To Shell

CLI vs. GUI

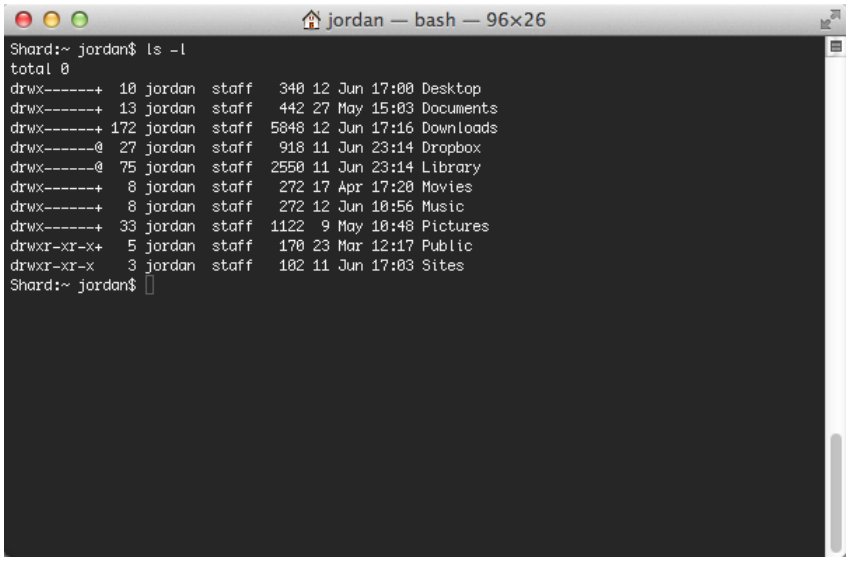
CLI (Command Line Interface)

- Μεγαλύτερη ευελιξία και ταχύτητα
- Πιο εύκολο να γίνουν πολύπλοκες λειτουργίες που συνδυάζουν διαφορετικά προγράμματα/εργαλεία
- Υποστήριξη scripting

GUI (Graphical User Interface)

- Γραφική αναπαράσταση προγραμμάτων/αρχείων
- Χρήση συμπληρωματικών συσκευών εισόδου (ποντίκι)
- Παραθυρικό περιβάλλον, πιο εύκολο για αρχάριους
- Διευκολύνει την ταυτόχρονη εκτέλεση διαφορετικών εργασιών (multitasking) υπό τον έλεγχο του χρήστη

Terminal



Παλιά:

Ειδικό περιφερειακό επικοινωνίας με τον ΗΥ, με οθόνη & πληκτρολόγιο

Τώρα:

Αφαίρεση υλοποιημένη σε λογισμικό

Shell

- Πρόγραμμα που υποστηρίζει την αλληλεπίδραση του χρήστη με το λειτουργικό, μέσω ενός τερματικού

- Βασική λειτουργικότητα: **commands & job control**

- Ερμηνεία εντολών που διαβάζονται από το τερματικό, εμφάνιση αποτελεσμάτων/μηνυμάτων στο τερματικό
- Δημιουργία και διαχείριση διεργασιών για την εκτέλεση των εντολών που δίνει ο χρήστης
- Υποστήριξη πολύπλοκων ροών εργασίας, με χρήση αγωγών και κατάλληλη ανακατεύθυνση Ε/Ε

Sessions & process groups

- **Process group:** ομάδα από ξεχωριστές διεργασίες που μοιράζονται το ίδιο group pid (gpid)
- **Session:** συλλογή από ξεχωριστά process groups

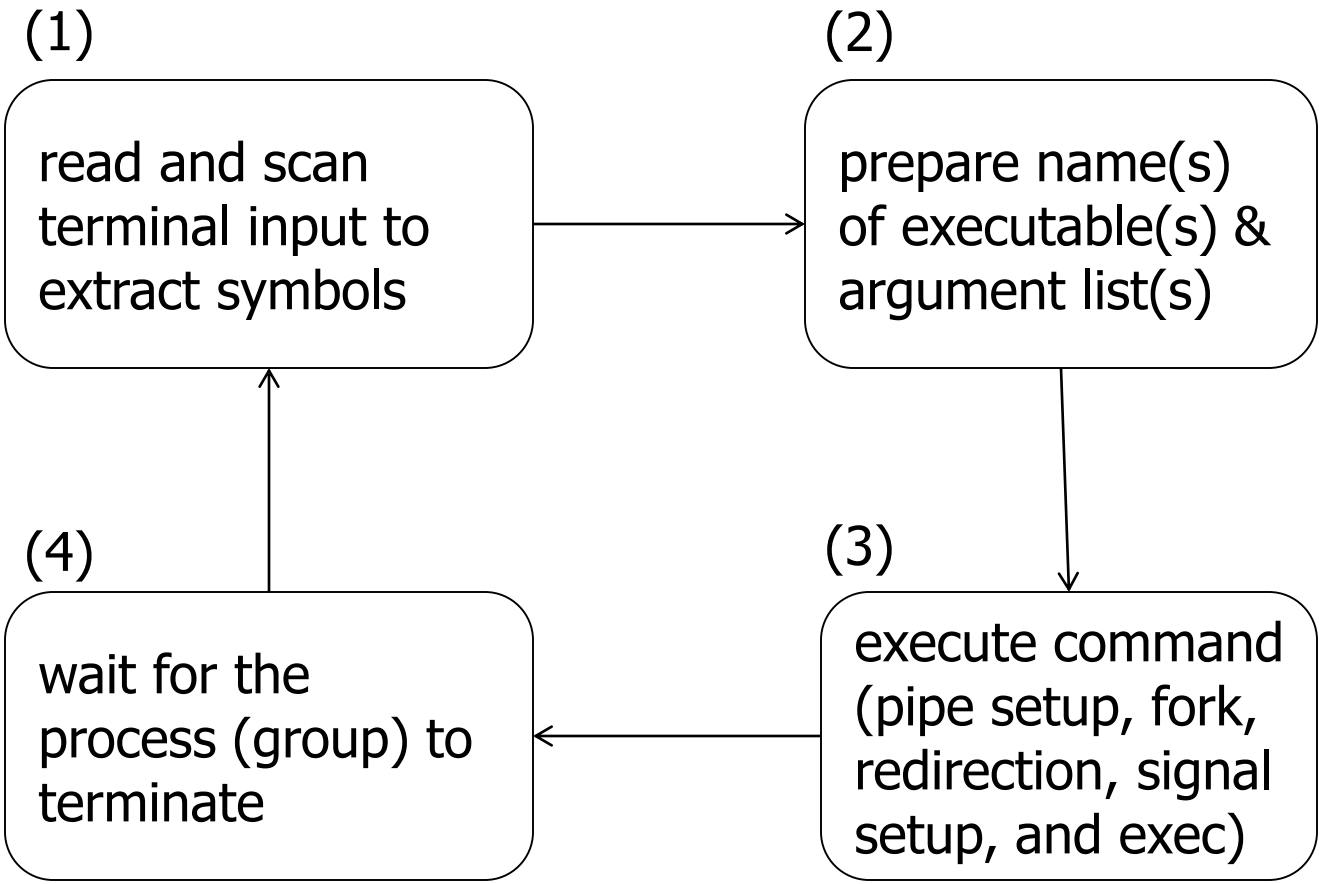
- Κάθε session συνήθως σχετίζεται με ένα συγκεκριμένο τερματικό, το οποίο ονομάζεται **controlling terminal**

- Οι διεργασίες του session διαβάζουν από / γράφουν σε αυτό το terminal
- Το controlling terminal διαχειρίζεται/ελέγχει τις διεργασίες του session στέλνοντας σήματα σε αυτές

Εκτέλεση εντολών από το shell

- Το shell διαβάζει την εντολή που δίνει ο χρήστης από το τερματικό
- Δημιουργεί μια ή περισσότερες διεργασίες που χρησιμοποιούνται για την εκτέλεση της εντολής
- Οι νέες διεργασίες αποτελούν μια **καινούργια** ομάδα (process group), έτσι ώστε να μπορεί να ελέγχονται πιο εύκολα ως σύνολο (π.χ. να τερματιστούν όλες μαζί)
- Το shell **περιμένει** μέχρι να τερματίσει η ομάδα διεργασιών που εκτελεί την εντολή του χρήστη
- Αν κατά την εκτέλεση το terminal παράγει ένα σήμα, αυτό στέλνεται σε όλη την ομάδα διεργασιών
- Κάθε εκτέλεση εντολής ονομάζεται και εργασία (**job**)

Βασικός κύκλος εκτέλεσης



Πρόβλημα πολυπλεξίας Ε/Ε

- Το shell μπορεί να εκτελεί πολλές εργασίες ταυτόχρονα
  - κάθε μια μπορεί να είναι μια ροή επεξεργασίας (pipeline)
- Κάθε εργασία (job) έχει **μια** διεργασία που μπορεί να διαβάσει από το τερματικό μέσω stdin, και **μια** που μπορεί να γράψει στο τερματικό μέσω stdout
  - οι διεργασίες που βρίσκονται στην αρχή/τέλος του pipeline
- Κάθε εργασία έχει **πολλές** διεργασίες που μπορεί να γράψουν στο τερματικό μέσω stderr
- Μπορείτε να γίνει αυτόματη πολυπλεξία των ροών γραψίματος των διεργασιών πάνω στο τερματικό
- **Αυτό δεν μπορεί να γίνει για τις ροές εισόδου**

Foreground & background processing

- Κατά σύμβαση, το shell εκτελεί τις εντολές/εργασίες που δίνει ο χρήστης στο προσκήνιο (**foreground**)
- Αν ο χρήστης προσθέσει `&` τότε η εντολή/εργασία εκτελείται στο παρασκήνιο (**background**)
  - στο προσκήνιο μένει το shell, που μπορεί να δεχτεί και να εκτελέσει νέες εντολές του χρήστη, ως συνήθως
- Ο χρήστης μπορεί να επαναφέρει μια εργασία στο προσκήνιο με την εντολή `fg <jobnr>`
- Ο χρήστης μπορεί να αναστείλει μια εργασία που τρέχει στο προσκήνιο με `Ctrl-Z` (αποστολή `SIGTSTP`)
- Και να την συνεχίσει στο παρασκήνιο με `bg <jobnr>`
- Διάφορες εντολές υποστηρίζονται και σε επίπεδο εργασιών, π.χ. `wait`, `kill` (αριθμός job με `%`)

Foreground/background & I/O multiplexing

- Μια διεργασία μπορεί να διαβάσει δεδομένα από το τερματικό **μόνο** αν αυτή εκτελείται στο προσκήνιο
- Αν μια διεργασία που είναι στο παρασκήνιο επιχειρήσει να διαβάσει από το τερματικό, δέχεται σήμα `SIGTTIN` (κατά σύμβαση οδηγεί σε **ανατολή** της εκτέλεσης)
- Αν μια διεργασία στο παρασκήνιο επιχειρήσει να γράψει στο τερματικό, αυτό συνήθως **πετυχαίνει** – μπορεί να αλλάξει με *`stty tostop`*, οπότε η διεργασία θα δεχτεί το σήμα `SIGTTOU` (οδηγεί σε αναστολή εκτέλεσης)
- Μια παρασκηνιακή διεργασία η εκτέλεση της οποίας έχει ανασταλεί λόγω Ε/Ε, **συνεχίζει αυτόματα** την εκτέλεση της όταν επανέλθει στο προσκήνιο

Μερικές δοκιμές

- `test1` διαβάζει από το stdin και γράφει στο stdout

Δοκιμάζουμε τα εξής:

```
./test1
Ctrl-Z
Ctrl-C
bg <jobnr>
fg <jobnr>
Ctrl-D
```

Μερικές δοκιμές

- `test2` γράφει τα ορίσματα της στο stdout

Δοκιμάζουμε τα εξής:

```
./test2 one two 3 four &
stty tostop
fg <jobnr>
Ctrl-Z
bg <jobnr>
stty -tostop
bg <jobnr>
```

Μερικές δοκιμές

- `test1` διαβάζει από το stdin και γράφει στο stdout
- `test2` γράφει τα ορίσματα της στο stdout

Δοκιμάζουμε τα εξής:

```
./test2 one two 3 four | ./test1 &
stty tostop
fg <jobnr>
Ctrl-Z
stty -tostop
bg <jobnr>
```

Προγραμματισμός με το Shell (shell scripting)

Shell scripts

- Ο χρήστης μπορεί να χρησιμοποιήσει το shell για να εκτελέσει εντολές, μια προς μια, μέσα από τον κλασικό (διαδραστικό) κύκλο εκτέλεσης
- Εναλλακτικά, μπορεί να ζητήσει να εκτελεστεί μια **ολόκληρη ακολουθία** από εντολές, στο πνεύμα ενός «κανονικού» προγράμματος

- Αυτά τα προγράμματα ονομάζονται **shell scripts**
- Ένα shell script εκτελείται μέσα από τον βασικό κύκλο εκτέλεσης, όπως οποιοδήποτε πρόγραμμα
- Σε αυτή την περίπτωση, το shell δημιουργεί μια διεργασία-παιδί shell που εκτελεί το shell script

μπορεί κανείς να γράφει shell scripts με έναν text editor

```
#!/bin/sh
echo Hello World
```

ή να τα παράγει με προγραμματιστικό τρόπο

```
$ echo '#!/bin/sh' > hello-world.sh
$ echo 'echo Hello World' >> hello-world.sh
```

και στην συνέχεια να τα τρέξει

```
$ chmod 755 hello-world.sh
$ ./hello-world.sh
Hello World
$
```

Φιλοσοφία

- Τα shell scripts **ερμηνεύονται** (δεν μεταφράζονται)
- Ο συντακτικός έλεγχος και η εκτέλεση γίνεται **ξεχωριστά** για κάθε εντολή

- Τα shell scripts δεν μπορεί να ανταγωνιστούν σε ταχύτητα εκτέλεσης ένα μεταφρασμένο πρόγραμμα, ούτε φτιάχνονται με αυτή τη λογική

- Το πλεονέκτημα του shell scripting βρίσκεται στο ότι μπορεί κανείς να **συνδυάσει**, εύκολα και ευέλικτα, υπάρχουσες εντολές και έτοιμα προγράμματα

Μεταβλητές

- Οι μεταβλητές δεν δηλώνονται ξεχωριστά
- «Προκύπτουν» όπως εμφανίζονται στο script
- Μη αρχικοποιημένες μεταβλητές είναι κενές/άδειες

- Η αναφορά στην μεταβλητή `X` γίνεται με `$X`

- Οι μεταβλητές δεν έχουν συγκεκριμένο τύπο
- Η ερμηνεία μιας μεταβλητής εξαρτάται από το πλαίσιο αναφοράς

```
#!/bin/sh

A="Hello World"
echo A
echo $A

echo Please enter something
read B
echo $B
echo "$B"

echo "The value of C is *$C*"

D=15
E="30"
F=`expr $D + $E`
echo $F
F=`expr $D+$E`
echo $F
```

Εμβέλεια μεταβλητών

- Οι μεταβλητές και οι όποιες αλλαγές που γίνονται σε αυτές ισχύουν μόνο στο πλαίσιο της αντίστοιχης εκτέλεσης

- Κατ' εξαίρεση, μια μεταβλητή που δημιουργείται στο πλαίσιο της εκτέλεσης Α μπορεί να γίνει ορατή στο πλαίσιο μιας εκτέλεσης Β, αν η Α δημιούργησε την Β
- Π.χ., ορατότητα μεταβλητής του διαδραστικού shell μέσα σε script που εκτελείται μέσα από το shell

- Εντολή για αυτό είναι η `export`
- Ακύρωση μεταβλητής (όχι εμβέλειας) με `unset`

```
#!/bin/sh
echo "X is: $X"
X="hi from the script"
echo "X is: $X"
```

```
$
$ ./test.sh
X is:
X is: hi from the script
$
```

```
$
$ X = "hi from the shell"
$ echo $X
hi from shell
$
$
$ ./test.sh
X is:
X is: hi from the script
$
$
$ echo $X
hi from the shell
$
```

```
$
$ X = "hi from the shell"
$ echo $X
hi from shell
$
$ export X
$ ./test.sh
X is: hi from the shell
X is: hi from the script
$
$
$ echo $X
hi from the shell
$
```

```
$ unset X
$ echo $X

$
$ . ./test.sh
X is:
X is: hi from the script
$
$
$ echo $X
hi from the script
$
$ unset X
$ echo $X

$
```

αντί να δημιουργηθεί νέα διεργασία για την εκτέλεση του script, οι εντολές εκτελούνται μέσα από το περιβάλλον του διαδραστικού shell

Ονόματα μεταβλητών

- Οι μεταβλητές μπορεί να έχουν περίπλοκα ονόματα
- Χρειάζεται προσοχή αν θέλουμε να συνδυάσουμε αναφορές σε μεταβλητές με άλλα strings
- Για να μην προκύψουν απρόβλεπτα αποτελέσματα στην αναγωγή μιας αναφοράς, η αναφορά στην μεταβλητή μπορεί να γίνει με `${VAR_NAME}`

```
#!/bin/sh
echo "What is your name?"
read X
echo "Hello $X"

echo "Going to create a file called $X_file"
touch $X_file

echo "And this time, do it the right way"

echo "Going to create a file called ${X}_file"
touch ${X}_file

echo "And one more time, the really right way"

touch "${X}_file"
```

Quotes, wildcards, escape characters

- Η μεγάλη ευελιξία που υπάρχει στην ερμηνεία των μεταβλητών, φέρνει εύκολα και διάφορα bugs
- Το αποτέλεσμα μιας εντολής δεν είναι πάντα προφανές
- Προσοχή στην χρήση quotes
- Προσοχή στην χρήση wildcards
- Προσοχή στην χρήση ειδικών χαρακτήρων

```
#!/bin/sh
echo "Hello      World"
echo "Hello World"
echo "Hello * World"
echo Hello * World
echo Hello      World
echo "Hello" World
echo Hello "     " World
echo "Hello "*" World"
echo Hello      "World"
echo Hello      \"World\"
echo "Hello     \"World\""
echo *.sh
echo "*.sh"
echo "A quote is \", backslash is \\, backtick is \`."
echo "A few spaces are    ; dollar is \$. \$X is ${X}."
```

Λογικές εκφράσεις

- Ειδικό πρόγραμμα `test` (ή `[`) για ελέγχους που απαιτούν κάτι παραπάνω από μια αριθμητική σύγκριση
- Παίρνει ως όρισμα μια έκφραση την οποία αποτιμά είτε ως αληθή είτε ως ψευδή
- Το αποτέλεσμα μπορεί να συνδυαστεί στο πλαίσιο μιας ευρύτερης λογικής έκφρασης
- Χρησιμοποιείται συχνά σε δομές διακλάδωσης και σε δομές επανάληψης

```
#!/bin/sh
echo "Please enter something"
read X

[ "$X" -le "0" ] && echo "X is less than or equal to  zero"

[ "$X" -ge "0" ] && echo "X is more than or equal to zero"

[ "$X" = "0" ] && echo "X is the string or number \"0\""

[ "$X" = "hello" ] && echo "X matches the string \"hello\""

[ "$X" != "hello" ] && echo "X is not the string \"hello\""

[ -n "$X" ] && echo "X is of nonzero length"

[ -f "$X" ] && echo "X is the path of a real file" || \
     echo "No such file: $X"

[ -x "$X" ] && echo "X is the path of an executable file"

[ "$X" -nt "/etc/passwd" ] && \
     echo "X is a file which is newer than /etc/passwd"
```

Διακλαδώσεις

- Όπως και σε μια «κανονική» γλώσσα προγραμματισμού, υπάρχει δομή διακλάδωσης
- Δομή `if-then-else-fi` όπου αν η συνθήκη αποτιμηθεί σε αληθή τότε εκτελείται ο κώδικας του `then`, διαφορετικά ο κώδικας του `else` (αν υπάρχει)

```
#!/bin/sh
echo "Please say \"hello\""
read X

if [ "$X" = "hello" ]
then
  echo "Have a nice day!"
else
  echo "You didn't say hello!"
fi

if [ "$X" = "hello" ] ; then
  echo "Have a nice day!"
else
  echo "You didn't say hello!"
fi

if test "$X" = "hello"
then
  echo "Have a nice day!"
else
  echo "You didn't say hello!"
fi
```

Επαναλήψεις

- Δομή `for` όπου ο αριθμός των επαναλήψεων καθορίζεται μέσω μιας ακολουθίας – γίνεται μια επανάληψη για κάθε στοιχείο της ακολουθίας

- Δομή `while` όπου ο αριθμός των επαναλήψεων καθορίζεται μέσω μιας συνθήκης – γίνονται επαναλήψεις μέχρι η συνθήκη να μην ισχύει

```
#!/bin/sh

for i in hello 1 2 3 goodbye
do
  echo "looping, and i is set to $i"
done

for i in hello 1 "*" 2 goodbye
do
  echo "looping, and i is set to $i"
done

for i in hello 1 * 2 goodbye
do
  echo "looping, and i is set to $i"
done
```

```
#!/bin/sh

X=notbye
while [ "$X" != "bye" ]
do
  echo "Please type something in (bye to quit)"
  read X
  echo "You typed: $X"
done

while :
do
  echo "Please type something (bye to quit)"
  read X
  echo "You typed: $X"
  if [ "$X" = "bye" ]; then
    break
  fi
done
```

```
#!/bin/sh
echo "Please type a file name"
read FNAME
while read L
do
  echo "$L"
done < $FNAME
```

Παράμετροι

- Ένα script παίρνει ορίσματα/παραμέτρους, όπως ακριβώς και ένα «κανονικό»/συμβατικό πρόγραμμα

- Τα ορίσματα προσπελάζονται μέσα από διάφορες προκαθορισμένες μεταβλητές

- `$0` όνομα του script
- `$1` πρώτο όρισμα που έδωσε ο χρήστης
- `$2` δεύτερο όρισμα που έδωσε ο χρήστης
- `$@` όλα τα ορίσματα μαζί
- `$#` αριθμός ορισμάτων

```
#!/bin/sh
echo "I was called with $# parameters"
echo "My name is $0"
echo "My first parameter is $1"
echo "My second parameter is $2"
echo "All parameters are $@"

echo "Iterating over all parameters"
while [ "$#" -gt "0" ]
do
  echo "\$1 is $1"
  shift
done
```

Εξωτερικά προγράμματα

- Η εκτέλεση ενός προγράμματος γίνεται μέσα από ξεχωριστή διεργασία που δημιουργείται για αυτό το σκοπό – όπως γίνεται και στο διαδραστικό shell
- Η τιμή επιστροφής αποθηκεύεται στην μεταβλητή `$?`
- Συνήθως δείχνει αν το πρόγραμμα εκτελέστηκε επιτυχώς (κατά σύμβαση `0` = επιτυχία)

- Με backtips `` `...` `` το shell καταγράφει και επιστρέφει τα δεδομένα που γράφονται από την εντολή στο (stdout)
- Ώστε να μπορεί να αποθηκευτούν σε μια μεταβλητή
- Εναλλακτική σύνταξη είναι το `$(...)`

```
#!/bin/sh
while :
do
  echo "Please enter a command"
  read C
  if [ -z "$C" ]; then
    break
  fi
  $C
  if [ $? -eq "0" ]
  then
    echo "worked fine"
  else
    echo "something went wrong"
  fi
  echo "And once more but store output in a variable"
  R=`$C`
  echo "And this is the output with proper formatting"
  echo "$R"
done
```

```
#!/bin/sh
echo "Please enter a path"
read P

echo "Please enter a file suffix, e.g., \"sh\""
read S

R=`find $P -name "*.$S" -print` #store result
echo "$R"

while :
do
  echo "Please enter a string to use as a filter"
  read Y
  if [ -z "$Y" ]
  then
    break
  fi
  echo "$R" | grep $Y
done
```

```
#!/bin/sh
echo "timing "$1" for $2 times"
TOT=0
USR=0
SYS=0
I=0

while [ "$I" -lt "$2" ]
do
  R=$( { time -p $1 ; } 2>&1 > /dev/null | awk 'NF{print $2}' )
  L=($R)
  echo "total ${L[0]}, user ${L[1]}, system ${L[2]}"
  TOT=$( echo "$TOT ${L[0]}" | awk '{ printf "%f", $1 + $2 }')
  USR=$( echo "$USR ${L[1]}" | awk '{ printf "%f", $1 + $2 }')
  SYS=$( echo "$SYS ${L[2]}" | awk '{ printf "%f", $1 + $2 }')
  I=`expr $I + 1`
done

TOT=$(echo "$TOT $2" | awk '{ printf "%f", $1 / $2 }')
USR=$(echo "$USR $2" | awk '{ printf "%f", $1 / $2 }')
SYS=$(echo "$SYS $2" | awk '{ printf "%f", $1 / $2 }')
echo "and the averages are:"
echo "total $TOT, user $USR, system $SYS"
```

redirect/dup stderr to stdout

redirect/dup stdout to /dev/null (ignore)

pipe stdout (previous stderr) down the line for processing

Συναρτήσεις

- Μια συνάρτηση δέχεται τις παραμέτρους της όπως ακριβώς το script δέχεται τα δικά του ορίσματα
- Μπορεί να επιστρέψει αποτελέσματα μέσω `echo`
  Μπορεί να έχει δικές της τοπικές μεταβλητές
  - δεν είναι ορατές εκτός της συνάρτησης
  - καταστρέφονται μετά την εκτέλεση της συνάρτησης
- Οι μεταβλητές που ορίζονται στο script έξω από τις συναρτήσεις είναι ορατές και μέσα στις συναρτήσεις
  - αν η συνάρτηση αλλάξει την τιμή τους, αυτή η αλλαγή θα είναι ορατή και αφού επιστρέψει η συνάρτηση
- Μπορεί να έχει ρητή τιμή επιστροφής με `return`
  - που ο καλών μπορεί να διαβάσει μέσω της μεταβλητής `$?`
  - όπως και για κανονικά προγράμματα, μόνο έως 255

```
#!/bin/sh

myfunc1()
{
  echo $(($1 + $2))
}

myfunc2()
{
  X=$(($1 + $2))
}

X=10

myfunc1 $X 5
echo $X
X=$(myfunc1 $X 5)
echo $X

myfunc2 $X 5
echo $X
X=$(myfunc2 $X 5)
echo $X
```

```
#!/bin/sh
square1()
{
  echo $(($1 * $1))
}

square2()
{
  return $(($1 * $1))
}

while :
do
  echo "Enter a number:"
  read X
  Y=$(square1 $X)
  echo $Y
  square2 $X
  echo $?
done
```