

Εισαγωγή στα Λειτουργικά Συστήματα



SET ΔΙΑΦΑΝΕΙΩΝ 15

SHELL SCRIPTS: BLOCKS ΕΝΤΟΛΩΝ

ΑΝΤΩΝΗΣ ΣΙΔΗΡΟΠΟΥΛΟΣ

blocks εντολών

2

- Τα blocks εντολών στο sh είναι:
 - (εντολές)
 - while ... do εντολές done ή σε if, until
- Στο bash υπάρχει και η δυνατότητα:
 - { εντολές }

(εντολές)

3

- χρησιμοποιώντας τις () δημιουργείται ένα νέο shell και εκτελεί τις εντολές που δίνονται μέσα στην παρένθεση.
- όλες οι εντολές κληρονομούν τα χαρακτηριστικά του shell που δημιουργείται (I/O streams, etc)
- μπορεί ο χρήστης να ορίσει ανακατεύθυνση-διασωλήνωση στο νέο shell.

(εντολές)

4

```
asidirop@antonis-PC:/tmp$ ( ls -l ; wc -l /etc/passwd ) > file1
asidirop@antonis-PC:/tmp$ more file1
total 192
-rw----- 1 root      root          0 2012-05-23 18:27 dkms.0OunigbS
-rw----- 1 root      root          0 2012-05-23 18:26 dkms.4PyreLnt
-rw----- 1 root      root          0 2012-05-23 18:27 dkms.7kCUFYKK
-rw----- 1 root      root          0 2012-05-23 18:26 dkms.bfoSWyRB
drwx----- 2 nikolia  nikolia      4096 2012-05-23 16:49 seahorse-IfCVlD
drwx----- 2 asidirop asidirop    4096 2012-05-23 17:29 seahorse-Z9pP57
drwx----- 2 asidirop asidirop    4096 2012-05-23 17:29 virtual-asidirop.uNvdpX
drwx----- 2 nikolia  nikolia      4096 2012-05-23 16:49 virtual-nikolia.BEIhbj
44 /etc/passwd
asidirop@antonis-PC:/tmp$
```

- ορίζουμε ως κανονική έξοδο το αρχείο file1. Όλες οι εντολές εσωτερικά των () κληρονομούν το ίδιο I/O stream.
- Το αρχείο file1 ανοίγει για εγγραφή (δημιουργείται) ΠΡIN ξεκινήσει η εκτέλεση των εντολών στις (), και κλείνει (close) όταν ολοκληρωθούν οι εντολές και τερματίσει το shell.
- Στα περιεχόμενα του αρχείου προστίθενται τα δεδομένα από κάθε εντολή μέσα στις ().

(εντολές)

5

```
asidirop@antonis-PC:/tmp$ (read a; read b; echo "a=$a"; echo  
"b=$b") < file1  
a=total 192  
b=-rw----- 1 root      root          0 2012-05-23 18:27  
dkms.0OunigbS  
asidirop@antonis-PC:/tmp$
```

- ορίζουμε ως κανονική είσοδο το αρχείο file1. Όλες οι εντολές εσωτερικά των () κληρονομούν το ίδιο I/O stream.
- Το αρχείο file1 ανοίγει για ανάγνωση ΠΡIN ξεκινήσει η εκτέλεση των εντολών στις (), και κλείνει (close) όταν ολοκληρωθούν οι εντολές και τερματίσει το shell.
- Όλες οι εντολές μέσα στις () θα διαβάσουν από το αρχείο file1.
- Κάθε εντολή θα διαβάζει τα επόμενα.... στο παράδειγμα η read a διαβάσει την 1^η γραμμή, η read b την 2^η. Αν υπήρχε και άλλη εντολή που θα προσπαθούσε να διαβάσει από την κανονική είσοδο θα διάβαζε τα επόμενα.

(εντολές)

6

```
asidirop@antonis-PC:/tmp$ (read a < file1; read b <
file1; echo "a=$a"; echo "b=$b")
a=total 192
b=total 192
asidirop@antonis-PC:/tmp$
```

- Το `read a<file1` έχει ως αποτέλεσμα να ανοίξει το αρχείο `file1` για ανάγνωση (πριν ξεκινήσει να εκτελείται η `read a`). Η εντολή `read` διαβάζει την 1^η γραμμή του αρχείου. τερματίζει η εντολή και κλείνει το αρχείο (`close`)
- Το `read b<file1` έχει ως αποτέλεσμα να ανοίξει το αρχείο `file1` για ανάγνωση (πριν ξεκινήσει να εκτελείται η `read b`). Η εντολή `read` διαβάζει την 1^η γραμμή του αρχείου. τερματίζει η εντολή και κλείνει το αρχείο (`close`)
- → Οι 2 εκτελέσεις των εντολών είναι ανεξάρτητες. Η κάθε μια διαβάζει την πρώτη γραμμή του αρχείου.

(εντολές)



```
asidirop@antonis-PC:/tmp$ (read a; read b;) < file1 ; echo  
"a=$a"; echo "b=$b"
```

```
a=
```

```
b=
```

```
asidirop@antonis-PC:/tmp$
```

- Στην περίπτωση που διαβάσουμε (ή ορίσουμε) τις μεταβλητές μέσα στο μπλόκ των (), τότε δεν μπορούμε να τις χρησιμοποιήσουμε έξω από το μπλοκ. Οι τιμές τους έχουν χαθεί !!!

(εντολές)

8

```
asidirop@antonis-PC:~$ echo $a
5
asidirop@antonis-PC:~$ ( a=10 ; echo $a)
10
asidirop@antonis-PC:~$ echo $a
5
asidirop@antonis-PC:~$
```

- Με τις (), δημιουργείται μια νέα διεργασία shell που εκτελεί τις εντολές στις ().
- ➔ μέσα και έξω από τις () δεν έχουμε τις ίδιες μεταβλητές.

(εντολές)

9

```
asidirop@antonis-PC:~$ ps
  PID TTY          TIME CMD
 3001 pts/0        00:00:00 bash
16607 pts/0        00:00:00 ps
asidirop@antonis-PC:~$ (ls>/dev/null ; ps)
  PID TTY          TIME CMD
 3001 pts/0        00:00:00 bash
16677 pts/0        00:00:00 bash
16679 pts/0        00:00:00 ps
asidirop@antonis-PC:~$
```

- Μπορούμε να το επιβεβαιώσουμε εκτελώντας την εντολή `ps` μέσα στις `()`. → έχει δημιουργηθεί μια νέα διεργασία `bash`.

{ εντολές }

10

```
asidirop@antonis-PC:/tmp$ { ls -l ; wc -l /etc/passwd ; } > file1
asidirop@antonis-PC:/tmp$ more file1
total 192
-rw----- 1 root      root          0 2012-05-23 18:27 dkms.0OunigbS
-rw----- 1 root      root          0 2012-05-23 18:26 dkms.4PyreLnt
-rw----- 1 root      root          0 2012-05-23 18:27 dkms.7kCUFYKK
-rw----- 1 root      root          0 2012-05-23 18:26 dkms.bfoSWyRB
drwx----- 2 nikolia  nikolia    4096 2012-05-23 16:49 seahorse-IfCVlD
drwx----- 2 asidirop asidirop   4096 2012-05-23 17:29 seahorse-Z9pP57
drwx----- 2 asidirop asidirop   4096 2012-05-23 17:29 virtual-asidirop.uNvdpX
drwx----- 2 nikolia  nikolia    4096 2012-05-23 16:49 virtual-nikolia.BEIhbj
44 /etc/passwd
asidirop@antonis-PC:/tmp$
```

- ορίζουμε ως κανονική έξοδο το αρχείο file1. Όλες οι εντολές εσωτερικά των {} κληρονομούν το ίδιο I/O stream.
- Ισχύουν ακριβώς τα ίδια με την περίπτωση των ().
- Η { πρέπει να χωρίζεται με space από την εντολή που ακολουθεί.
- Πριν από την } πρέπει να υπάρχει «;» (ή αλλαγή γραμμής)

{ εντολές }

11

```
asidirop@antonis-PC:/tmp$ { read a; read b; echo "a=$a"; echo  
"b=$b" ;} < file1  
a=total 192  
b=-rw----- 1 root      root          0 2012-05-23 18:27  
dkms.0OunigbS  
asidirop@antonis-PC:/tmp$
```

- Το ίδιο ακριβώς με την περίπτωση των ().

{ εντολές }

12

```
asidirop@antonis-PC:/tmp$ { read a; read b; } <file1 ;echo  
"a=$a"; echo "b=$b"  
a=total 192  
b=-rw----- 1 root      root          0 2012-05-23 18:27  
dkms.0OunigbS  
asidirop@antonis-PC:/tmp$
```

- Εδώ δεν έχουμε την ίδια συμπεριφορά με τις (). Όταν χρησιμοποιούνται {} για ορισμό μπλοκ εντολών, τότε οι μεταβλητές είναι «ορατές» και έξω από το μπλοκ.

{ εντολές }

13

```
asidirop@antonis-PC:/tmp$ a=5
asidirop@antonis-PC:/tmp$ { a=10 ; echo $a ; }
10
asidirop@antonis-PC:/tmp$ echo $a
10
asidirop@antonis-PC:/tmp$
```

- Με τις `{ }`, ΔΕΝ δημιουργείται μια νέα διεργασία shell, αλλά οι εντολές ερμηνεύονται και εκτελούνται από το τρέχον shell.
- ➔ μέσα και έξω από τις `{ }` έχουμε τις ίδιες μεταβλητές.

{ εντολές }

14

```
asidirop@antonis-PC:/tmp$ ps
  PID TTY          TIME CMD
 3001 pts/0        00:00:00 bash
20485 pts/0        00:00:00 ps
asidirop@antonis-PC:/tmp$ { ls> /dev/null; ps ; }
  PID TTY          TIME CMD
 3001 pts/0        00:00:00 bash
20557 pts/0        00:00:00 ps
asidirop@antonis-PC:/tmp$
```

- Μπορούμε να το επιβεβαιώσουμε εκτελώντας την εντολή `ps` μέσα στις `{}` . → ΔΕΝ έχει δημιουργηθεί νέα διεργασία `bash`.

blocks σε if,while,until

15

- Αντίστοιχα και με τα blocks εντολών σε if, while, until, μπορούμε να κάνουμε «συνολική» ανακατεύθυνση, Δηλαδή όλες οι εντολές μέσα στο block κληρονομούν το αρχείο file2 ως έξοδο. Στο αρχείο θα περιέχονται όλα τα δεδομένα που «τύπωσαν» οι εντολές.
- Το αρχείο δημιουργείται πριν ξεκινήσει να εκτελείται η if (while, until), και γίνεται close μόλις ολοκληρωθούν.

```
asidirop@antonis-PC:/tmp$ if true ; then echo
"TRUE" ; ls; fi > file2
asidirop@antonis-PC:/tmp$ more file2
TRUE
ClearCalendarScreenlet.py.log
ClearWeatherScreenlet.py.log
file1
file2
fileV94KIi
asidirop@antonis-PC:/tmp$
```

(τι είναι η true?)

16

- Η true είναι μια εντολή η οποία επιστρέφει πάντα 0 (true) ως κωδικό εξόδου και δεν εμφανίζει τίποτε.
- Αντίστοιχα υπάρχει και η εντολή false η οποία επιστρέφει ως κωδικό εξόδου πάντα 1 (false).

```
asidirop@antonis-PC:/tmp$ true
asidirop@antonis-PC:/tmp$ echo $?
0
asidirop@antonis-PC:/tmp$ which true
/bin/true
asidirop@antonis-PC:/tmp$ which false
/bin/false
asidirop@antonis-PC:/tmp$ false
asidirop@antonis-PC:/tmp$ echo $?
1
asidirop@antonis-PC:/tmp$
```


blocks σε if, while, until

17

- Αντιστοίχως και με την κανονική είσοδο.
- Από το παρακάτω παράδειγμα επίσης διαπιστώνουμε ότι δεν έχει δημιουργηθεί νέα διεργασία shell διότι οι μεταβλητές a και b ισχύουν και έξω από το block.

```
asidirop@antonis-PC:/tmp$ if true ; then read a;  
read b; fi < file2  
asidirop@antonis-PC:/tmp$ echo $a  
TRUE  
asidirop@antonis-PC:/tmp$ echo "$b"  
ClearCalendarScreenlet.py.log  
asidirop@antonis-PC:/tmp$
```

blocks εντολών

18

- Προσοχή χρειάζεται αν σε ένα block εντολών περιέχεται εντολή φίλτρο – η οποία θα διαβάσει από την κανονική είσοδο μέχρι το τέλος του αρχείου.
- Στο παράδειγμα, έχουμε αλλάξει την κανονική είσοδο για το block εντολών. Η 1^η όμως εντολή (grep) διαβάζει μέχρι το EOF. Άρα δεν «περίσσεψαν» δεδομένα στο stream για να διαβάσει η 2^η εντολή (wc) – γι' αυτό και μετράει 0.

```
asidirop@antonis-PC:/tmp$ ( grep 'Info'; wc) < file1
-rw-r--r-- 1 asidirop asidirop      0 2012-05-24 16:38
InfoPanelScreenlet.py.log
      0      0      0
asidirop@antonis-PC:/tmp$
```

η εντολή read (περισσότερα)

19

- Η εντολή read, εάν δεν καταφέρει να διαβάσει κανένα byte από την είσοδό της, τότε επιστρέφει exit code 1 (false). Στο παρακάτω παράδειγμα την βάλαμε να διαβάσει από ένα κενό αρχείο – άρα διάβασε κατευθείαν το EOF.

```
asidirop@antonis-PC:/tmp$ touch file3
asidirop@antonis-PC:/tmp$ read a < file3
asidirop@antonis-PC:/tmp$ echo $?
1
asidirop@antonis-PC:/tmp$
```

η εντολή read (περισσότερα)

20

- Θα μπορούσε να χρησιμοποιηθεί σε ένα while για να διαβάσει ολόκληρη την είσοδο.

Πατήσαμε ^D
– δηλαδή EOF

```
asidirop@antonis-PC:/tmp$ while read a;  
do echo "you typed $a"; done  
hello  
you typed hello  
testing  
you typed testing  
ta ta tatata  
you typed ta ta tatata  
asidirop@antonis-PC:/tmp$
```

η εντολή read (περισσότερα)

21

- Οποιαδήποτε εντολή περιέχεται μέσα στο script μας, κληρονομεί ως stdin το file4.
- Το file4 ανοίγει για ανάγνωση με την εκκίνηση του script και κλείνει με τον τερματισμό του.

```
asidirop@antonis-PC:/tmp$ ./reader.sh < file4
diavasa: line 1
diavasa: deyterh grammh
diavasa: third line
diavasa: grammh 4
asidirop@antonis-PC:/tmp$
```

file4

```
line 1
deyterh grammh
third line
grammh 4
```

reader.sh

```
#!/bin/sh
#file: reader.sh
while read a; do
    echo "diavasa: $a";
done
```

η εντολή read (περισσότερα)

22

- Αν θέλαμε να μετατρέψουμε το script έτσι ώστε να μην διαβάζει τα δεδομένα από την κανονική είσοδο αλλά από αρχείο το οποίο δίνουμε ως παράμετρο, θα έπρεπε να κάνουμε την ανακατεύθυνση μέσα στο script.
- Θα θέλαμε να εκτελούμε:
./reader.sh file4
και κάπου μέσα στο script θα πρέπει να συμβεί:
<“\$1”

η εντολή read (περισσότερα)

23

- η 1^η προσπάθεια μας οδηγεί σε ατέρμον βρόγχο. γιατί?

```
as@a:/tmp$ ./reader2.sh file4
diavasa: line 1
diavasa: line 1
diavasa: line 1
diavasa: line 1
^C
as@a:/tmp$
```

reader2.sh (1^η προσπάθεια)

```
#!/bin/bash

if (($#!=1)); then
    echo "Usage: $0 file" 1>&2
    exit 1
fi
file="$1"
if [ ! -f "$file" ]; then
    echo "$file not exists" 1>&2
    exit 1
fi
if [ ! -r "$file" ]; then
    echo "$file cannot read" 1>&2
    exit 1
fi
while read a < "$file" ; do
    echo "diavasa: $a";
done
```

η εντολή read (περισσότερα)

24

- διότι κάθε φορά που εκτελείται η εντολή:
`read a < "$file"`
είναι ανεξάρτητη από την προηγούμενη.
 - σε κάθε εκτέλεση, ανοίγει το αρχείο για ανάγνωση, διαβάζεται μια γραμμή, κλείνει το αρχείο.
 - Στην επόμενη εκτέλεση θα ξανανοίξει το αρχείο, άρα θα διαβαστεί πάλι η 1^η γραμμή.

```
asidirop@antonis-PC:/tmp$ read a < file4
asidirop@antonis-PC:/tmp$ echo "$a"
line 1
asidirop@antonis-PC:/tmp$ read a < file4
asidirop@antonis-PC:/tmp$ echo "$a"
line 1
asidirop@antonis-PC:/tmp$
```


η εντολή read (περισσότερα)

25

- Η ανακατεύθυνση πρέπει να γίνει στο block εντολών.
- έτσι το αρχείο θα ανοίξει για ανάγνωση με την είσοδο στο while, θα κλείσει με την έξοδο από το while.

```
asidirop@antonis-PC:/tmp$  
./reader2.sh file4  
diavasa: line 1  
diavasa: deyterh grammh  
diavasa: third line  
diavasa: grammh 4  
asidirop@antonis-PC:/tmp$
```

reader2.sh (σωστή προσπάθεια)

```
#!/bin/bash  
  
if (($#!=1)); then  
    echo "Usage: $0 file" 1>&2  
    exit 1  
fi  
file="$1"  
if [ ! -f "$file" ]; then  
    echo "$file not exists" 1>&2  
    exit 1  
fi  
if [ ! -r "$file" ]; then  
    echo "$file cannot read" 1>&2  
    exit 1  
fi  
while read a; do  
    echo "diavasa: $a";  
done < "$file"
```

blocks και διασωλήνωση

26

- Με τον ίδιο τρόπο θα μπορούσε να χρησιμοποιηθεί block εντολών σε μια διασωλήνωση - ουσιαστικά είναι σαν να υπήρχαν οι εντολές () μέσα σε ένα shell script.

```
asidirop@antonis-PC:/tmp$ ls | ( read a; read b ; echo  
"$a"; echo "$b" )  
ClearCalendarScreenlet.py.log  
ClearWeatherScreenlet.py.log  
asidirop@antonis-PC:/tmp$ ls | ( read a; read b ; echo  
$a; echo $b ) | wc  
      2      2      59  
asidirop@antonis-PC:/tmp$
```

παράδειγμα

27

- Δημιουργήστε ένα script το οποίο να υπολογίζει το συνολικό μέγεθος των txt αρχείων που υπάρχουν στον τρέχον φάκελο.
- 1^ο βήμα: ποια εντολή θα χρησιμοποιήσουμε για να βρούμε τα αρχεία και τα μεγέθη τους ?
 - `ls -l`
- 2^ο βήμα: πως θα εντοπίσουμε τα txt αρχεία?
 - `*.txt`
 - ή `grep` στην έξοδο της `ls`.
- έστω ότι έχουμε ως cwd τον φάκελο `/usr/share/vim/vim73/doc` που περιέχει αρκετά txt αρχεία.

```
asidirop@antonis-PC:/usr/share/vim/vim73/doc$ ls -l | grep '^-*\.txt$'
-rw-r--r-- 1 root root 11935 2011-03-24 09:10 arabic.txt
-rw-r--r-- 1 root root 54242 2011-03-24 09:10 autocmd.txt
```

παράδειγμα

28

- 3^ο βήμα: πως θα απομονώσουμε την στήλη που αφορά το μέγεθος του αρχείου?
 - υπάρχουν πολλοί τρόποι. ένας τρόπος είναι χρησιμοποιώντας την εντολή `cut` (περιγράφεται στα εργαστήρια)

```
asidirop@antonis-PC:/usr/share/vim/vim73/doc$ ls -l | grep '^-*\.txt$'
-rw-r--r-- 1 root root 11935 2011-03-24 09:10 arabic.txt
-rw-r--r-- 1 root root 54242 2011-03-24 09:10 autocmd.txt
```

29

- [illegible]

cut

30

- Με το όρισμα `-f` (fields) επιλέγει συγκεκριμένα πεδία. Τα πεδία χωρίζονται μεταξύ τους με το TAB, εκτός και αν δηλωθεί διαφορετικά με το `-d`.
- Το παρακάτω θα θεωρήσει διαχωριστή το space, και θα τυπώσει τα πεδία 1 και 2 από κάθε γραμμή.

```
asidirop@antonis-PC:/usr/share$ who
asidirop tty7          2012-05-24 16:37 (:0)
asidirop pts/0         2012-05-24 16:38 (:0)
asidirop pts/1         2012-05-24 22:43 (:0)
asidirop@antonis-PC:/usr/share$ who | cut -f1,2 -d' '
asidirop tty7
asidirop pts/0
asidirop pts/1
asidirop@antonis-PC:/usr/share$
```

cut

31

- Αν θέλουμε να τυπώσουμε όμως τα πεδία 1 και 3, θα διαπιστώσουμε ότι υπάρχει πρόβλημα. μόνο το 1^ο πεδίο εμφανίζεται.
- Αυτό διότι **κάθε** διαχωριστής ορίζει ένα νέο πεδίο. Δεν λαμβάνει δηλαδή υπόψη της τα πολλαπλά spaces.
- Στο παρακάτω παράδειγμα η ημερομηνία είναι (περίπου) το 11^ο πεδίο.

```
asidirop@antonis-PC:/usr/share$ who
asidirop tty7          2012-05-24 16:37 (:0)
asidirop pts/0         2012-05-24 16:38 (:0)
asidirop pts/1         2012-05-24 22:43 (:0)
asidirop@antonis-PC:/usr/share$ who | cut -f1,3 -d' '
asidirop
asidirop
asidirop
asidirop@antonis-PC:/usr/share$
```

cut

32

- Για να δουλέψουμε αποδοτικά με την cut θα πρέπει πρώτα να αφαιρέσουμε τις πολλαπλές εμφανίσεις του space. Αυτό γίνεται με την `tr -s ' '`
- Τα καταφέραμε !!!!

```
asidirop@antonis-PC:/usr/share$ who
asidirop tty7          2012-05-24 16:37 (:0)
asidirop pts/0         2012-05-24 16:38 (:0)
asidirop pts/1         2012-05-24 22:43 (:0)
asidirop@antonis-PC:/usr/share$ who | tr -s ' ' | cut -
d' ' -f1,3
asidirop 2012-05-24
asidirop 2012-05-24
asidirop 2012-05-24
asidirop@antonis-PC:/usr/share$
```


παράδειγμα

33

- 3^ο βήμα: πως θα απομονώσουμε την στήλη που αφορά το μέγεθος του αρχείου?
 - υπάρχουν πολλοί τρόποι. ένας τρόπος είναι χρησιμοποιώντας την εντολή `cut` (περιγράφεται στα εργαστήρια)
- Το παρακάτω θα τυπώσει μόνο την ζητούμενη στήλη (το μέγεθος των αρχείων)

```
asidirop@antonis-PC:/usr/share/vim/vim73/doc$ ls -l | grep '^-*\.txt$' |  
tr -s ' ' | cut -d' ' -f5  
11935  
54242  
68169  
44526  
5643  
7172
```

παράδειγμα

34

- 4^ο βήμα: πως θα αθροίσουμε?
 - μπορούμε να βρούμε μια εντολή που να κάνει αυτή τη δουλειά.
 - μπορούμε να τα αθροίσουμε με ένα while. αρκεί σε κάθε επανάληψη να διαβάζουμε σε μια μεταβλητή (a) ένα νούμερο.

```
asidirop@antonis-PC:/usr/share/vim/vim73/doc$ ls -l | grep '^-
.*\.txt$' | tr -s ' ' | cut -d' ' -f5 > /tmp/$$
asidirop@antonis-PC:/usr/share/vim/vim73/doc$ s=0
asidirop@antonis-PC:/usr/share/vim/vim73/doc$ while read a; do
((s+=a)); done < /tmp/$$
asidirop@antonis-PC:/usr/share/vim/vim73/doc$ echo "$s  bytes"
5143538  bytes
asidirop@antonis-PC:/usr/share/vim/vim73/doc$
```

παράδειγμα

35

- 5^ο βήμα: υλοποίηση script
- αποθηκεύουμε τα νούμερα σε ένα αρχείο, και μετά τα διαβάζουμε με την while read από το αρχείο.
- Θα μπορούσαμε να τα διαβάσουμε απ'ευθείας με διασωλήνωση?

FILE: counter

```
#!/bin/bash

s=0
ls -l | grep '^-*\.txt$' | tr -s ' ' | cut -d' ' -f5 > /tmp/$$
while read a; do
    ((s+=a))
done < /tmp/$$
echo "$s bytes"
```

παράδειγμα

36

- Ναι... Θα μπορούσαμε να τα διαβάσουμε από διασωλήνωση (pipe)
- Διαπιστώνουμε όμως ότι το παρακάτω μας εμφανίζει: “0 bytes” – γιατί?

FILE: counter

```
#!/bin/bash

s=0
ls -l | grep '^-*\.txt$' | tr -s ' ' | cut -d' ' -f5 | while read a;
do
    ((s+=a))
done
echo "$s bytes"
```

παράδειγμα

37

- Σε μια διασωλήνωση οι εντολές εκτελούνται παράλληλα!!!!
- όταν σε μια διασωλήνωση υπάρχει block εντολών, υποχρεωτικά το shell δημιουργεί μια νέα διεργασία (shell) που θα εκτελέσει αυτό το block.
- Νέα διεργασία → δεν έχουμε πρόσβαση στις μεταβλητές της.

FILE: counter

```
#!/bin/bash
```

```
s=0
```

```
ls -l | grep '^-*\.txt$' | tr -s ' ' | cut -d' ' -f5 | while read a; do  
    ((s+=a))
```

```
done
```

```
echo "$s bytes"
```

s=0

ls -l

grep

tr

cut

```
bash:  
while read a; do  
    ((s+=a))  
done
```

echo "\$s bytes"

παράδειγμα

38

- ΛΥΣΗ
- Η μεταβλητή *s* είναι τοπική στις ().
- Είναι σαν να έχουμε φτιάξει ένα script το οποίο διαβάσει νούμερα και τα αθροίζει.

FILE: counter

```
#!/bin/bash

ls -l | grep '^-*\.txt$' | tr -s ' ' | cut -d' ' -f5 | (
s=0
while read a; do
    ((s+=a))
done
echo "$s bytes"
)
```

ls -l

grep

tr

cut

```
bash:
s=0
while read a; do
    ((s+=a))
done
echo "$s"
```

διασωλήνωση

39

- όταν σε μια διασωλήνωση εμφανίζεται ένα block εντολών
 - είτε ()
 - είτε {}
 - είτε while, until, if
- τότε δημιουργείται μια νέα διεργασία shell που εκτελεί αυτό το block → άρα οι μεταβλητές είναι τοπικές.