Εισαγωγή στα Λειτουργικά Συστήματα

SET ΔΙΑΦΑΝΕΙΩΝ 9

IO STREAMS

ΑΝΤΩΝΗΣ ΣΙΔΗΡΟΠΟΥΛΟΣ

I/O streams

- Για κάθε διεργασία, αποθηκεύεται η πληροφορία ποια αρχεία έχει ανοιχτά η διεργασία
  - Στο unix αναφέρονται ως file descriptors
  - Στα Windows αναφέρονται ως file handlers
  - Για κάθε ανοιχτό αρχείο διατηρούνται διάφορες πληροφορίες. Σημαντική πληροφορία είναι η θέση του «κέρσορα» μέσα στο αρχείο.... δηλαδή το επόμενο read που θα κάνει η διεργασία από το αρχείο από ποια θέση του αρχείου θα δώσει δεδομένα?
  - όταν ένα πρόγραμμα ανοίγει ένα αρχείο (πχ για ανάγνωση) ο κέρσορας είναι στην θέση 0. εάν το πρόγραμμα διαβάσει (read) 5 bytes από το αρχείο, τότε ο κέρσορας μετακινείται 5 θέσεις. Την επόμενη φορά που θα κάνει read το πρόγραμμα θα διαβάσει τα επόμενα bytes του αρχείου.

I/O streams

- Μπορούμε να δούμε ποια αρχεία έχει ανοιχτά μια διεργασία (αρκεί να μας το επιτρέπουν οι άδειες)
- Στο παρακάτω παράδειγμα βλέπουμε τα αρχεία που έχει ανοιχτά η διεργασία που αντιστοιχεί τον πρόγραμμα “firefox” (στην πραγματικότητα είναι 10άδες στην περίπτωση του firefox)

```
asidirop@antonis-PC:~$ ls -l /proc/10569/fd
total 0
lr-x------ 1 asidirop asidirop 64 2012-04-26 19:14 0 -> /dev/null
lrwx------ 1 asidirop asidirop 64 2012-04-26 19:14 1 -> /home/asidirop/.xsession-errors
l-wx------ 1 asidirop asidirop 64 2012-04-26 19:14 10 -> pipe:[46911]
lrwx------ 1 asidirop asidirop 64 2012-04-26 19:14 101 -> socket:[3928988]
lrwx------ 1 asidirop asidirop 64 2012-04-26 19:14 102 -> socket:[637537]
lr-x------ 1 asidirop asidirop 64 2012-04-26 19:14 103 ->
/home/asidirop/.mozilla/firefox/9nb3c2bh.default/places.sqlite
lrwx------ 1 asidirop asidirop 64 2012-04-26 19:14 106 ->
/home/asidirop/.mozilla/firefox/9nb3c2bh.default/places.sqlite-wal
```

I/O streams

- Όλες οι διεργασίες έχουν εξ' ορισμού 3 ανοιχτά αρχεία.
  - input
  - output
  - error
- Αυτό ισχύει σχεδόν σε όλα τα Λειτουργικά συστήματα και προφανώς δεν εξαρτάται από την γλώσσα προγραμματισμού με την οποία δημιουργήθηκε το εκτελέσιμο που αντιστοιχεί στην διεργασία.

I/O streams

- Σε όλες τις γλώσσες προγραμματισμού υπάρχουν μεταβλητές που αντιστοιχούν σε αυτά τα κανάλια εισόδου-εξόδου:
  - C: stdin, stdout and stderr
  - C++: cin, cout, cerr
  - Perl: STDOUT, STDIN, STDERR
  - Java: System.in, System.out, System.err
  - .....

I/O streams

- Σε κάποιες γλώσσες, όταν παραλείπεται η αναφορά σε αρχείο εννοείται το stdin όταν πρόκειται για ανάγνωση και το stdout όταν πρόκειται για εγγραφή. Πχ:
  - C: εγγραφή:
  - `fprintf(stdout,"TEST\n");`
  - `printf("TEST\n");`
  - ανάγνωση:
  - `fscanf(stdin,"%s",tmp);`
  - `scanf("%s",tmp);`

I/O streams

- Ο “file descriptor” είναι ένας integer που μας δίνει την θέση του αρχείου στον πίνακα με τα ανοιχτά αρχεία της διεργασίας.
- Μια διεργασία έχει 3 ανοιχτά αρχεία τα οποία βρίσκονται στις θέσεις 0,1,2 του πίνακα ανοιχτών αρχείων:

Θέση	αρχείο
0	stdin
1	stdout
2	stderr

- Το λειτουργικό κρατάει διάφορες πληροφορίες για τα ανοιχτά αρχεία όπως:
- Θέση του κέρσορα στο αρχείο
- Mode κατάστασης (read, write, …)
- ….

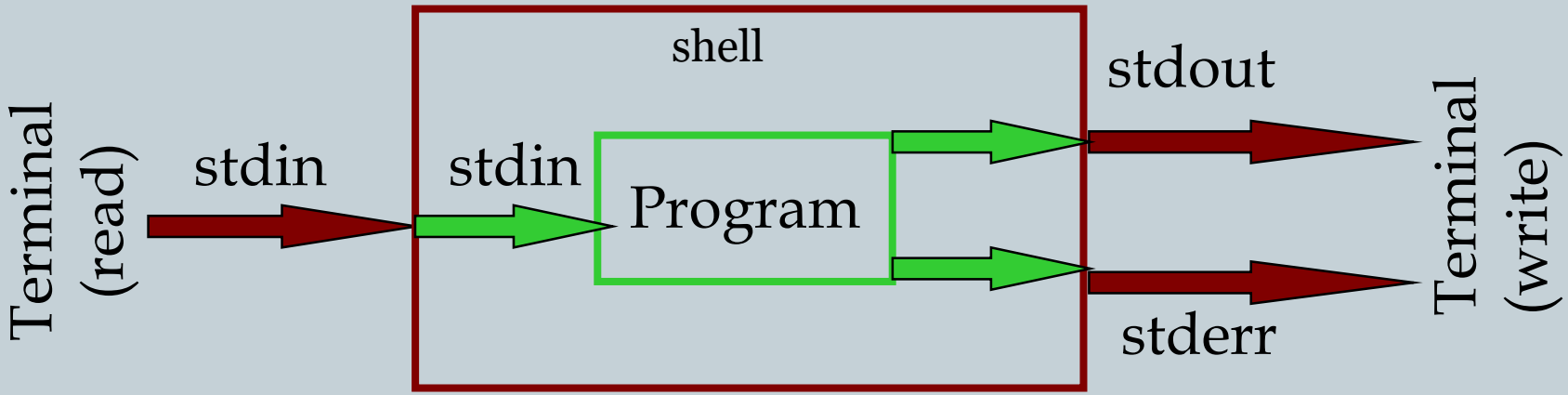
I/O streams

- Στην περίπτωση του shell (που εκτελείται στο παρακάτω παράδειγμα) αυτά τα 3 I/O streams αντιστοιχούν στο αρχείο που αντιστοιχεί στην συσκευή τερματικού

```
asidirop@aetos:~$ ps
  PID TTY          TIME CMD
 7218 pts/1    00:00:00 ps
22967 pts/1    00:00:00 bash
asidirop@aetos:~$ ls -l /proc/22967/fd
total 0
lrwx------ 1 asidirop conit 64 Apr 27 02:18 0 -> /dev/pts/1
lrwx------ 1 asidirop conit 64 Apr 27 02:18 1 -> /dev/pts/1
lrwx------ 1 asidirop conit 64 Apr 27 00:35 2 -> /dev/pts/1
```

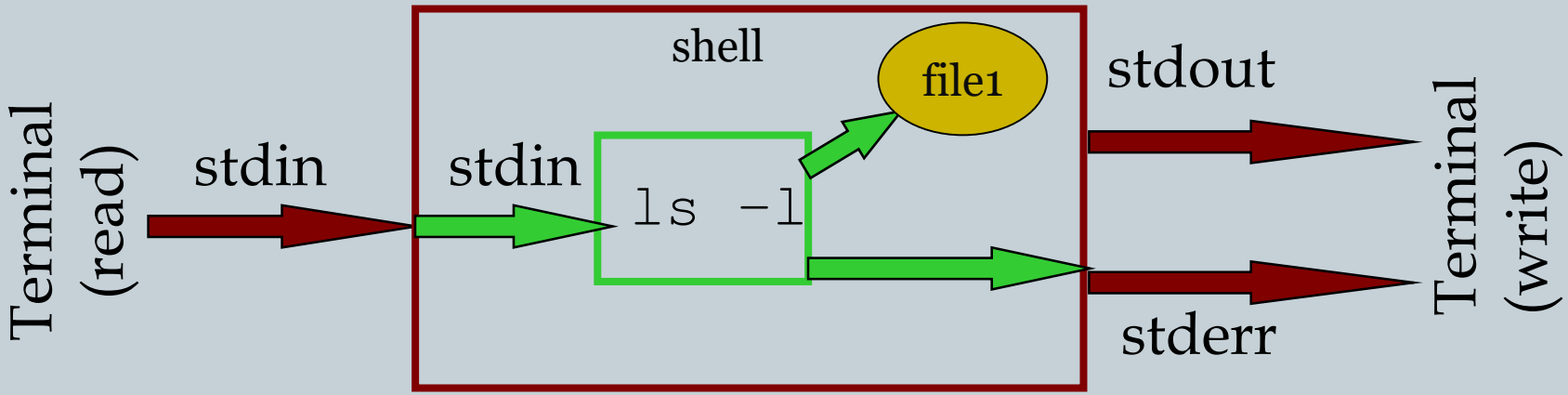
I/O streams

- Κάθε διεργασία κληρονομεί από την γονική της τον πίνακα ανοιχτών αρχείων (τις 3 πρώτες θέσεις με την exec(), όλο τον πίνακα με την fork())
- Ένα πρόγραμμα που εκτελείται από ένα shell, θα κληρονομήσει τα stdin, stdout, stderr του shell.



I/O streams

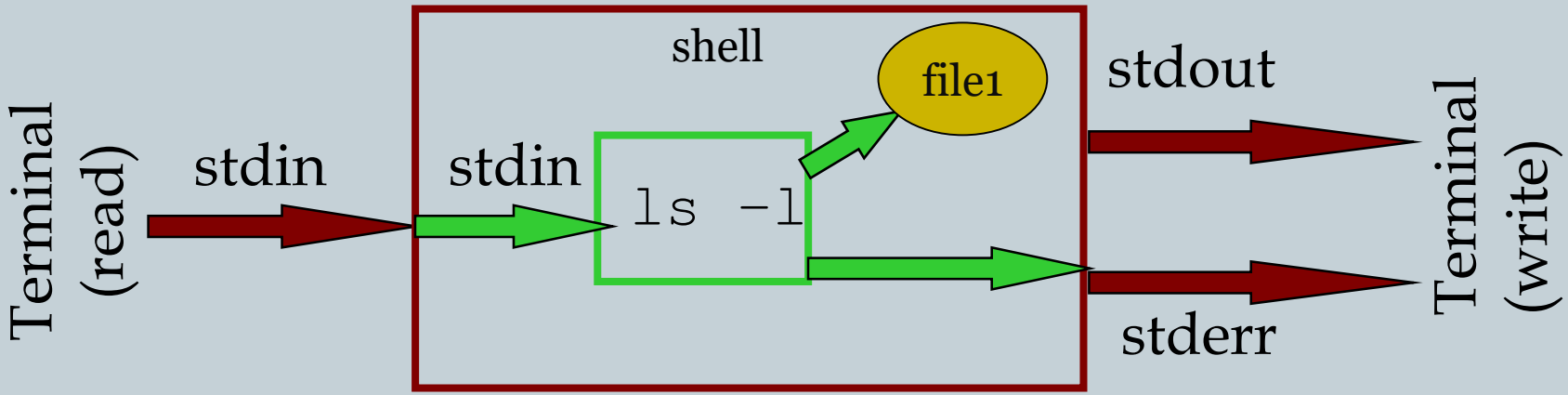
- Μπορούμε να δώσουμε «οδηγία» στο shell να αλλάξει τα κανάλια IO της διεργασίας που θα δημιουργήσει (του προγράμματος που θα τρέξουμε)
  - `ls -l > file1`
  - Η ls δεν θα εμφανίσει τίποτα στην οθόνη, αλλά η έξοδός της θα αποθηκευτεί στο αρχείο file1



I/O streams

- Η ls δεν θα εμφανίσει τίποτα στην οθόνη, αλλά η έξοδός της θα αποθηκευτεί στο αρχείο file1
- Αν όμως η ls εμφανίσει μήνυμα λάθους, τότε αυτό θα σταλεί στο “stderr” και άρα στην οθόνη.
- Πχ:

```
asidirop@aetos:~$ ls -l /mitsos > file1
ls: cannot access /mitsos: No such file or directory
asidirop@aetos:~$
```



Εκτύπωση λαθών

- Υπάρχει η γενική πολιτική: μέσα από ένα πρόγραμμα αν θέλουμε να τυπώσουμε ένα μήνυμα λάθους, τότε αυτό το τυπώνουμε στην έξοδο λαθών (stderr) και ΌΧΙ στην τυπική έξοδο (stdout)
- Αυτή η πολιτική πρέπει να ακολουθείται σε όλες τις περιπτώσεις.

Σύμβολα

>file, 1> file	Άλλαξε το stdout να «γράφει» στο αρχείο file και όχι στο προκαθορισμένο. Το file αν υπάρχει θα διαγραφεί, αν δεν υπάρχει θα δημιουργηθεί.
>>file, 1>> file	Άλλαξε το stdout να «γράφει» στο αρχείο file και όχι στο προκαθορισμένο. Το file αν υπάρχει ΔΕΝ θα διαγραφεί, αν δεν υπάρχει θα δημιουργηθεί. Τα δεδομένα από την εντολή θα γίνουν append στο file.
2> file	Άλλαξε το stderr να «γράφει» στο αρχείο file και όχι στο προκαθορισμένο. Το file αν υπάρχει θα διαγραφεί, αν δεν υπάρχει θα δημιουργηθεί.
2>> file	Άλλαξε το stderr να «γράφει» στο αρχείο file και όχι στο προκαθορισμένο. Το file αν υπάρχει ΔΕΝ θα διαγραφεί, αν δεν υπάρχει θα δημιουργηθεί. Τα δεδομένα από την εντολή θα γίνουν append στο file.
< file	Άλλαξε το stdin να «διαβάζει» από το αρχείο file και όχι από το προκαθορισμένο (τερματικό). Αν δεν υπάρχει το file τότε θα πάρουμε σφάλμα.
<< STRING	Άλλαξε το stdin να «διαβάζει» από το shell. Το shell θα περιμένει να πληκτρολογήσουμε δεδομένα μέχρι να πληκτρολογήσουμε την γραμμή STRING. Τότε, ό,τι πληκτρολογήσαμε θα το στείλει στο stdin της διεργασίας.

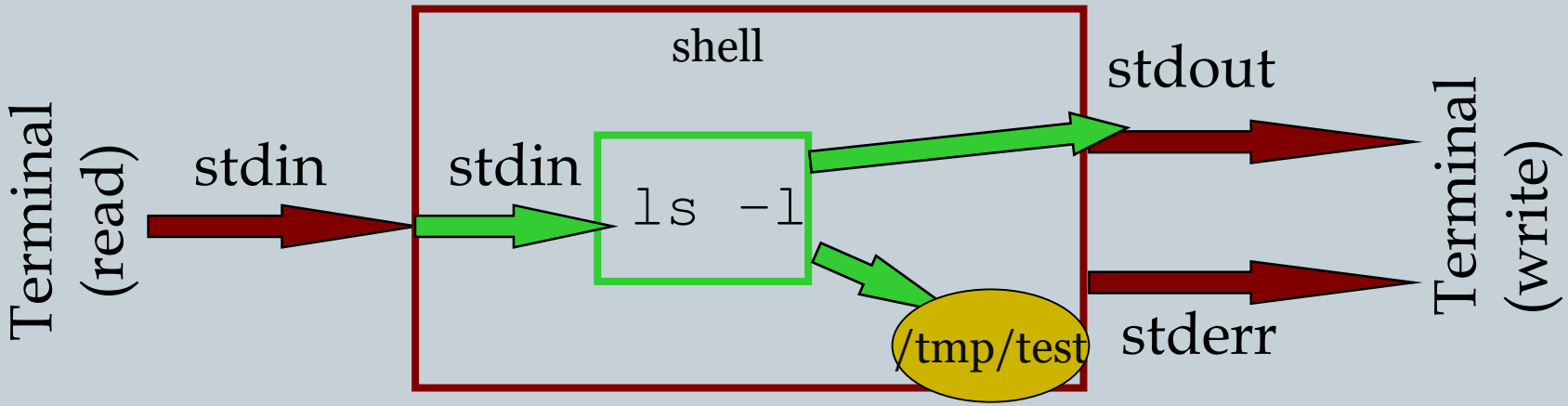
Σύμβολα

>file, 1> file	Άλλαξε το stdout να «γράφει» στο αρχείο file και όχι στο προκαθορισμένο. Το file αν υπάρχει θα διαγραφεί, αν δεν υπάρχει θα δημιουργηθεί.
>>file, 1>> file	Άλλαξε το stdout να «γράφει» στο αρχείο file και όχι στο προκαθορισμένο. Το file αν υπάρχει ΔΕΝ θα διαγραφεί, αν δεν υπάρχει θα δημιουργηθεί. Τα δεδομένα από την εντολή θα γίνουν append στο file.
2> file	Άλλαξε το stderr να «γράφει» στο αρχείο file και όχι στο προκαθορισμένο. Το file αν υπάρχει θα διαγραφεί, αν δεν υπάρχει θα δημιουργηθεί.
2>> file	Άλλαξε το stderr να «γράφει» στο αρχείο file και όχι στο προκαθορισμένο. Το file αν υπάρχει ΔΕΝ θα διαγραφεί, αν δεν υπάρχει θα δημιουργηθεί. Τα δεδομένα από την εντολή θα γίνουν append στο file.
< file	Άλλαξε το stdin να «διαβάζει» από το αρχείο file και όχι από το προκαθορισμένο (τερματικό). Αν δεν υπάρχει το file τότε θα πάρουμε σφάλμα.
<< STRING	Άλλαξε το stdin να «διαβάζει» από το shell. Το shell θα περιμένει να πληκτρολογήσουμε δεδομένα μέχρι να πληκτρολογήσουμε την γραμμή STRING. Τότε, ό,τι πληκτρολογήσαμε θα το στείλει στο stdin της διεργασίας.
1>&2	Στείλε την έξοδο από το 1 (stdout) στο 2 (stderr). Στην γενική περίπτωση x>&y (στείλε το x στο y)

I/O streams

- Η ls δεν θα εμφανίσει τίποτα στην οθόνη, αλλά η έξοδος λαθών θα αποθηκευτεί στο αρχείο /tmp/test
- Αν όμως η ls δεν εμφανίσει μήνυμα λάθους, τότε αυτό θα σταλεί στο “stdout” και άρα στην οθόνη.
- Πχ:

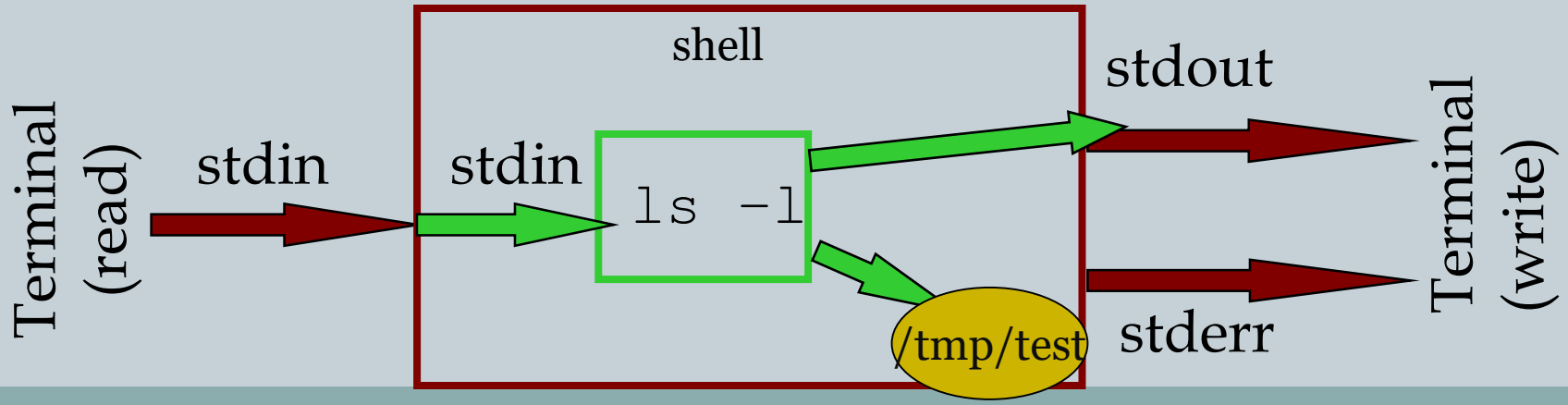
```
asidirop@aetos:~$ ls -l /mitsos 2> /tmp/test
asidirop@aetos:~$ cat /tmp/test
ls: cannot access /mitsos: Δεν υπάρχει τέτοιο αρχείο ή κατάλογος
```



I/O streams

- τα λάθη πάνε στο /tmp/test, η έξοδος στο τερματικό.

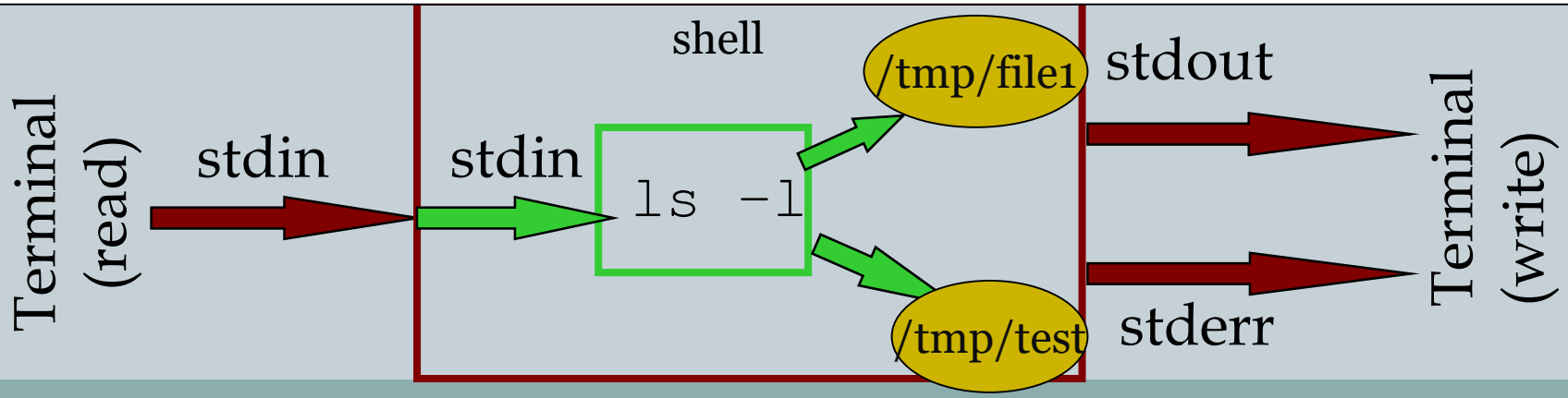
```
asidirop@aetos:~$ ls /tmp /mitsos 2> /tmp/test
/tmp:
20112xeim-adopse.tar.bz2  cake        filename     myid2     t
textVi.txt
arx                       fff         leleris.txt  myid9     test       too2l
arxeio2                   file1       ls           myidnik   test01     too3l
asd                       file1.txt   mc-antant    o         test.txt
ask3erg4                  file3       mybest       otinane   testvi1
asidirop@aetos:~$ cat /tmp/test
ls: cannot access /mitsos: Δεν υπάρχει τέτοιο αρχείο ή κατάλογος
```



I/O streams

- τα λάθη πάνε στο /tmp/test, η έξοδος στο /tmp/file1.

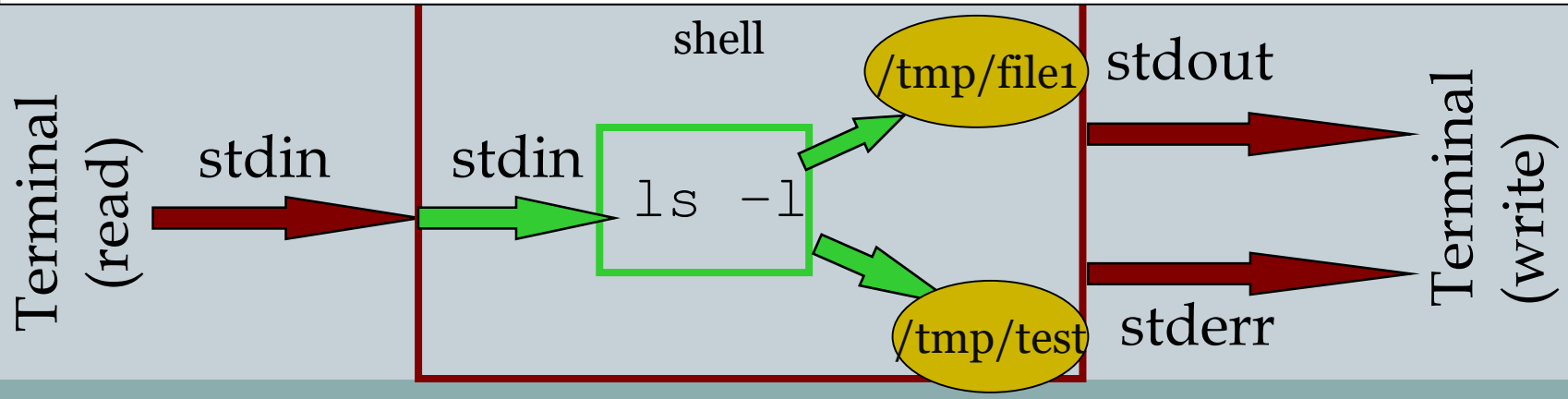
```
asidirop@aetos:~$ ls /tmp /mitsos 2> /tmp/test > /tmp/file1
asidirop@aetos:~$ cat /tmp/file1
/tmp:
20112xeim-adopse.tar.bz2  cake        filename     myid2     t
textVi.txt
arx                       fff         leleris.txt  myid9     test       too2l
arxeio2                   file1       ls           myidnik   test01     too3l
asd                       file1.txt   mc-antant    o         test.txt
ask3erg4                  file3       mybest       otinane   testvi1
asidirop@aetos:~$ cat /tmp/test
ls: cannot access /mitsos: Δεν υπάρχει τέτοιο αρχείο ή κατάλογος
```



I/O streams

- Το ίδιο αποτέλεσμα με πριν

```
asidirop@aetos:~$ ls /tmp /mitsos > /tmp/file1 2> /tmp/test
asidirop@aetos:~$ cat /tmp/file1
/tmp:
20112xeim-adopse.tar.bz2  cake        filename     myid2     t
textVi.txt
arx                       fff         leleris.txt  myid9     test       too2l
arxeio2                   file1       ls           myidnik   test01     too3l
asd                       file1.txt   mc-antant    o         test.txt
ask3erg4                  file3       mybest       otinane   testvi1
asidirop@aetos:~$ cat /tmp/test
ls: cannot access /mitsos: Δεν υπάρχει τέτοιο αρχείο ή κατάλογος
```

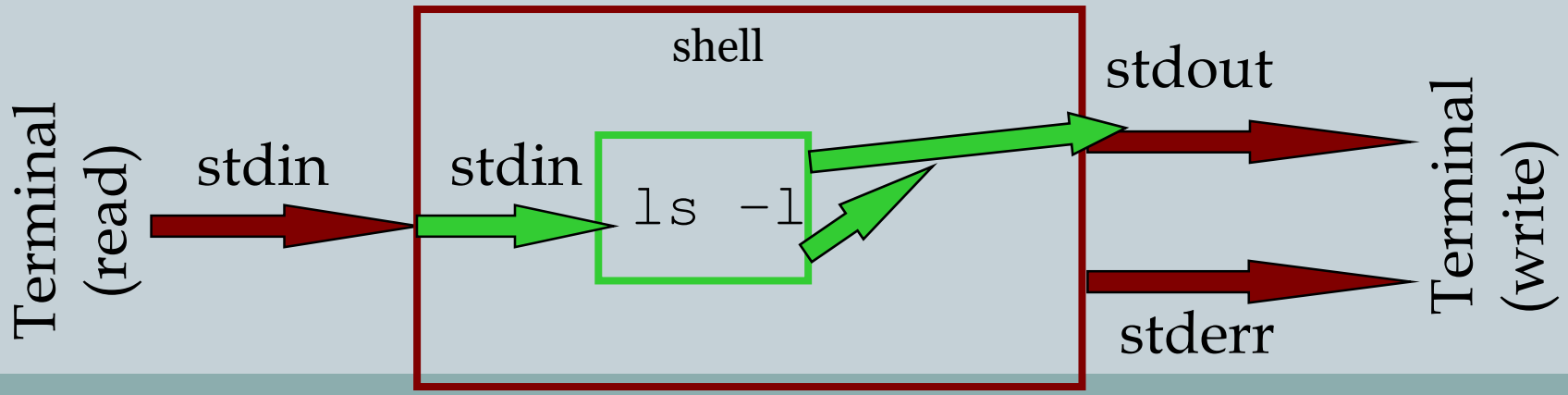


I/O streams

- Μπορούμε να κάνουμε ανακατεύθυνση ένα κανάλι σε ένα άλλο:
- Πχ:

```
bash-2.05a$ ls -l /mitsos 2>&1
Cannot access /mitsos: No such file or directory
bash-2.05a$
```

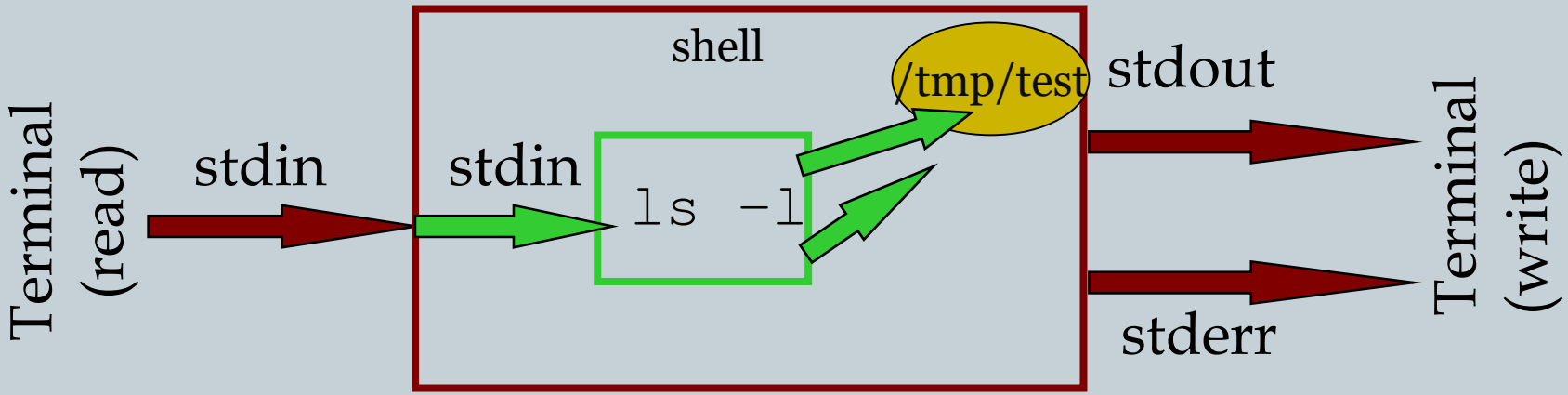
- Δεν είναι εμφανής η διαφορά



I/O streams

- και το out και το error θα πάνε στο αρχείο.

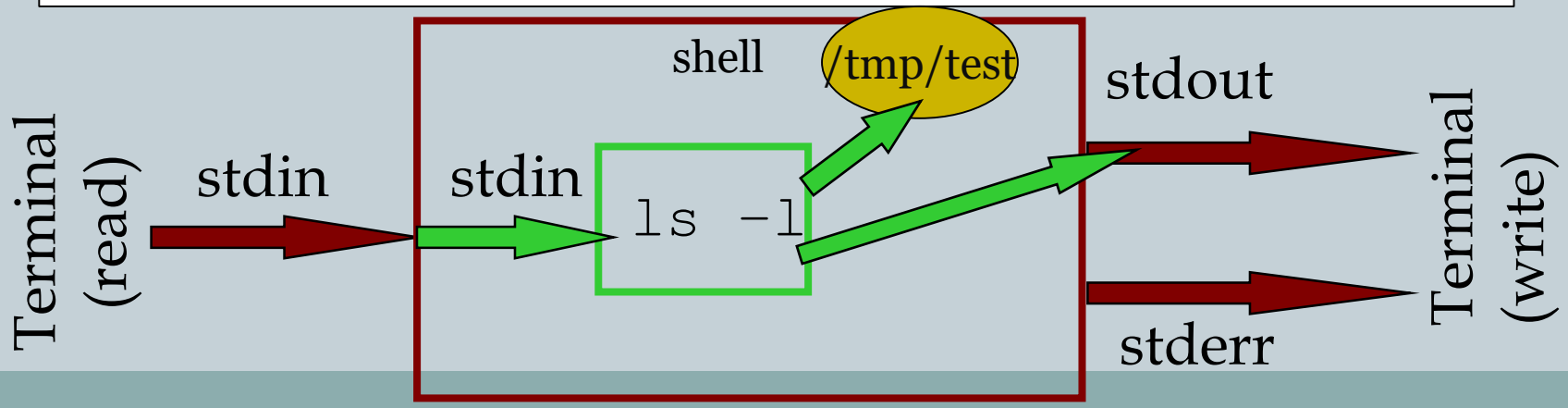
```
asidirop@aetos:~$ ls /tmp /mitsos > /tmp/test 2>&1
asidirop@aetos:~$ cat /tmp/test
ls: cannot access /mitsos: Δεν υπάρχει τέτοιο αρχείο ή
κατάλογος
/tmp:
20112xeim-adopse.tar.bz2
arx
arxeio2
```



I/O streams

- και το error θα πάει εκεί που έδειχνε αρχικά το out, και μετά ορίζεται ότι το out θα πάει στο αρχείο.

```
asidirop@aetos:~$  ls /tmp /mitsos  2>&1 > /tmp/test
ls: cannot access /mitsos: Δεν υπάρχει τέτοιο αρχείο ή
κατάλογος
asidirop@aetos:~$ cat /tmp/test
/tmp:
20112xeim-adopse.tar.bz2
arx
arxeio2
asd
```



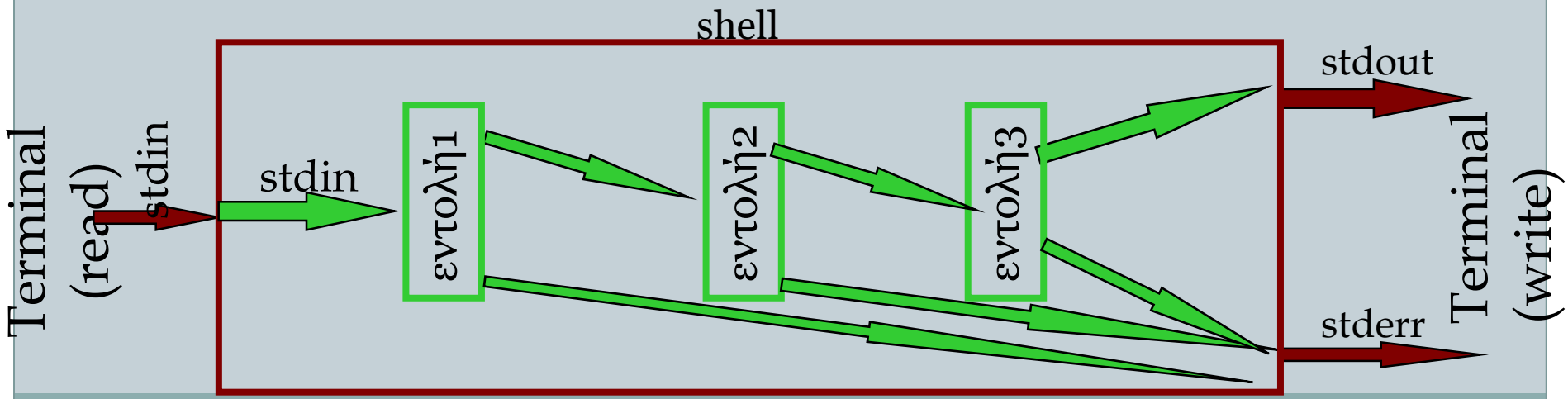
I/O streams

- Συμπέρασμα:
  - Η σειρά με την οποία ορίζουμε τις ανακατευθύνσεις καναλιών IO παίζει ρόλο στην περίπτωση που κάνουμε ανακατεύθυνση ένα κανάλι σε άλλο.

I/O streams - διασωλήνωση

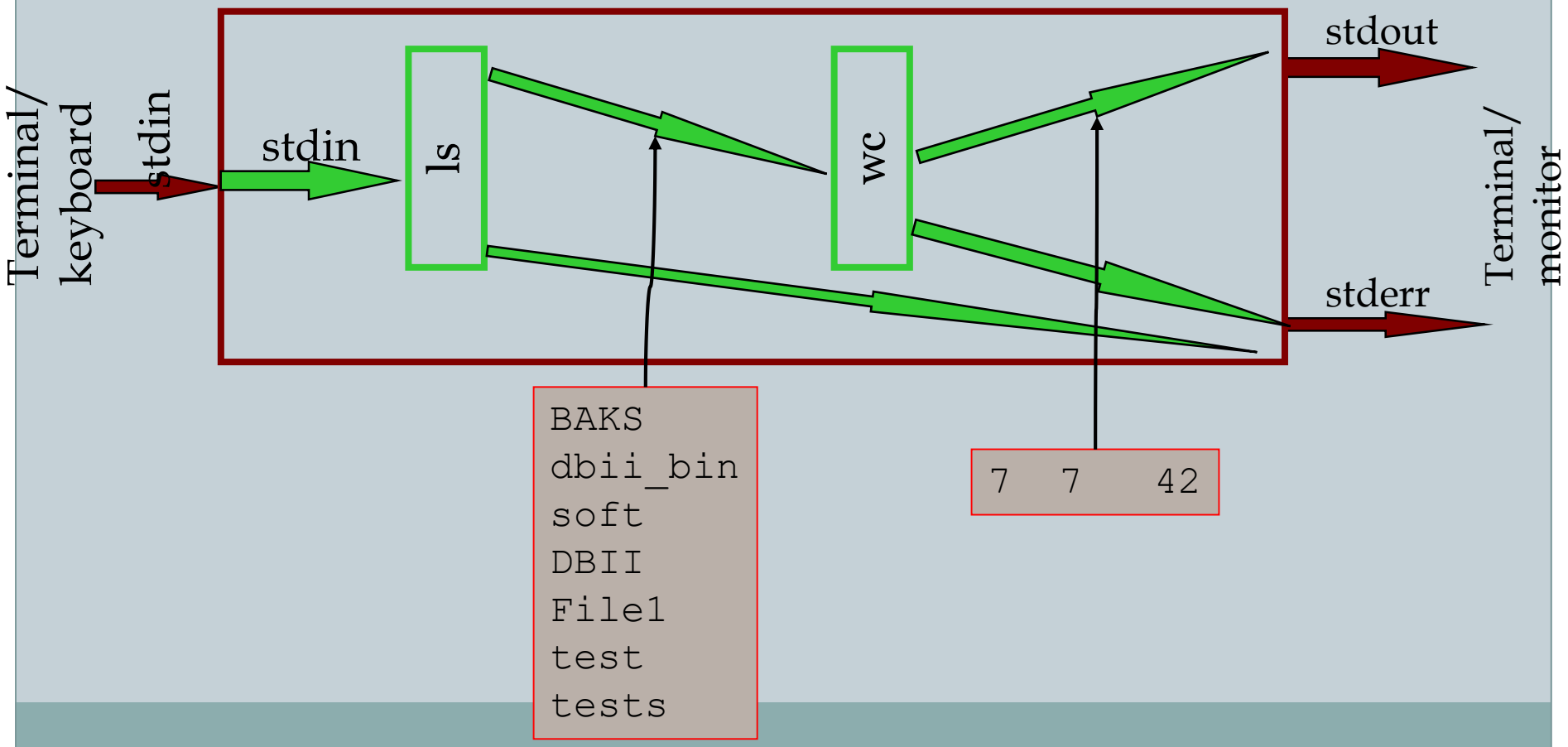
- Μπορούμε να κάνουμε ανακατεύθυνση ενός IO Stream μιας διεργασίας σε ένα IO Stream μιας άλλης διεργασίας
- με το ε1|ε2 ορίζεται ως είσοδος στην εντολή ε2 η έξοδος από την εντολή ε1
- Στην γενική περίπτωση μπορούμε να έχουμε πολλές εντολές στην σειρά.

```
εντολή1 | εντολή2 | εντολή3
```



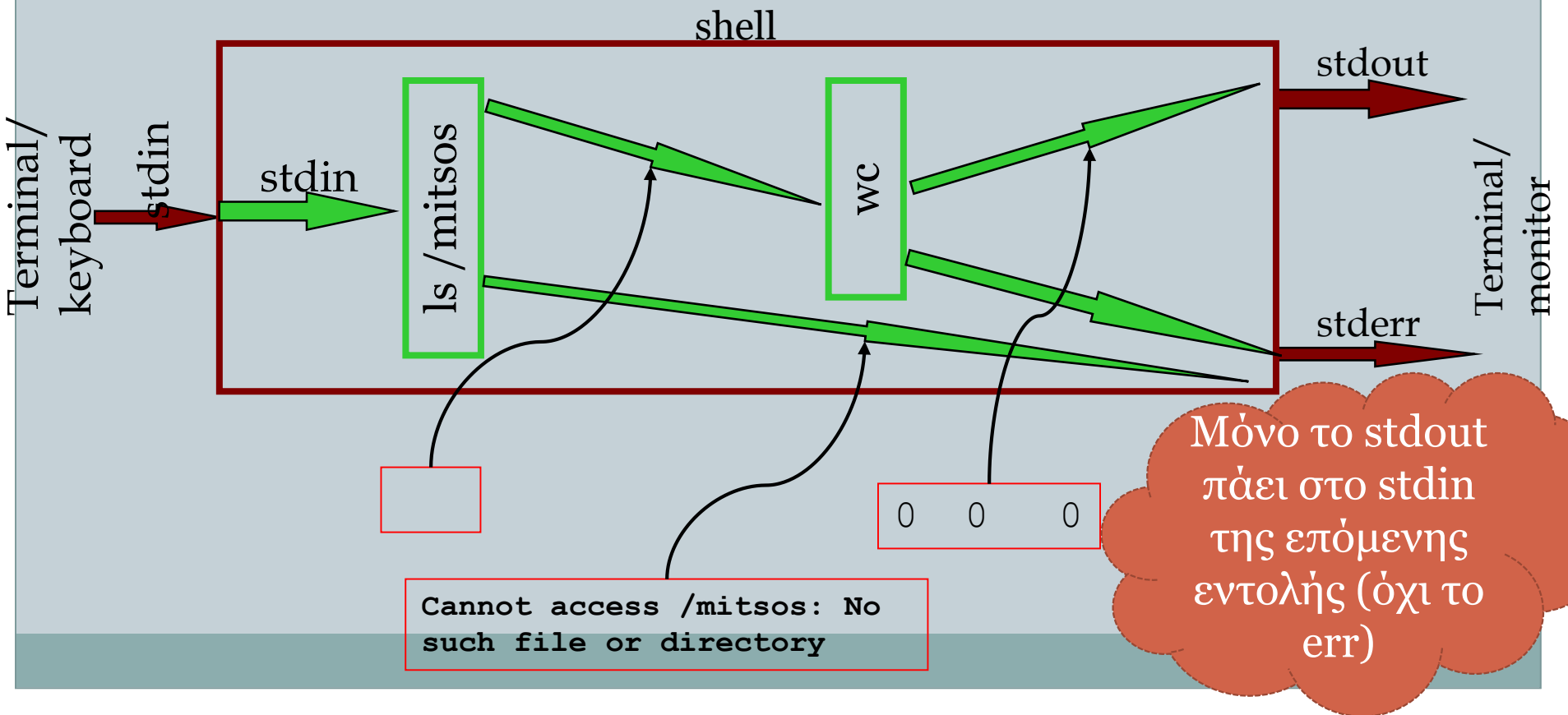
Είσοδος – έξοδος - διασωλήνωση


```
asidirop@aetos:~$ ls | wc
      7       7      42
asidirop@aetos:~$
```



Είσοδος – έξοδος - διασωλήνωση


```
bash-2.05a$ ls /mitsos | wc
Cannot access /mitsos: No such file or directory
                0               0               0
bash-2.05a$
```



Είσοδος – έξοδος - διασωλήνωση


```
bash-2.05a$ ls;ls -l
cc    list
total 16
-rw-r--r--    1 asidirop it           314 Jan 11  2003 cc
-rw-r--r--    1 asidirop it           183 Jan  8  2003 list
bash-2.05a$
```

```
bash-2.05a$ ls;ls -l > /tmp/test
cc    list
bash-2.05a$ cat /tmp/test
total 16
-rw-r--r--    1 asidirop it           314 Jan 11  2003 cc
-rw-r--r--    1 asidirop it           183 Jan  8  2003 list
bash-2.05a$
```

- Ο διαχωρισμός εντολών γίνεται με το <enter> ή το ;
- Κάθε εντολή είναι ανεξάρτητη από την προηγούμενη
- το ; έχει προτεραιότητα έναντι του >

Είσοδος – έξοδος - διασωλήνωση

Αν θέλουμε να ανακατευθύνουμε την έξοδο πολλών εντολών πρέπει να κάνουμε «ομαδοποίηση»

```
bash-2.05a$ (ls;ls -l) > /tmp/test
bash-2.05a$ cat /tmp/test
cc
list
total 16
-rw-r--r--    1 asidirop it             314 Jan 11  2003 cc
-rw-r--r--    1 asidirop it             183 Jan  8  2003 list
```

Με την ομαδοποίηση δημιουργείται ένα νέο shell που τρέχει τις εντολές που ερίσαμε στην παρένθεση.

Είσοδος – έξοδος - διασωλήνωση

- Αν θέλουμε να ανακατευθύνουμε την έξοδο πολλών εντολών σε διασωλήνωση, πρέπει να κάνουμε «ομαδοποίηση»

```
bash-2.05a$ ls ; ls -l |wc
cc    list
              3              20             127
bash-2.05a$
```

```
bash-2.05a$ (ls;ls -l ) | wc
              5              22             135
bash-2.05a$
```

- Με την ομαδοποίηση δημιουργείται ένα νέο shell που τρέχει τις εντολές της «ομάδας».

Είσοδος – έξοδος - διασωλήνωση

- Υπάρχει ένα ειδικό αρχείο στο οποίο μπορούμε να στείλουμε δεδομένα που θέλουμε να «πετάξουμε»

```
bash-2.05a$ ls -l > /dev/null
bash-2.05a$
```

- Το αρχείο αυτό «καταπίνει» όλα τα δεδομένα που αποθηκεύονται σε αυτό.
- Στο πάνω παράδειγμα δεν θα βρούμε ποτέ/πουθενά την έξοδο της εντολής - απλά θα χαθεί.

Είσοδος – έξοδος - διασωλήνωση

- Οι περισσότερες εντολές-φίλτρα διαβάζουν δεδομένα:
  - από αρχεία, αν δεχθούν ως ορίσματα ονόματα αρχείων
  - από την κανονική είσοδο αν δεν δοθεί ως όρισμα κανένα όνομα αρχείου δεδομένων

```
bash-2.05a$ wc file1
 1               2                 10   file1
bash-2.05a$ wc < file1
 1               2                 10
```

  - Στο παραπάνω παράδειγμα ποια είναι η διαφορά των 2 εκτελέσεων;

Είσοδος – έξοδος - διασωλήνωση


```
bash-2.05a$ wc file1
 1               2                 10   file1
bash-2.05a$ wc < file1
 1               2                 10
```

- Στην πρώτη περίπτωση έχουμε δώσει ως όρισμα στην wc το αρχείο file1. Η εντολή αναγνωρίζει το όρισμα και ανοίγει (open) το αρχείο για ανάγνωση. Διαβάζει τα δεδομένα και τυπώνει το αποτέλεσμα. Δεν διαβάζει τίποτε από την κανονική της είσοδο.
- Στην 2η περίπτωση δεν έχουμε δώσει όρισμα στην εντολή wc. Η εντολή θα διαβάσει τα δεδομένα από την κανονική της είσοδο. Έχουμε δώσει όμως την οδηγία στο shell να θέσει ως κανονική είσοδο στην εντολή το αρχείο file1. Άρα πρακτικά, πάλι τα δεδομένα θα διαβαστούν από το αρχείο file1, όμως η εντολή δεν γνωρίζει από πού προέρχονται τα δεδομένα (γι' αυτό και δεν εμφανίζει το όνομα αρχείου).

Είσοδος – έξοδος - διασωλήνωση

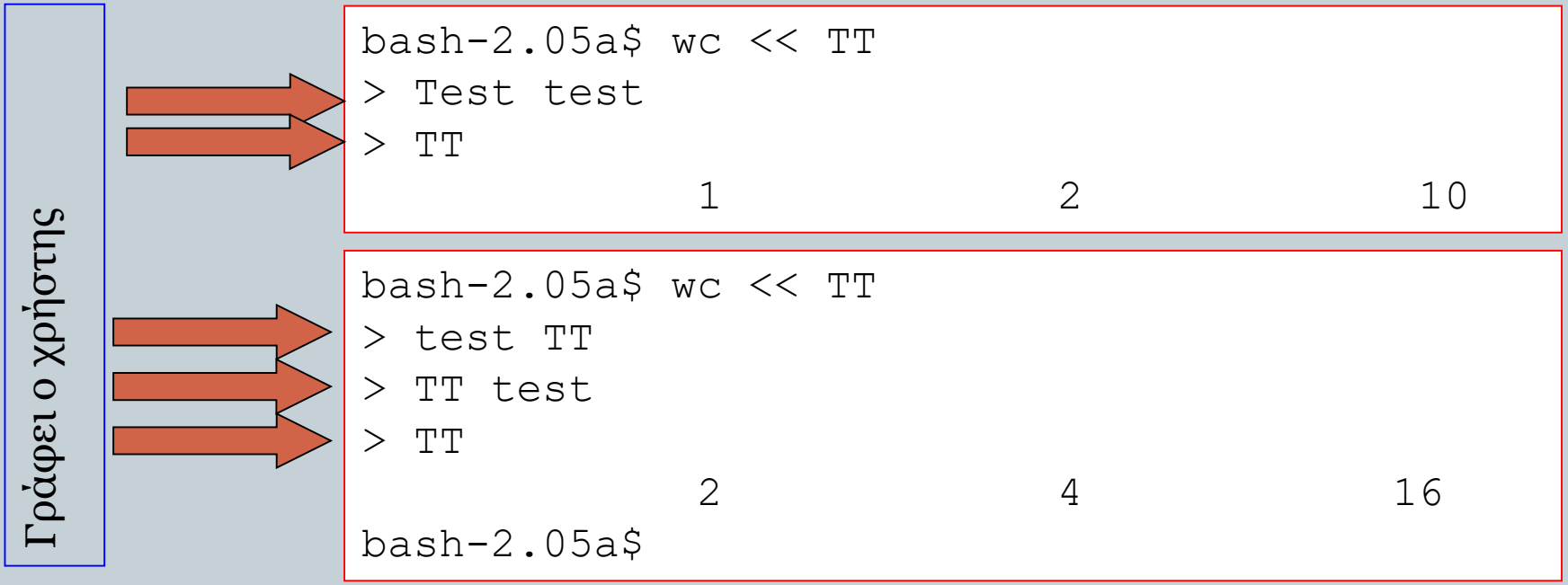
- Μια εντολή που διαβάζει από την είσοδό της δεδομένα, σταματάει όταν διαβάσει τον ειδικό χαρακτήρα EOF (End of File). Στο UNIX το EOF συμβολίζεται με το ^D (Cntrl-D) και μπορούμε να το στείλουμε από το πληκτρολόγιο σε μια διεργασία.

```
bash-2.05a$ wc
Test test
^D
               1               2              10
```

Είσοδος – έξοδος - διασωλήνωση

Εντολή << string

Το ειδικό σύμβολο << δίνει την οδηγία στο shell να διαβάσει από την είσοδό του, και αυτά που θα διαβάσει να τα στείλει σαν είσοδο στην εντολή. Όταν διαβάσει μια γραμμή που αποτελείται μόνο από το string, τότε θα στείλει στην εντολή τον χαρακτήρα EOF, και η εντολή θα σταματήσει να διαβάσει.



shell

- **Το shell είναι ένα πρόγραμμα που διαβάζει δεδομένα από το stdin, τα οποία τα θεωρεί εντολές και τις εκτελεί.**
- Αν δεν ορίσουμε διαφορετικά το stdin ενός shell είναι το τερματικό
- Μπορούμε όμως να δώσουμε είσοδο σε ένα shell από αρχείο, όπως σε κάθε άλλο πρόγραμμα.

```
bash-2.05a$ cat file1
ls -l
whoami
date
bash-2.05a$ sh < file1
total 24
-rw-r--r--    1 asidirop it          314 Jan 11  2003 cc
-rw-r--r--    1 asidirop it           18 Nov 20 15:22 file1
-rw-r--r--    1 asidirop it          183 Jan  8  2003 list
asidirop
Mon Nov 20 15:23:05 EET 2006
bash-2.05a$
```

Είναι μια κακή μέθοδος για να εκτελέσουμε εντολές από αρχείο....!!!!!

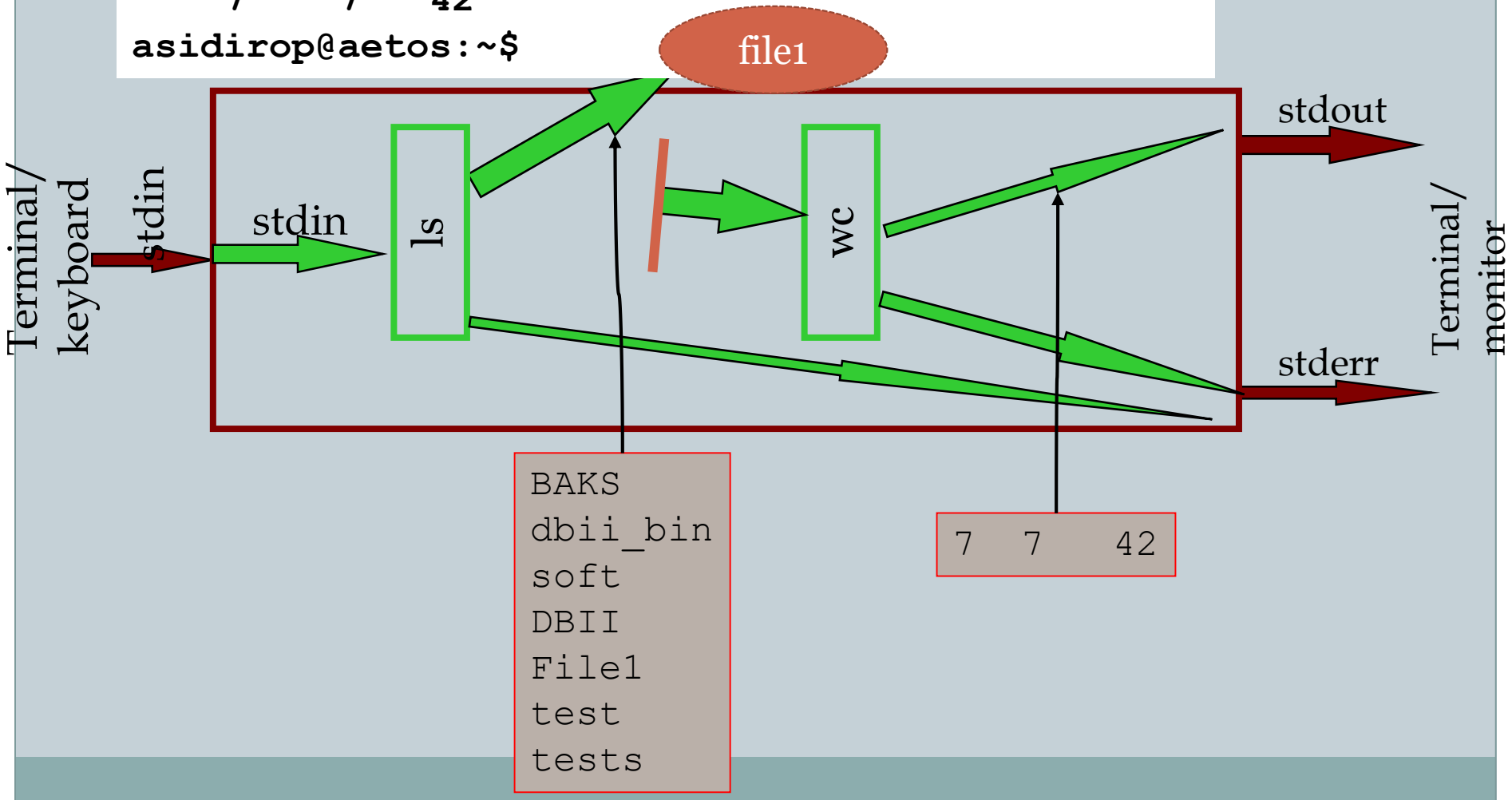
Συνηθισμένα σφάλματα

- `ls -l > file1 > file2` ΛΑΘΟΣ
  - Στο sh δεν μπορούμε να στείλουμε ένα κανάλι σε 2 μέρη. Πρακτικά η έξοδος της εντολής θα αποθηκευτεί στο αρχείο file2.
- `ls -l > file1 | wc` ΛΑΘΟΣ
  - Το ίδιο σφάλμα με την προηγούμενη περίπτωση. δεν μπορούμε να στείλουμε την έξοδο μιας εντολής σε 2 σημεία (και στο αρχείο file1 αλλά και στο pipe προς την εντολή wc). Πρακτικά η εντολή wc δεν θα διαβάσει τίποτε και τα αποτελέσματα της ls θα αποθηκευτούν στο αρχείο.

- Υπάρχουν άλλα shells (όπως το zsh) που επιτρέπουν διακλάδωση ενός I/O stream (πχ: ls > file1 > file2).
- Εάν θέλουμε να "στείλουμε" ένα κανάλι σε 2 μέρη χρησιμοποιώντας το sh, τότε πρέπει να χρησιμοποιήσουμε την εντολή tee.

Περίπτωση λάθους


```
asidirop@aetos:~$ ls > file1 | wc
      7       7      42
asidirop@aetos:~$
```



Συνηθισμένα σφάλματα

- `grep 'xyz' /etc/passwd | wc /etc/passwd`

  ΛΑΘΟΣ

  - ζητάμε από την grep να ψάξει κάτι στο αρχείο /etc/passwd. Το αποτέλεσμα της αναζήτησης θα τυπωθεί στην κανονική της έξοδο. Στέλνουμε την έξοδο της grep ως είσοδο στην wc για να μετρήσουμε τις λέξεις. Στην wc όμως δώσαμε όρισμα όνομα αρχείου, άρα δεν θα διαβάσει από την κανονική της είσοδο, θα διαβάσει δεδομένα από το αρχείο /etc/passwd.
  - Τα δεδομένα που έστειλε η grep θα χαθούν, ενώ η wc θα μετρήσει ΟΛΕΣ τις γραμμές/λέξεις του αρχείου /etc/passwd και όχι μόνο αυτές που ταίριαξαν στην Κ.Ε.