if ls "$dir" ; then
> echo "OK"
> else
> echo "NOT OK"
> fi
file1          file2
file3          file4
OK
asidirop@dellpc:~$
```

Η εντολή if

23

- Στο παρακάτω, εκτελείται η εντολή `ls "$dir"`, και ΔΕΝ εκτελείται σωστά (επιστρέφει `false/ !=0`)
- προφανώς εκτελείται το block `"else"` .

```
asidirop@dellpc:~$ dir='/sdfd'
asidirop@dellpc:~$ if ls "$dir" ; then
> echo "OK"
> else
> echo "NOT OK"
> fi
ls: cannot access /sdfd: No such file or directory
NOT OK
asidirop@dellpc:~$
```

Η εντολή test

24

- Συνήθως, θέλουμε να ελέγξουμε το αποτέλεσμα μιας αριθμητικής συνθήκης ή μια σύγκριση από strings.
- Η if από μόνη της δεν είναι ικανή να κάνει αυτούς τους ελέγχους.
- Υπάρχει η εντολή test, η οποία δέχεται ως όρισμα μια συνθήκη-έλεγχο και επιστρέφει ως exit code
 - true (0) αν η συνθήκη είναι αληθής
 - false (>0) αν η συνθήκη είναι ψευδής

Η εντολή test

25

- Η εντολή test μπορεί να πραγματοποιήσει διάφορες κατηγορίες ελέγχων:
 - σε strings
 - σε ακεραίους
 - σε αρχεία (file system)
 - να υπολογίσει λογικούς τελεστές (AND, OR, NOT)

Η εντολή test

26

- Έλεγχοι σε strings

<i>Expression</i>	<i>Description</i>
<code>-z string</code>	True if <i>string</i> is empty.
<code>-n string</code>	True if <i>string</i> is not empty.
<code>string1 = string2</code>	True if <i>string1</i> equals <i>string2</i> .
<code>string1 != string2</code>	True if <i>string1</i> does not equal <i>string2</i> .

true

```
asidirop@dellpc:~$ test HELLO = HELLO
asidirop@dellpc:~$ echo $?
0
```

false

```
asidirop@dellpc:~$ test HELLO = AAAA
asidirop@dellpc:~$ echo $?
1
```

true

```
asidirop@dellpc:~$ test -z ''
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$
```

Η εντολή test

27

- Έλεγχοι σε ακραίους

<i>Expression</i>	<i>Description</i>
INTEGER1 -eq INTEGER2	INTEGER1 is equal (=) to INTEGER2
INTEGER1 -ge INTEGER2	INTEGER1 is greater than or equal (>=) to INTEGER2
INTEGER1 -gt INTEGER2	INTEGER1 is greater than (>) INTEGER2
INTEGER1 -le INTEGER2	INTEGER1 is less than or equal (<=) to INTEGER2
INTEGER1 -lt INTEGER2	INTEGER1 is less than (<) INTEGER2
INTEGER1 -ne INTEGER2	INTEGER1 is not equal (!=) to INTEGER2

Η εντολή test

28

**5>10
false**

```
asidirop@dellpc:~$ test 5 -gt 10
asidirop@dellpc:~$ echo $?
1
```

**50>10
true**

```
asidirop@dellpc:~$ test 50 -gt 10
asidirop@dellpc:~$ echo $?
0
```

**5==005
true**

```
asidirop@dellpc:~$ test '5' -eq '005'
asidirop@dellpc:~$ echo $?
0
```

**'5'=='005'
false**

```
asidirop@dellpc:~$ test '5' = '005'
asidirop@dellpc:~$ echo $?
1
```

**false (2)
με
σφάλμα**

```
asidirop@dellpc:~$ test '5' -eq '005x'
bash: test: 005x: integer expression expected
asidirop@dellpc:~$ echo $?
2
asidirop@dellpc:~$
```

Η εντολή test

29

- Έλεγχοι σε αρχεία (Περισσότεροι έλεγχοι στο manual page της test)

<i>Expression</i>	<i>Description</i>
<code>-d file</code>	True if <i>file</i> is a directory.
<code>-e file</code>	True if <i>file</i> exists.
<code>-f file</code>	True if <i>file</i> exists and is a regular file.
<code>-L file</code>	True if <i>file</i> is a symbolic link.
<code>-r file</code>	True if <i>file</i> is a file readable by you.
<code>-w file</code>	True if <i>file</i> is a file writable by you.
<code>-x file</code>	True if <i>file</i> is a file executable by you.
<code>-s file</code>	true if file exists and has a size greater than zero.
<code>file1 -nt file2</code>	True if <i>file1</i> is newer than (according to modification time) <i>file2</i>
<code>file1 -ot file2</code>	True if <i>file1</i> is older than <i>file2</i>

Η εντολή test

30

υπάρχει?
true

```
asidirop@dellpc:~$ test -e Documents
asidirop@dellpc:~$ echo $?
0
```

είναι αρχείο?
false

```
asidirop@dellpc:~$ test -f Documents
asidirop@dellpc:~$ echo $?
1
```

είναι κατάλογος?
true

```
asidirop@dellpc:~$ test -d Documents
asidirop@dellpc:~$ echo $?
0
```

έχω δικαίωμα read?
true

```
asidirop@dellpc:~$ test -r Documents
asidirop@dellpc:~$ echo $?
0
```

έχω δικαίωμα read?
true

```
asidirop@dellpc:~$ test -r /bin
asidirop@dellpc:~$ echo $?
0
```

έχω δικαίωμα write?
false

```
asidirop@dellpc:~$ test -w /bin
asidirop@dellpc:~$ echo $?
1
```

```
asidirop@dellpc:~$
```

Η εντολή test

31

- Λογικοί τελεστές

<i>Expression</i>	<i>Description</i>
!	unary negation operator.
-a	binary and operator.
-o	binary or operator (-a has higher precedence than -o).
(expr)	parentheses for grouping.

a>3 AND
a<100
true

a>3 AND
a<100
false

```
asidirop@dellpc:~$ a=5
asidirop@dellpc:~$ test "$a" -gt 3 -a "$a" -lt 100
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$ a=1
asidirop@dellpc:~$ test "$a" -gt 3 -a "$a" -lt 100
asidirop@dellpc:~$ echo $?
1
```

Η εντολή test

32

- Η test (όπως και η expr) θέλει κάθε τελεστής/τελεστέος να είναι διαφορετικό όρισμα.

!!!!

δεν δούλεψε σωστά

```
asidirop@dellpc:~$ test "HELLO"="HELLO2"  
asidirop@dellpc:~$ echo $?  
0
```

false

```
asidirop@dellpc:~$ test "HELLO" = "HELLO2"  
asidirop@dellpc:~$ echo $?  
1
```

!!!!

δεν δούλεψε σωστά

```
asidirop@dellpc:~$ test 5-gt 1  
bash: test: 5-gt: unary operator expected  
asidirop@dellpc:~$ echo $?  
2  
asidirop@dellpc:~$
```


Η εντολή test

33

- Οι () έχουν ειδική σημασία για το shell. Για να τις χρησιμοποιήσουμε στην test, πρέπει να αναιρέσουμε την ειδική σημασία.

!!!!

δεν δούλεψε σωστά

```
asidirop@dellpc:~$ b=1
asidirop@dellpc:~$ a=5
asidirop@dellpc:~$ test ( "$a" -gt 1 -a
"$a" -lt 100 ) -o "$b" -ne 0
bash: syntax error near unexpected token
`"$a"'
asidirop@dellpc:~$ echo $?
2
asidirop@dellpc:~$ test \( "$a" -gt 1 -a
"$a" -lt 100 \) -o "$b" -ne 0
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$
```

δούλεψε σωστά

Η εντολή test

34

- προσοχή αν κάποια μεταβλητή δεν έχει τιμή:

δεν δούλεψε σωστά,
σαν να έγραφα:
`test = "HELLO"`

δούλεψε σωστά
`false`

δούλεψε σωστά
`true`

```
asidirop@dellpc:~$ test $w = "HELLO"
bash: test: ==: unary operator expected
asidirop@dellpc:~$ echo $?
2
asidirop@dellpc:~$ test "$w" = "HELLO"
asidirop@dellpc:~$ echo $?
1
asidirop@dellpc:~$ w='HELLO'
asidirop@dellpc:~$ test "$w" = "HELLO"
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$
```

Η εντολή test

35

- προσοχή με την χρήση αριθμών:

δεν δούλεψε σωστά,
σαν να έγραφα:
`test -eq 0`

δεν δούλεψε σωστά,
σαν να έγραφα:
`test "" -eq 0`

δούλεψε σωστά μετά
από αρχικοποίηση
τη w
`true`

```
asidirop@dellpc:~$ test $w -eq 0
bash: test: -eq: unary operator expected
asidirop@dellpc:~$ echo $?
2
asidirop@dellpc:~$ test "$w" -eq 0
bash: test: : integer expression expected
asidirop@dellpc:~$ echo $?
2
asidirop@dellpc:~$ w=0
asidirop@dellpc:~$ test "$w" -eq 0
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$
```

test & if

36

- Η test συνήθως χρησιμοποιείται σε συνδυασμό με την if:

FILE: test_test1

```
if test -f ~/.bash_profile ; then
    echo "You have a .bash_profile."
else
    echo "OOOps! You have no .bash_profile!"
fi
```

test & if

37

- Η test συνήθως χρησιμοποιείται σε συνδυασμό με την if:

FILE: test_test2

```
echo -n "give a number less than 10: "  
read a  
if test "$a" -ge 10 ; then  
    echo "You gave wrong number"  
else  
    echo "OK. You gave $a"  
fi
```

test & if

38

- Αν το προηγούμενο ήταν γραμμένο σε κάποια άλλη γλώσσα προγραμματισμού (java,C,C++,Javascript, κτλ.) η if θα συντάσσονταν:
if(a>=10) ...
- Η χρήση της λέξης test καθώς και το ότι δεν είναι εμφανές που τελειώνει η συνθήκη -- δεν υπάρχει παρένθεση που κλείνει, δυσκολεύει τους προγραμματιστές που έχουν συνηθίσει σε άλλες γλώσσες.

FILE: test_test2

```
echo -n "give a number less than 10: "  
read a  
if test "$a" -ge 10 ; then  
    echo "You gave wrong number"  
else  
    echo "OK. You gave $a"  
fi
```

[= test

39

- Η [είναι εντολή και είναι ισοδύναμη με την εντολή test.
- Η εντολή [περιμένει η τελευταία παράμετρος που θα της δώσουμε να είναι το string] (για λόγους συμμετρίας και ομορφιάς), αλλιώς μας δίνει μήνυμα λάθους.

FILE: test_test3

```
echo -n "give a number less than 10: "  
read a  
if [ "$a" -ge 10 ]; then  
    echo "You gave wrong number"  
else  
    echo "OK. You gave $a"  
fi
```

[= test

40

- Μια εντολή από τα ορίσματα της χωρίζεται με space. Άρα μετά την [πρέπει να υπάρχει space.
- Στο παράδειγμα το "10" είναι το 3^ο όρισμα και η "]" είναι το 4^ο. Τα ορίσματα μεταξύ τους χωρίζονται με space. Άρα πριν την τελευταία] πρέπει να υπάρχει space.

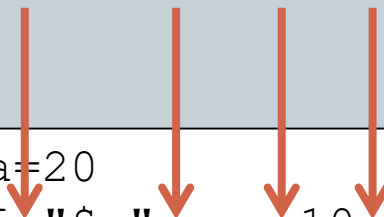
FILE: test_test3

```
echo -n "give a number less than 10: "  
read a  
if [ "$a" -ge 10 ]; then  
    echo "You gave wrong number"  
else  
    echo "OK. You gave $a"  
fi
```


[= test

41

- Πρέπει να υπάρχουν spaces...



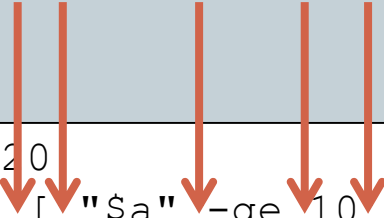
```
asidirop@dellpc:~$ a=20
asidirop@dellpc:~$ [ "$a" -ge 10 ]
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$ [ "$a" -ge 10 ]
[20: command not found
asidirop@dellpc:~$ [ "$a" -ge 10]
bash: [: missing `]'
asidirop@dellpc:~$ [ "$a" -ge10 ]
bash: [: 1: unary operator expected
asidirop@dellpc:~$
```

- Διαφορετικά θα γίνει εσφαλμένη ερμηνεία από το shell και έπειτα από την test

[= test

42

- Ομοίως όταν η [χρησιμοποιείται με την if
- επιπλέον πρέπει να υπάρχει space μετά την if και πριν την [



```
asidirop@dellpc:~$ a=20
asidirop@dellpc:~$ if [ "$a" -ge 10 ] ; then echo "something"; fi
something
asidirop@dellpc:~$ if ["$a" -ge 10 ] ; then echo "something"; fi
[20: command not found
asidirop@dellpc:~$ if [ "$a" -ge 10] ; then echo "something"; fi
bash: [: missing `]'
asidirop@dellpc:~$ if [ "$a" -ge10 ] ; then echo "something"; fi
bash: [: 20: unary operator expected
asidirop@dellpc:~$
```

Συνήθη έλεγχοι

43

- Θέλουμε να κάνουμε ένα script το οποίο θα δέχεται ακριβώς 2 παραμέτρους

Το μήνυμα σφάλματος πρέπει να το στείλουμε την τυπική έξοδο λαθών

Πρέπει να κάνουμε exit με κωδικό σφάλματος, ώστε το λειτουργικό και το shell να καταλάβει ότι κάτι δεν πήγε καλά.

FILE: test_test4

```
if [ "$#" -ne 2 ] ; then
    echo "Wrong number of args." 1>&2
    echo "You must give 2 args." 1>&2
    exit 1
fi
```

Συνήθη έλεγχοι

44

- Το 1^ο όρισμα θέλουμε να είναι όνομα αρχείου (διαδρομή προς αρχείο)

Το μήνυμα σφάλματος πρέπει να το στείλουμε την τυπική έξοδο λαθών

Πρέπει να κάνουμε exit με κωδικό σφάλματος. Αυτός μπορεί να είναι διαφορετικός σε κάθε περίπτωση σφάλματος.

FILE: test_test4

```
...  
if [ ! -f "$1" ] ; then  
    echo "$1 is not a file." 1>&2  
    exit 2  
fi
```

Συνήθη έλεγχοι

45

- Το 2^ο όρισμα θέλουμε να είναι όνομα καταλόγου (διαδρομή προς κατάλογο)

Πρέπει να κάνουμε exit με κωδικό σφάλματος.

FILE: test_test4

```
...  
if [ ! -d "$2" ] ; then  
    echo "$2 is not a directory." 1>&2  
    exit 3  
fi
```

Συνήθη έλεγχοι

46

FILE: test_test4

```
if [ "$#" -ne 2 ] ; then
    echo "Wrong number of args." 1>&2
    echo "You must give 2 args." 1>&2
    exit 1
fi
if [ ! -f "$1" ] ; then
    echo "$1 is not a file." 1>&2
    exit 2
fi
if [ ! -d "$2" ] ; then
    echo "$2 is not a directory." 1>&2
    exit 3
fi
echo "File: $1"
echo "Directory: $2"
```

Συνήθη έλεγχοι

47

FILE: test_test4

```
if [ "$#" -ne 2 ] ; then
    echo "Wrong number of args." 1>&2
    echo "You must give 2 args." 1>&2
    exit 1
fi
file="$1"
dir="$2"
if [ ! -f "$file" ] ; then
    echo "$file is not a file." 1>&2
    exit 2
fi
if [ ! -d "$dir" ] ; then
    echo "$dir is not a directory." 1>&2
    exit 3
fi
echo "File: $file"
echo "Directory: $dir"
```

Αριθμητικές παραστάσεις στο bash

- Σε προηγούμενο μάθημα αναφέρθηκαν οι αριθμητικές πράξεις που μπορεί να διεκπεραιώσει το bash.

πράξη	εξήγηση
id++ id--	variable post-increment and post-decrement
++id --id	variable pre-increment and pre-decrement
- +	unary minus and plus
! ~	logical and bitwise negation
**	exponentiation
* , / , %	multiplication, division, remainder
+, -	addition, subtraction
<< , >>	left and right bitwise shifts
<= , >= , < , >	comparison
== , !=	equality and inequality
& , ^ ,	bitwise AND, bitwise exclusive OR, bitwise OR
&& ,	logical AND, logical OR
expr?expr:expr r	conditional operator
= *= /= %= += -= <<= >>= &= ^= =	assignment

Αριθμητικές παραστάσεις στο bash

49

- όταν μέσα στις (()) περιέχεται έλεγχος, τότε επιστρέφεται ως 'exit status' το αποτέλεσμα του ελέγχου.

false

true

true

αριθμητική σύγκριση

true

αριθμητική σύγκριση

false

η μεταβλητή test=0

true

```
asidirop@dellpc:~$ ((5>10))
asidirop@dellpc:~$ echo $?
1
asidirop@dellpc:~$ ((50>10))
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$ ((5==05))
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$ (('5'=='05'))
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$ (('test'==5))
asidirop@dellpc:~$ echo $?
1
asidirop@dellpc:~$ (('test'==0))
asidirop@dellpc:~$ echo $?
0
asidirop@dellpc:~$
```

Αριθμητικές παραστάσεις στο bash

50

- εφόσον τα (()) επιστρέφουν exit status, Μπορούν να χρησιμοποιηθούν και μέσα στην if.

false → μπαίνει
στο else.

true → μπαίνει
στο then.

```
asidirop@dellpc:~$ if (( 5>10 )) ; then
> echo "gt"
> else
> echo "le"
> fi
le
asidirop@dellpc:~$ if (( 50>10 )) ; then
> echo "gt"
> else
> echo "le"
> fi
gt
asidirop@dellpc:~$
```

Αριθμητικές παραστάσεις στο bash

51

- Στην περίπτωση χρήσης των (()), δεν απαιτείται η ύπαρξη κενών διαστημάτων πριν και μετά τις (()), διότι οι (()) είναι ειδικοί χαρακτήρες.
- μέσα στις (()) δεν απαιτείται η ύπαρξη κενών διαστημάτων διότι το εσωτερικό γίνεται parse (ερμηνεύεται) ως αριθμητική παράσταση από το bash.

```
asidirop@dellpc:~$ if (( 50 > 10 )) ; then echo "gt"; else  
echo "le"; fi  
gt  
asidirop@dellpc:~$ if((50>10)); then echo "gt"; else echo  
"le"; fi  
gt  
asidirop@dellpc:~$
```

Συνήθη έλεγχοι

52

- Στο παράδειγμα στο οποίο ζητούνται 2 ορίσματα, ο πρώτος έλεγχος (αριθμητικός) θα μπορούσε να γίνει με την χρήση (()).
- Οι υπόλοιποι ΌΧΙ. δεν είναι αριθμητικοί έλεγχοι.


FILE: test_test4

```
if (( $# != 2 )) ; then
    echo "Wrong number of args." 1>&2
    echo "You must give 2 args." 1>&2
    exit 1
fi
file="$1"
dir="$2"
if [ ! -f "$file" ] ; then
    echo "$file is not a file." 1>&2
    exit 2
fi
if [ ! -d "$dir" ] ; then
    echo "$dir is not a directory." 1>&2
    exit 3
fi
echo "File: $file"
echo "Directory: $dir"
```

bash και [[]]

53

- Για τους υπόλοιπους ελέγχους που μπορεί να πραγματοποιήσει η εντολή `test`, δηλαδή η εντολή `[`, το `bash` έχει αντίστοιχη εσωτερική εντολή (built-in): την `[[]]`.
- η χρήση των `[[]]` είναι παρόμοια με την χρήση των `[]`, μόνο που τους ελέγχους τους κάνει το ίδιο το shell και δεν εκτελείται η επιπλέον εντολή (η `test`). άρα η χρήση των `[[]]` υπερτερεί στην ταχύτητα εκτέλεσης.
- Η χρήση των κενών διαστημάτων είναι πάλι υποχρεωτική.



```
asidirop@dellpc:~$ if [[ -d WORKING.ods ]]; then echo "dir exists";fi
asidirop@dellpc:~$ if [[ -f WORKING.ods ]]; then echo "file
exists";fi
file exists
asidirop@dellpc:~$ if[[ -f WORKING.ods ]]; then echo "file exists";fi
bash: syntax error near unexpected token `then'
asidirop@dellpc:~$
```