



Διάλεξη 10: Προχωρημένος Προγραμματισμός Κελύφους & Παραδείγματα

Δημήτρης Ζεϊναλιπούρ



Περιεχόμενο Διάλεξης

Προχωρημένος Προγραμματισμός

- Αποσφαλμάτωση (set)
- Διαχείριση Σημάτων
(Signals και trap)
- Διοχετεύσεις & Συσκευές
(η εντολή exec)



Παραδείγματα Προγραμμάτων

Αποσφαλμάτωση Προγραμμάτων



- Για την αποσφαλμάτωση μπορεί να χρησιμοποιηθούν οι ακόλουθοι τρόποι
 - **echo/printf** : Εκτύπωση Απλών Μηνυμάτων
 - **set -xv debug on** **set +xv: debug off**
- Το μειονέκτημα του echo/printf, είναι ότι δεν μπορεί να δείξει εύκολα την ροή του προγράμματος.
- Επίσης δεν μπορούμε εύκολα να κάνουμε μετάβαση από τον υπό ανάπτυξη κώδικα στον τελικό κώδικα

Αποσφαλμάτωση με την “set”



- Το “set” είναι μια built-in εντολή του κελύφους
- Έχει επιλογές, οι οποίες μας επιτρέπουν να ελέγχουμε την ροή της εκτέλεσης
 - **-v** Εκτύπωση εντολών όπως είναι στο script γραμμή-γραμμή
 - **-x** Εκτύπωση τιμής μεταβλητών γραμμή-γραμμή
- Πριν να εκδώσετε την τελική έκδοση του script σας θέσετε το debugging off
 - Debugging on: **set -xv**
 - Debugging off : **set +xv**
- Αυτές οι επιλογές μπορούν επίσης να τεθούν μέσω του **she-bang** (στην αρχή του script)
 - **#! /bin/bash -xv**

Αποσφαλμάτωση με την “set”



```
$ cat ZipCode
#!/bin/bash
```

```
set -xv    # Debugging On
```

```
read -p "Enter Zip code " Zip
echo "Zip Code is: " $Zip
readonly Zip
read -p "Attempting to change Zip Code: "
Zip
echo "Zip Code is: " $Zip
```

```
set +xv    # Debugging Off
```

-v : Εκτύπωση εντολών όπως εκτελούνται (με κόκκινο)

-x : Δείχνει με + πως εκτελείται κάθε εντολή και με τι τιμή στις μεταβλητές εισόδου

Εκτέλεση του Script

```
$ ZipCode
read -p "Enter Zip code " Zip
+ read -p 'Enter Zip code ' Zip
Enter Zip code 60563
```

```
echo "Zip Code is: " $Zip
+ echo 'Zip Code is: ' 60563
Zip Code is: 60563
```

```
readonly Zip
+ readonly Zip
```

```
read -p "Attempting to change Zip Code: " Zip
+ read -p 'Attempting to change Zip Code: ' Zip
Attempting to change Zip Code: 60115
./ZipCode: Zip: readonly variable
```

```
echo "Zip Code is: " $Zip
+ echo 'Zip Code is: ' 60563
Zip Code is: 60563
```

```
set +xv
+ set +xv
```

Error message

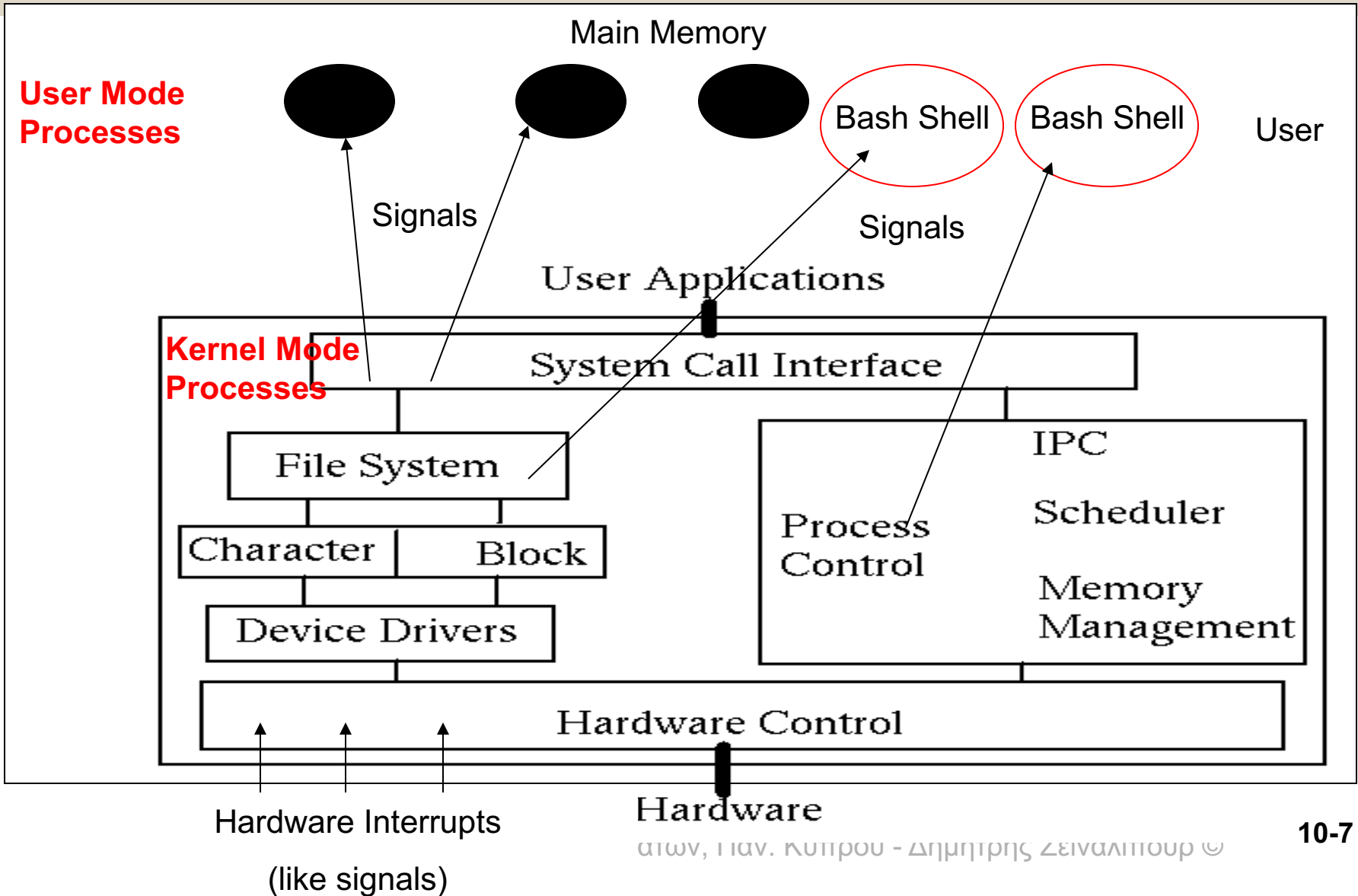


Διαχείριση Σημάτων (Signals)

- **Signals:** Μικρά μηνύματα, τα οποία στέλνονται σε μια **καθορισμένη** διεργασία ή ομάδες διεργασιών.
- Τα σήματα στέλνονται μεταξύ Διεργασιών (μέσω του πυρήνα, εάν υπάρχουν τα κατάλληλα δικαιώματα) ή από τον Πυρήνα στην Διεργασία.
- Όταν λάβει το signal μια διεργασία, **διακόπτει άμεσα την εκτέλεση της**, και διαχειρίζεται το signal.
 - Μπορεί να αγνοήσει το σήμα (όχι όλα)
 - Μπορεί να το χειριστεί με ένα signal handler.
- Εδώ θα μελετήσουμε τα σήματα στο πλαίσιο του κελύφους αλλά θα τα διαχειριστούμε και μέσω προγραμμάτων C, στην συνέχεια του μαθήματος.



Διαχείριση Σημάτων (Signals)





Διαχείριση Σημάτων (Signals)

- Παράδειγμα Σήματος Διεργασία ⇔ Διεργασία

Καθώς εκτελείται το πιο κάτω πρόγραμμα στο κέλυφος πληκτρολογείτε **Ctrl-C**, τότε στέλνετε το **INT** signal (**SIGINT, #2**), το οποίο αντιστοιχεί στο σήμα Terminal Interrupt (δηλαδή έξοδος του προγράμματος)

```
while true; do  
    echo "Still Alive"  
    sleep 3 # second  
done
```

Το πρόγραμμα διακόπτει
την εκτέλεση του



Διαχείριση Σημάτων

- Στο κέλυφος (και στο UNIX γενικότερα) μπορούμε να στέλνουμε σήματα σε διεργασίες με διάφορους τρόπους
- Παράδειγμα με την εντολή KILL
 - `kill -INT 1234` # αποστολή σήματος INT στο processID#1234
 - `kill -KILL 1235`
 - `kill -9 1236` # αποστολή σήματος KILL στο processID#1236
 - Ένα bash script μπορεί να διαχειριστεί εισερχόμενα σήματα με την εντολή **trap**
 - Εξαίρεση Αποτελεί το “KILL” ή “9” το οποίο δεν μπορούμε να το διαχειριστούμε και οδηγεί πάντα σε τερματισμό μιας διεργασίας.

- Σύνταξη:

```
trap 'εντολή' signals
```

- Παράδειγμα:

```
trap 'echo "Unable to Interrupt"' 1 2
```

μήνυμα

σήματος



Παράδειγμα

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

```
trap 'echo "Unable to Interrupt"' 2
```

```
while [ true ]
```

```
do
```

```
    echo "Try to press Ctrl-C"
```

```
    sleep 1
```

```
done
```

Αποτέλεσμα Εκτέλεσης

```
$run
```

```
Try to press Ctrl-C
```

```
Try to press Ctrl-C
```

```
Ctrl-C
```

```
Unable to Interrupt
```

Πίνακας Σημάτων Κελύφους



POSIX Compliant Σήματα

#0 On exit from Terminal

`HUP' #1. Hangup (terminal line got hung up), στέλνεται στις διεργασίες με την έξοδο μας από ένα σύστημα!

nohup - run a command immune to hangups!

`INT' #2. Terminal interrupt. Ctrl-C (terminate)

`QUIT' #3. Terminal quit. (Ctrl-| , terminate and core dump)

`ABRT' #6. Process abort. Ctrl-\ (παρόμοιο με την Ctrl-C – γράφει το memory image σε ένα αρχείο core)

`KILL' #9. Kill (not caught or ignored).

`ALRM' #14. Alarm Clock.

(Στην C μπορούμε να καλέσουμε την εντολή `alarm(seconds) - <unistd.h>` , με το οποία ο πυρήνας θα στέλνει κάθε `seconds` ένα `Signal#14` στο process)

`TERM' #15. Termination – terminate "politely", allow time to cleanup(), e.g., JAVA .finalize().

Πίνακας Σημάτων Κελύφους



Μερικά Επιπλέον Σήματα

(με διαφορετικό αριθμό σε διαφορετικές υλοποιήσεις του Unix

- `BUS`** Access to an undefined portion of a memory object.
- `CHLD`** Child process terminated, stopped, or continued.
- `CONT`** Continue executing, if stopped. (όπως η εντολή `fg - foreground`)
- `FPE`** Erroneous arithmetic operation.
- `ILL`** Illegal Instruction.
- `PIPE`** Write on a pipe with no one to read it.
- `SEGV`** Invalid memory reference. (Segmentation Violation – Core Dumped – π.χ. σε ένα πρόγραμμα C κάνετε πρόσβαση σε κάποια λάθος διεύθυνση μνήμης).
Αυτό το signal προέρχεται από τον πυρήνα του ΛΣ.
- `STOP`** Stop executing (cannot be caught or ignored).
- `TSTP`** Terminal stop. (Ctrl-Z) – can be ignored !
- `TTIN`** Background process attempting read.
- `TTOU`** Background process attempting write.



Παράδειγμα με την trap

```
#!/bin/bash
trap 'cleanup; exit' 2 3
cleanup () {
    /bin/rm -f /tmp/tempfile.$$.*
}
```

```
echo "process id: $$"
for i in 1 2 3 4 5 6 7 8
do
```

```
    echo "$i.iteration"
```

```
    # Δημιούργησε το (κενό) Αρχείο /tmp/tempfile.ProcessID.$i
```

```
    touch /tmp/tempfile.$$.$i
```

```
    sleep 1
```

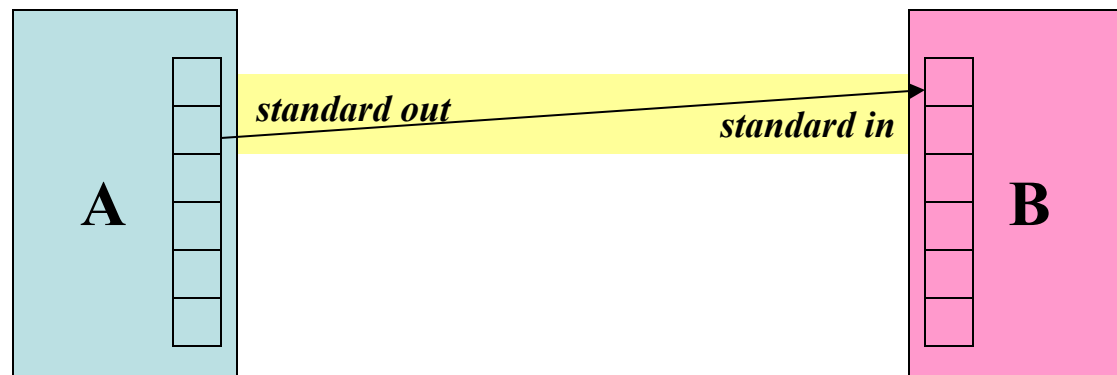
```
done
cleanup
```

Το πρόγραμμα δημιουργεί 8 αρχεία και σβήνει αυτά τα αρχεία οπότε εμείς στείλουμε το σήμα Ctrl-C (**#2-Terminal Interrupt**) ή όταν τελειώσει το κέλυφος (**#3 – Terminal Quit**)

Διοχετεύσεις και Συσκευές



- Διοχέτευση (Pipe |) : Τα δεδομένα εξόδου του A είναι τα δεδομένα εισόδου του B.



- Και τα δυο προγράμματα εκτελούνται παράλληλα στην μνήμη και να αποτελέσματα είναι buffered στην μνήμη (σε ενδιάμεσο χώρο στον πυρήνα).



Υλοποίηση Διοχετεύσεων με δυο μεταβλητές

- Μέχρι τώρα το pipelining γινόταν σε συναρτήσεις που έπαιρναν 1 παράμετρο.
- π.χ. **sort info.txt | uniq**
- Τι γίνεται εάν θέλουμε να χρησιμοποιήσουμε μια εντολή η οποία παίρνει δυο παραμέτρους; π.χ. **diff file1 file2**, συγκρίνει δυο αρχεία και εκτυπώνει τις διαφορές τους
- Λύση: Θα χρησιμοποιήσουμε το «-»

sort info.txt | diff - info.txt

Δηλαδή,

Αυτό είναι το standard input της εντολής diff

sort info.txt | diff /dev/fd/0 info.txt



Διοχετεύσεις - Internals

- **Τι γίνεται εάν μια διεργασία B χρειάζεται να διαβάσει από μια διοχέτευση αλλά δεν υπάρχει κάτι διαθέσιμο για ανάγνωση, π.χ., `cat | more` ;**
 - Το UNIX θα βάλει τον reader B σε sleep μέχρι τα δεδομένα να είναι διαθέσιμα (από τον A)
- **Τι γίνεται εάν μια διεργασία B δεν μπορεί να ανταποκριθεί στον ρυθμό με τον οποίο γράφει ο A στην διοχέτευση;**

Απάντηση : Επόμενη Διαφάνεια



Διοχετεύσεις - Internals

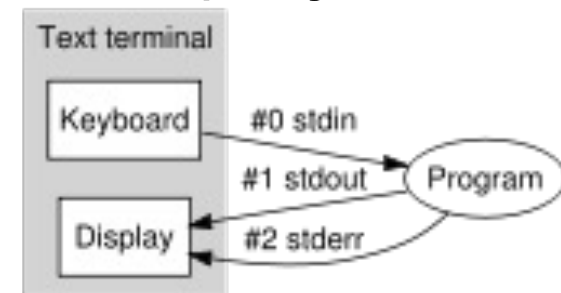
- Για να λύσει το πρόβλημα του μεταβλητού ρυθμού γραφής/ανάγνωσης μεταξύ διεργασιών, το UNIX διατηρεί buffers (στον πυρήνα, στη RAM) από τα unread δεδομένα μεταξύ piped διεργασιών.
 - Αυτό αναφέρεται σαν το pipe size
 - Εάν το **pipe γεμίσει**, το UNIX θα **θέσει τον writer σε sleep** μέχρι ο **reader να ελευθερώσει** κάποιο χώρο (ουσιαστικά κάνοντας **read από το pipe**) “**ulimit -a**”
- Με τις διοχετεύσεις, μπορούμε να έχουμε πολλαπλούς readers και writers να εκτελούνται ψευδό-παράλληλα! (το **stdout_A => stdin_B**



Standard in/out/err

- Οι πρώτες τρεις εγγραφές στο ***File Descriptor Table*** είναι ορισμένες με κάθε εκκίνηση του κελύφους.

- Entry 0 = Standard Input
- Entry 1 = Standard Output
- Entry 2 = Standard Error



- Για να καταλάβουμε καλύτερα τι γίνεται μπορούμε να αναλύσουμε το ***/proc*** filesystem, περιέχει πληροφορίες για κάθε διεργασία στο σύστημα υπό μορφή text)

- `/dev/stdin => /proc/self/fd/0 ή /proc/$$/fd/0` : Standard Input
`/dev/stdout => /proc/self/fd/1 ή /proc/$$/fd/1` : Standard Output
`/dev/stderr (/proc/self/fd/2 ή /proc/$$/fd/2)` : Standard Error

Η εντολή ***ls*** παρουσιάζει όλα τα ανοικτά αρχεία του συστήματος
Όλα τα ανοικτά αρχεία του κελύφους σας; ***ls | grep \$\$***

Bash Process ID



Παράδειγμα 8

*Να γραφεί ένα πρόγραμμα το οποίο να προσομοιώνει την εντολή **more**. Δηλαδή το πρόγραμμα παρουσιάζει οτιδήποτε προέρχεται από το *standard input* (#0), σε γραμμές των 30 και στην συνέχεια **“Press enter for more...”**. Όταν ο χρήστης δώσει *enter*, τότε παρουσιάζεται την επόμενη σελίδα.*

Λύση Παραδείγματος 8

(Παράδειγμα I/O Redirect)



```
$ls /bin | ~/.mymore.sh
```

```
#!/bin/bash
#
# Max number of rows per screen
ROWS=30

clear

# Current number of rows on screen
i=0

read line
while [ "$line" != "" ];
do
    handlescreen:
    echo $line
    read line
    ((i++))
done

exit 0
```

```
handlescreen() {
# Εάν εκτυπωθήκαν ROWS γραμμές στο αρχείο τότε
if [ "$i" -gt "$ROWS" ]; then
    echo "Please Press Enter to continue..."

    #FILE DESCRIPTOR (FD)
    # store stdin (FD#0) into some random (and available) FD#6
    # Avoid FD#5: See Advanced Bash Scripting book
    # recall FD#0: stdin, FD#1: stdout, FD#2: stderr
    exec 6<&0 # προσωρινή αποθήκευση ροής #0 στο #6.

    # get the stdio from /dev/tty (the file that captures the keyboard)
    exec < /dev/tty # η ροή εισόδου έρχεται από το keyboard τώρα

    read userenter
    #echo "User Input: $userenter"

    # 0<&6 : restore FD#6 into FD#0
    # (so that we can continue reading the file)
    # 6<&- : deallocate FD#6 so that it can be utilized later on
    exec 0<&6 6<&- # μεταβολή ροής #0 στο #6 και κλείσιμο #6

    # reset the counter
    i=0; clear

fi
```

Ανακατεύθυνση (Redirect) Εισόδου / Εξόδου



- **Redirect File Descriptor to File: $M > N$** (also applies to $>>$)
 - π.χ.: M: file descriptor (default=1), N: file name
 - π.χ. `$ ls 2> error.log # stderr is redirected to log`
- **Redirect Input $<$:**
 - π.χ.: `$ cat <input.data`
 - π.χ.: `$ cat <input.data 2> /dev/null`
- **Redirect Output and Error $2>1$ ή $&>$:**
 - `ls 2>1 output_and_error_log # same ls &> oelogs`
- **Redirect FD/FD: $M<&N$** (M: file descriptor, N: file descriptor)
 - **File Descriptor:** Ένας ακέραιος ο οποίος μας δίνει πρόσβαση σε ένα αρχείο που έχει ήδη ανοίξει από μια διεργασία (π.χ., μετά την `fork` στην C) ή την ακόλουθη εντολή στο unix.
 - π.χ. `exec 6<&0 #store stdin (FD#0) into FD#6`



Διάφορες Χρήσεις των Συσκευών

- Έστω ότι θέλουμε να ψάξουμε για όλες τις εμφανίσεις της λέξης “man” στο σύστημα
\$find / -name "man"
- Προφανώς αυτό θα αναγκάσει το κέλυφος να ψάξει σε καταλόγους όπου δεν έχουμε πρόσβαση, με αποτέλεσμα να πάρουμε πολλά «**Permission Denied**» στην οθόνη.
- Για να διορθώσουμε το πρόβλημα δίνουμε
\$find / -name "man" 2>/dev/null
Τώρα όλα τα stderr διοχετεύονται στο /dev/null (μια μαύρη τρύπα!)

Ανακατεύθυνση (Redirect) Εισόδου / Εξόδου



- Τα δυο ακόλουθα προγράμματα κάνουν ακριβώς την ίδια λειτουργία με την εντολή `ls -al`
- Σημειώστε ότι τα δυο προγράμματα έχουν υλοποιηθεί σαν φίλτρα.
- Για την εκτέλεση τους πληκτρολογούμε

ls -al | test.sh

```
#!/bin/bash  
while read line  
do  
    echo $line  
done
```

```
#!/bin/bash  
#store stdin (FD#0) into FD#6 (that is  
arbitrarily chosen)  
exec 6<&0  
# cat anything that comes from FD#6  
cat <&6  
# close file number 6  
exec 6<&-  
#redirect nothing "-" into FD#6
```



Παραδείγματα Προγραμματισμού Κελύφους

(Περισσότερα θα καλυφθούν
στο εργαστήριο)



Παράδειγμα 1

*Να γραφεί ένα πρόγραμμα **Isdir** για το κέλυφος **Bash**, το οποίο με τιμή εισόδου ένα κατάλογο **A**, εμφανίζει τους καταλόγους που βρίσκονται κάτω από τον **A**, σε οποιοδήποτε βάθος, καθώς επίσης και την ημερομηνία και ώρα τελευταίας τροποποίησης τους (ή δημιουργίας τους).*



Λύση Παραδείγματος 1

```
#!/bin/sh
#
# Usage: lsdir directory
#
if [ $# -eq 0 ]; then
    echo "Exactly one argument is required"
elif [ $# -ge 2 ]; then
    echo "Too many arguments"
else
    # ls -l(list) -R(recursively)
    #
    # grep "^d"
    #-rwxrwxrwx  1 dzeina None  514 Jan 14 14:00 a.sh
    #drwxrwxrwx+  2 dzeina None   0 Jan  5  2006 bin
    #drwxrwxrwx+  5 dzeina None   0 Jan  5  2006 blib
```

```
Αποτέλεσμα - Έξοδος
Jan 15 12:45 advio
Jan 15 12:45 call
Jan 15 12:45 calld
Jan 15 12:45 datafiles
Jan 15 12:45 db.lock.fine
Jan 15 12:45 environ
Jan 15 12:45 file
Jan 15 12:45 ipc
Jan 15 12:45 lib.44
```

ls -lR | grep "^d" | awk '{print \$6" "\$7"\t"\$8"\t"\$9}'

fi



Παράδειγμα 5

Να γραφεί ένα πρόγραμμα για το κέλυφος `Bash` που να υπολογίζει το παραγοντικό ($x!$) ενός ακεραίου αριθμού x , επαναληπτικά και αναδρομικά

Λύση Παραδείγματος 5 (Παράδειγμα Αναδρομής)



Για μεγάλες τιμές προκαλείται stack overflow

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

```
echo -n "Give input number: "  
read n
```

```
((m=n)) # copy variable  
result=1
```

A) Repetitive solution

```
until [ $m -eq 0 ]  
do  
    ((result*=m))  
    ((m--))  
done  
echo "Factorial of $n is $result"
```

B) Recursive solution

```
fact $n  
echo "Factorial of $n is $?."
```

```
fact () {  
    # Variable "number" must be declared as local,  
    #+ otherwise this doesn't work.  
    local number=$1  
  
    if [ "$number" -eq 1 ]; then  
        factorial=1  
    else  
        ((decrnum=number-1))  
        fact $decrnum # Recursive function call  
        ((factorial = number * $?))  
    fi  
  
    return $factorial  
}
```

→ Το \$? είναι η τιμή επιστροφής (return value), της τελευταίας συνάρτησης



Παράδειγμα 7

Να γραφεί ένα πρόγραμμα για το κέλυφος `Bash`, που να υπολογίζει το μέγιστο χώρο που καταλαμβάνουν τα περιεχόμενα μιας λίστας καταλόγων, η οποία δίδεται σαν δεδομένο εισόδου

Λύση Παραδείγματος 7



```
#!/bin/bash
#
# Usage: maxsize
#
echo -n "Please specify the directory names: "
read input; # e.g. "cvroot bin blib c"
set - $input; # this commands segments input into "$1 $2 $3 ...."
# based on the envir. Variable $IFS
#e.g. IFS=$' \n' IFS="+", etc...
maxdirsize=0
```

```
for i # goes through all the command line arguments (i.e., $*)
do
  if [ ! -d $i ]; then
    echo "Warning: There is no directory $i"
  else
    # du -s(supress): summarize disk use - i.e, show only total space, output: 20 bin (i.e. 20 KB)
    dirsize=`du -s $i | awk '{print $1}'`
    if [ $dirsize -gt $maxdirsize ]; then
      maxdirsize=$dirsize
      maxdirname=$i
    fi
  fi
done
echo "Biggest Directory:$maxdirname, Size:$maxdirsize KB"
```

Παράδειγμα Εξόδου

```
./test.sh
```

Please specify the directory names:

```
test1 test2 cvroot a b c d e f
```

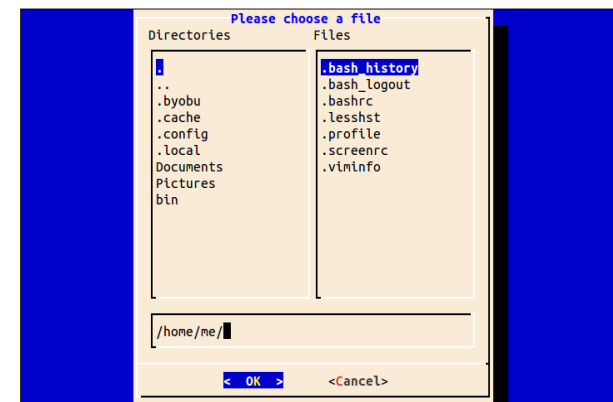
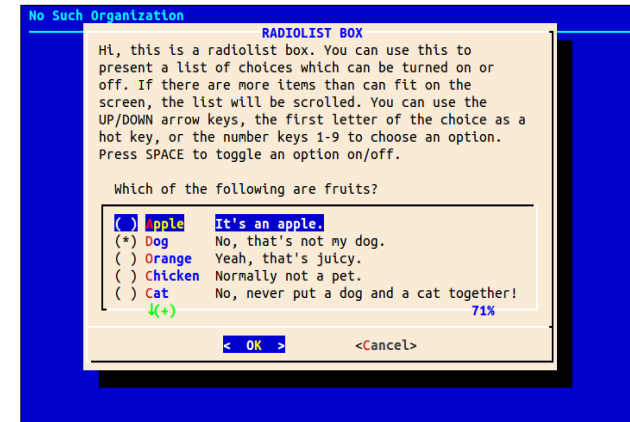
Warning : There is no directory cvroot

Biggest Directory:c, Size:20 KB



Dialogs

- **Dialog** is an application used in [shell scripts](#) which displays [text user interface widgets](#).
 - Uses the [curses](#) or [ncurses](#) library.
 - The latter provides users with the ability to use a mouse, e.g., in an [xterm](#).
- The data that dialog takes in (such as a string entered into a input box) is normally returned **on standard error**.
 - This is because dialog uses **standard output** to display text on the terminal when it is drawing the dialog box itself.
 - We will see next how to handle this ...



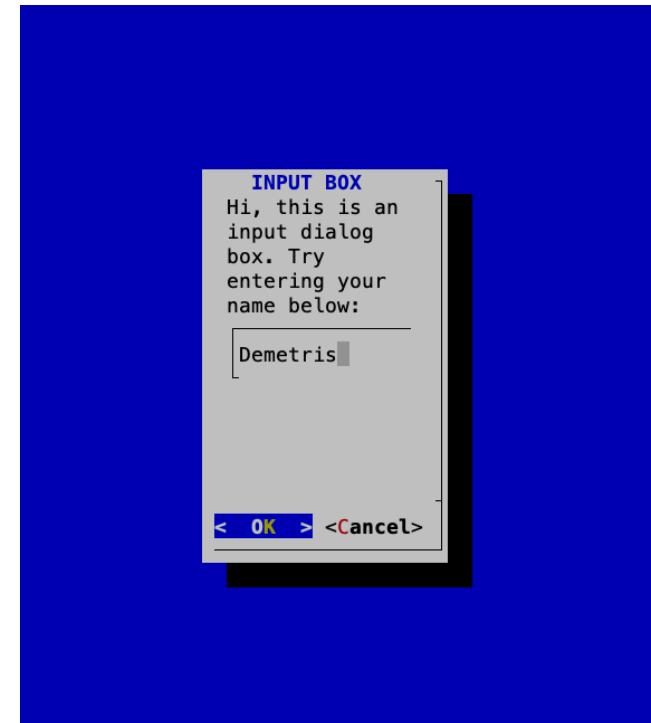


Dialog install & usage

- Debian or Ubuntu Linux
 - `$ sudo apt-get update`
 - `$ sudo apt-get install dialog`
- CentOS or Red Hat Linux:
 - `$ sudo yum install dialog`

- **Usage**

```
dialog --title "INPUT BOX" --clear  
--inputbox "Hi, this is an input  
dialog box. Try entering your name  
below:" 16 12 # inputbox size
```



Example #1

dialog, trap, exit values &



```
#!/bin/bash
# inputbox - demonstrate the input box with a temp
file
# Define the dialog exit status codes
: ${DIALOG_OK=0}
: ${DIALOG_CANCEL=1}
: ${DIALOG_ESC=255}

# Create a temp file and trap to clear when done
tmp_file="/tmp/test$$RANDOM"
trap "rm -f $tmp_file" 0 1 2 5 15
# 0 On exit from shell, 1 SIGHUP, 2 SIGINT, 6 SIGABRT, 15 SIGTERM

# Generate the dialog box
dialog --title "INPUT BOX" \
  --clear \
  --inputbox \
  "Hi, this is an input dialog box:" \
  16 51 2> $tmp_file
```

Process id

```
# Get the exit status
return_value=$?

# Act on it
case $return_value in
  $DIALOG_OK)
    echo "Result: `cat $tmp_file`";;
  $DIALOG_CANCEL)
    echo "Cancel pressed.";;
  $DIALOG_ESC)
    if test -s $tmp_file ; then
      cat $tmp_file
    else
      echo "ESC pressed."
    fi
  ;;
esac
```

Output goes to file

Example #1

dialog, trap, exit values



INPUT BOX

Hi, this is an input dialog box. You can use this to ask questions that require the user to input a string as the answer. You can input strings of length longer than the width of the input box, in that case, the input field will be automatically scrolled. You can use BACKSPACE to correct errors.

Try entering your name below:

< OK > <Cancel>

ESC pressed.

Example 2: Menu Set



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

```
DIALOG_CANCEL=1
```

```
DIALOG_ESC=255
```

```
HEIGHT=0
```

```
WIDTH=0
```

```
display_result() {  
  dialog --title "$1" \  
    --no-collapse \  
    --msgbox "$result" 0 0  
}
```

```
while true; do
```

```
  exec 3>&1 # backup FD#1 on FD#3 (opens FD#3)
```

```
  selection=$(dialog \  
    --backtitle "System Information" \  
    --title "Menu" \  
    --clear \  
    --cancel-label "Exit" \  
    --menu "Please select:" $HEIGHT $WIDTH 4 \  
    "1" "Display System Information" \  
    "2" "Display Disk Space" \  
    "3" "Display Home Space Utilization" \  
    2>&1 1>&3) # (1 and 2) to 3
```

```
  exit_status=$? # exit code of prior subshell
```

```
  exec 3>&- #Close FD#3
```

**We don't want to
output on
screen but on
FD#3! (dialogs)**



subshell



```
  case $exit_status in  
    $DIALOG_CANCEL)  
    clear  
    echo "Program terminated."  
    exit  
    ;;  
    $DIALOG_ESC)  
    clear  
    echo "Program aborted." >&2  
    exit 1  
    ;;  
  esac  
  case $selection in #sub windows open here  
    1 )  
    1 result=$(echo "Hostname: $HOSTNAME"; uptime)  
    display_result "System Information"  
    ;;  
    2 )  
    result=$(df -h)  
    display_result "Disk Space"  
    ;;  
    3 )  
    if [[ $(id -u) -eq 0 ]]; then  
      result=$(du -sh /home/* 2> /dev/null)  
      display_result "Home Space Utilization (All Users)"  
    else  
      result=$(du -sh $HOME 2> /dev/null)  
      display_result "Home Space Utilization ($USER)"  
    fi  
    ;;  
  esac  
done
```



System Information

Menu

Please select:

- 1 Display System Information
- 2 **Display Disk Space**
- 3 Display Home Space Utilization

< OK > < Exit >

Disk Space

Filesystem	Size	Used	Avail	Use%	Mounted on
rootfs	5.8G	4.5G	984M	83%	/
/dev/root	5.8G	4.5G	984M	83%	/
devtmpfs	215M	0	215M	0%	/dev
tmpfs	44M	256K	44M	1%	/run
tmpfs	5.0M	0	5.0M	0%	/run/lock
tmpfs	88M	0	88M	0%	/run/shm
/dev/mmcblk0p5	60M	15M	45M	25%	/boot

< OK >

Home Space Utilization (ubuntu)

120K /home/ubuntu

< OK >