Προγραμματισμός II (ECE116)

#21

shell & scripting

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)

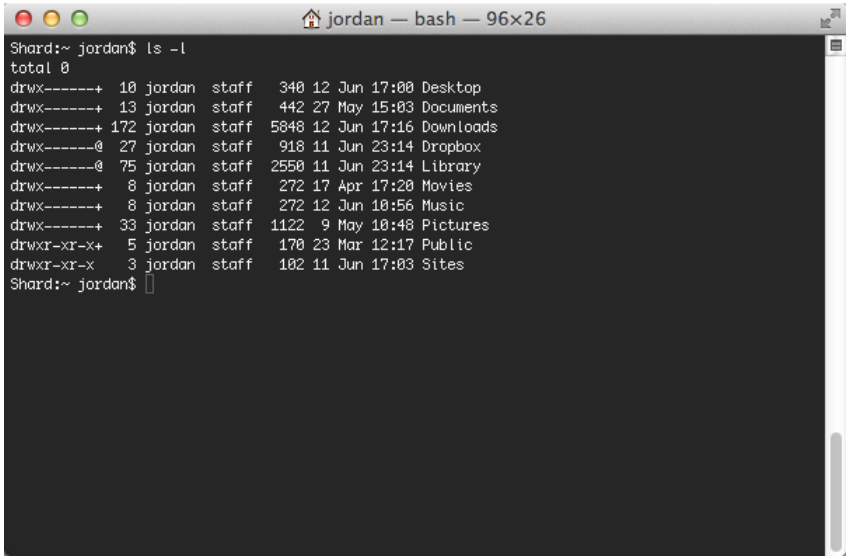
- Γραφική αναπαράσταση προγραμμάτων/αρχείων
- Χρήση συμπληρωματικών συσκευών εισόδου (ποντίκι)
- Παραθυρικό περιβάλλον, πιο εύκολο για αρχάριους
- Διευκολύνει την ταυτόχρονη εκτέλεση διαφορετικών εργασιών (multitasking) υπό τον έλεγχο του χρήστη

Terminal

παλιότερα

Ειδικό περιφερειακό επικοινωνίας με τον ΗΥ, με οθόνη & πληκτρολόγιο

τώρα



Αφαίρεση του τερματικού, υλοποιημένη σε λογισμικό

Shell

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

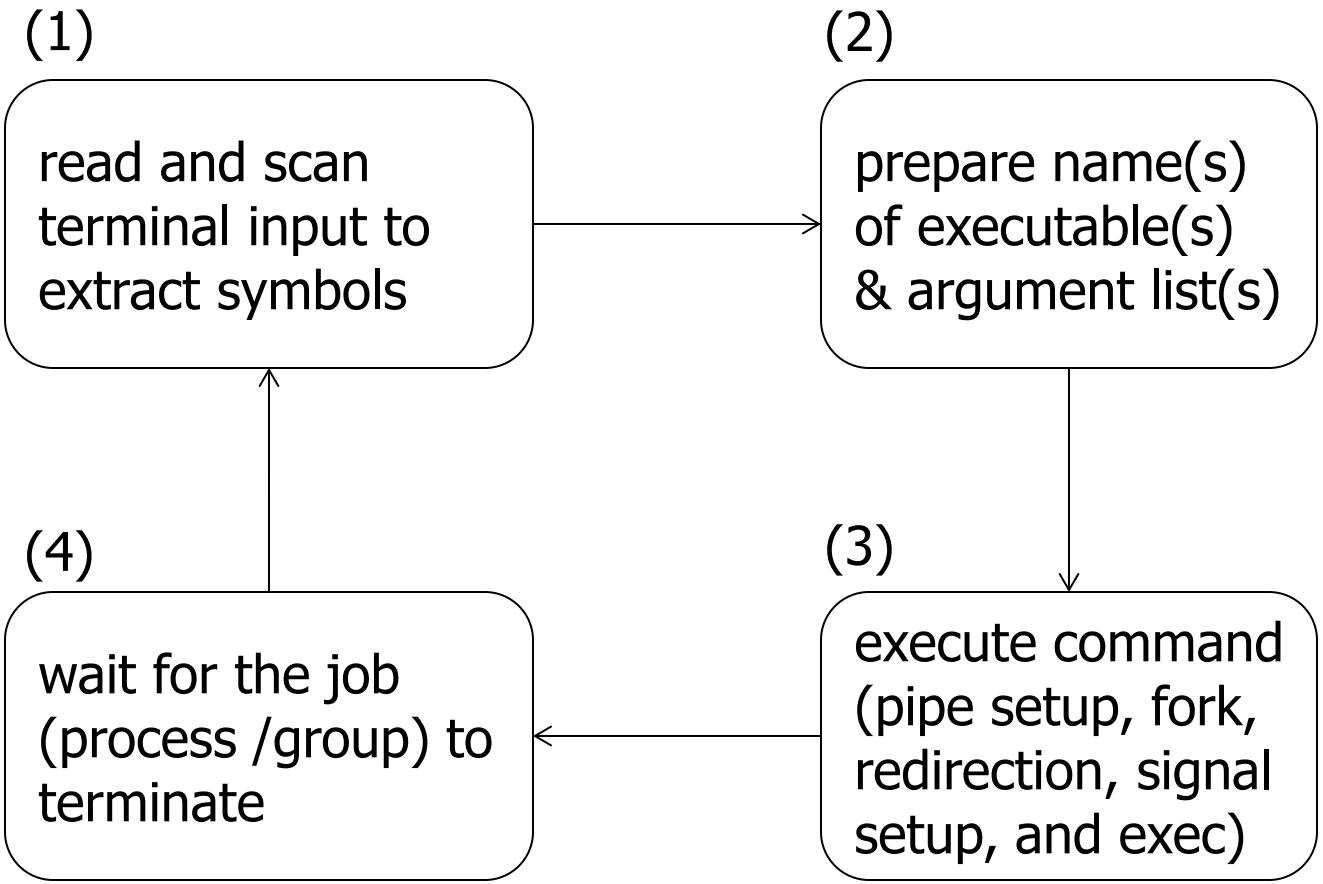
- Βασική λειτουργικότητα: **commands & job control**

- **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
- **Ροές επεξεργασίας**, με χρήση αγωγών και κατάλληλη ανακατεύθυνση Ε/Ε

Εκτέλεση εντολών από το shell

- Το shell διαβάζει και εκτελεί εντολές από το τερματικό
- Μια εντολή μπορεί να οδηγεί στην δημιουργία **πολλών** διεργασιών, οργανωμένων σε ένα **pipeline**
  - αποτελούν μια **νέα ομάδα** (process group)
- Η εκτέλεση μιας εντολής ονομάζεται **εργασία** (**job**)
- Το shell **περιμένει** μέχρι να τερματίσει η εργασία
  - όλες οι διεργασίες της ομάδας
- Αν, κατά την εκτέλεση, το terminal παράγει ένα σήμα, αυτό στέλνεται σε όλη την ομάδα διεργασιών
- Υποστηρίζονται διάφορες εντολές σε επίπεδο εργασιών
  - π.χ. `wait`, `kill` (ο αριθμός της εργασίας δίνεται με `%`)

Βασικός κύκλος εκτέλεσης



Ροή επεξεργασίας – processing pipeline

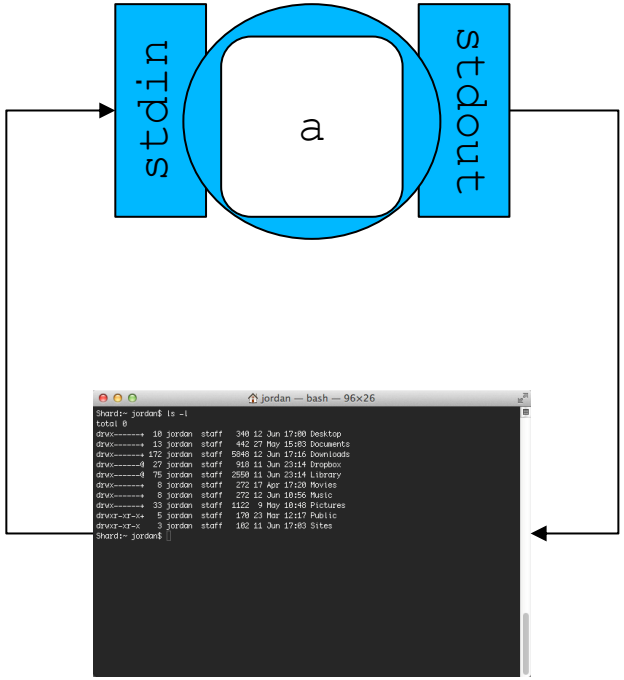
Μια εντολή μπορεί να αποτελείται από **ένα** πρόγραμμα

- Δημιουργείται **μια μοναδική** διεργασία
- Το stdin και stdout δείχνουν στο τερματικό

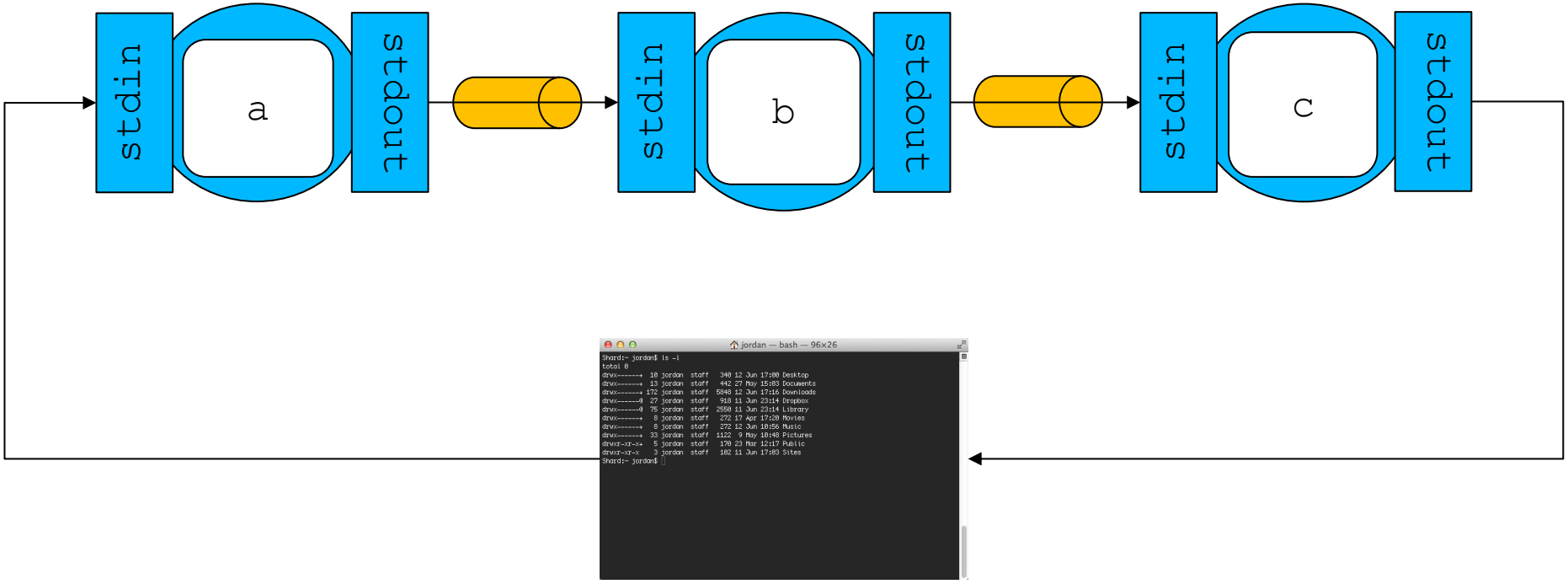
Μια εντολή μπορεί να συνδυάζει **πολλά** προγράμματα σε μια **ροή επεξεργασίας** (**process pipeline**)

- δημιουργούνται **περισσότερες** διεργασίες
  - μια διεργασία για κάθε πρόγραμμα
- Γίνεται **ανακατεύθυνση**, μέσω αγωγών, του stdout της προηγούμενης στο stdin της επόμενης διεργασίας
- Το stdin της πρώτης και το stdout της τελευταίας διεργασίας δεν αλλάζουν – δείχνουν στο τερματικό

```
> a
```


stdin
a
stdout
jordan — bash — 96×26
Shard:~ jordan$ ls -l
total 0
drwx------+ 10 jordan staff 340 12 Jun 17:00 Desktop
drwx------+ 13 jordan staff 442 27 May 15:03 Documents
drwx------+ 172 jordan staff 5848 12 Jun 17:16 Downloads
drwx------@ 27 jordan staff 918 11 Jun 23:14 Dropbox
drwx------@ 75 jordan staff 2550 11 Jun 23:14 Library
drwx------+ 8 jordan staff 272 17 Apr 17:28 Movies
drwx------+ 8 jordan staff 272 12 Jun 10:56 Music
drwx------+ 33 jordan staff 1122 9 May 10:48 Pictures
drwxr-xr-x+ 5 jordan staff 170 23 Mar 12:17 Public
drwxr-xr-x 3 jordan staff 102 11 Jun 17:03 Sites
Shard:~ jordan$

```
> a | b | c
```


stdin
a
stdout
stdin
b
stdout
stdin
c
stdout
jordan — bash — 96×26
Shard:~ jordan$ ls -l
total 0
drwx------+ 10 jordan staff 340 12 Jun 17:00 Desktop
drwx------+ 13 jordan staff 442 27 May 15:03 Documents
drwx------+ 172 jordan staff 5848 12 Jun 17:16 Downloads
drwx------@ 27 jordan staff 918 11 Jun 23:14 Dropbox
drwx------@ 75 jordan staff 2550 11 Jun 23:14 Library
drwx------+ 8 jordan staff 272 17 Apr 17:28 Movies
drwx------+ 8 jordan staff 272 12 Jun 10:56 Music
drwx------+ 33 jordan staff 1122 9 May 10:48 Pictures
drwxr-xr-x+ 5 jordan staff 170 23 Mar 12:17 Public
drwxr-xr-x 3 jordan staff 102 11 Jun 17:03 Sites
Shard:~ jordan$

Πρόβλημα πολυπλεξίας Ε/Ε

- Κάθε εργασία (job) έχει μόνο **μια** διεργασία που διαβάζει από το τερματικό μέσω stdin, και μόνο **μια** διεργασία που γράφει στο τερματικό μέσω stdout
- Το shell μπορεί να εκτελεί πολλές εργασίες (jobs) **ταυτόχρονα** μεταξύ τους
- Μπορεί να γίνει **αυτόματη πολυπλεξία** των διαφορετικών ροών **εξόδου** πάνω στο τερματικό
  - τα δεδομένα προορίζονται για τον (έναν) χρήστη
- Αυτό **δεν** μπορεί να γίνει για τις ροές **εισόδου**
  - δεν είναι ξεκάθαρο για ποια από όλες τις εργασίες που εκτελούνται ανά πάσα στιγμή προορίζονται τα δεδομένα που δίνει ο χρήστης
- Το βασικό ερώτημα είναι: ποια από τις εργασίες είναι στο **επίκεντρο της αλληλεπίδρασης** με τον χρήστη;

Foreground & background processing

- Κατά σύμβαση, το shell εκτελεί τις εργασίες στο προσκήνιο (**foreground**)
- Αν ο χρήστης προσθέσει στο τέλος της εντολής `&`, η εργασία εκτελείται στο παρασκήνιο (**background**)

- Ο χρήστης μπορεί να σταματήσει μια εργασία που τρέχει στο προσκήνιο με `Ctrl-Z` (σήμα `SIGTSTP`)
  - η εργασία αναστέλλεται και πηγαίνει στο παρασκήνιο
- Ο χρήστης μπορεί να επαναφέρει μια εργασία από το παρασκήνιο στο προσκήνιο με `fg <jobnr>`
- Ή να την συνεχίσει στο παρασκήνιο με `bg <jobnr>`

Foreground/background I/O

- 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 **αναστολή** της εκτέλεσης
- Αν μια διεργασία στο παρασκήνιο επιχειρήσει να γράψει στο τερματικό, αυτό συνήθως **πετυχαίνει**
  - αυτό μπορεί να αλλάξει με *`stty tostop`*, οπότε η διεργασία θα δεχτεί το σήμα `SIGTTOU` που κατά σύμβαση οδηγεί σε **αναστολή**
- Μια παρασκηνιακή διεργασία η εκτέλεση της οποίας έχει ανασταλεί λόγω Ε/Ε, **συνεχίζει αυτόματα** την εκτέλεση της όταν επανέλθει στο προσκήνιο

Διάβασμα από το παρασκήνιο

- `echoloop` διαβάζει από το `stdin` και γράφει στο `stdout` για έναν περιορισμένο αριθμό φορών ή μέχρι να τερματιστεί η είσοδος

Δοκιμάζουμε:

```
> ./echoloop 10
…
Ctrl-Z
> bg <jobnr>
…
> fg <jobnr>
…
Ctrl-D
```

Γράψιμο από το παρασκήνιο

- `pingloop` γράφει επαναληπτικά ένα μήνυμα στο `stdout` για έναν περιορισμένο αριθμό φορών

Δοκιμάζουμε:

```
> ./pingloop 20
…
Ctrl-Z
> bg <jobnr>
…
> stty tostop
…
> fg <jobnr>
```

Μερικές ακόμα δοκιμές

Δοκιμάζουμε:

```
> ./pingloop 30 | ./echoloop 40 &
…
> 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
- Το shell δημιουργεί μια διεργασία-παιδί που επίσης τρέχει το shell για να εκτελέσει τις εντολές του script

Φιλοσοφία

- Τα shell scripts **ερμηνεύονται**
  - δεν μεταφράζονται
  - κάθε εντολή ελέγχεται συντακτικά και εκτελείται **ξεχωριστά**, η μια εντολή μετά την άλλη
- Τα shell scripts δεν μπορεί να ανταγωνιστούν σε ταχύτητα εκτέλεσης ένα μεταφρασμένο πρόγραμμα
  - αλλά δεν φτιάχνονται για αυτό το σκοπό
- Το πλεονέκτημα του shell scripting βρίσκεται στο ότι μπορεί κανείς να **συνδυάσει**, εύκολα και ευέλικτα, υπάρχουσες εντολές και έτοιμα προγράμματα

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
$
```

Μεταβλητές

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

- Αναφορά στην **τιμή** της μεταβλητής `X` γίνεται με `$X`

- Οι μεταβλητές **δεν** έχουν αυστηρό 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 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 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 (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 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 μέσα από ένα shell script γίνεται μέσα από **ξεχωριστή** διεργασία
  - που δημιουργείται αποκλειστικά για αυτό το σκοπό

- Η τιμή επιστροφής αποθηκεύεται στην μεταβλητή `$?`
  - συνήθως δείχνει αν το πρόγραμμα εκτελέστηκε επιτυχώς
  - κατά σύμβαση `0` σηματοδοτεί επιτυχή εκτέλεση του προγράμματος

- Με backticks `` `...` `` 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
  echo "Running $C ..."
  $C
  if [ $? -eq "0" ]
  then
    echo "it worked fine!"
  else
    echo "oops, something went wrong."
  fi
  echo "Running $C again, capturing its output ..."
  R=`$C`
  echo "here it is:"
  echo $R
  echo "and again, keeping its format:"
  echo "$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
```

Συναρτήσεις

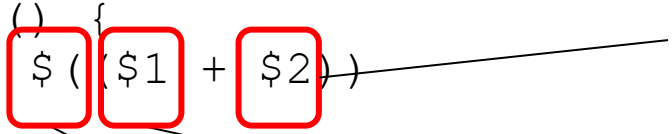
- Δέχονται παραμέτρους όπως το script δέχεται ορίσματα
- Μπορεί να έχουν δικές τους τοπικές μεταβλητές
  - δεν είναι καθολικά ορατές, καταστρέφονται μετά την επιστροφή
- Οι μεταβλητές που ορίζονται «καθολικά», έξω από τις συναρτήσεις, **είναι** ορατές και μέσα στις συναρτήσεις
  - οι όποιες αλλαγές είναι ορατές μετά την επιστροφή
- Μπορεί να επιστρέψουν αποτελέσματα μέσω `echo`
  - ο καλών μπορεί να εκτελέσει την συνάρτηση μέσα από **ξεχωριστή** διεργασία για να αποθηκεύσει τα δεδομένα εξόδου
  - τι γίνεται με τις αναφορές σε καθολικές μεταβλητές; (δοκιμάστε το!)
- Μπορεί να επιστρέψουν μια τιμή μέσω `return`
  - όπως για προγράμματα, **μόνο** έως `255`
  - ο καλών μπορεί να την διαβάσει μέσω της μεταβλητής `$?`

```
#!/bin/sh

myadd1() {
  echo $(($1 + $2))
}

X=2
myadd1 $X 1
echo "retval=$?, X=$X"


Y=$(myadd1 $X 1)
echo "retval=$?, X=$X, Y=$Y"
```



δεύτερο όρισμα συνάρτησης

πρώτο όρισμα συνάρτησης

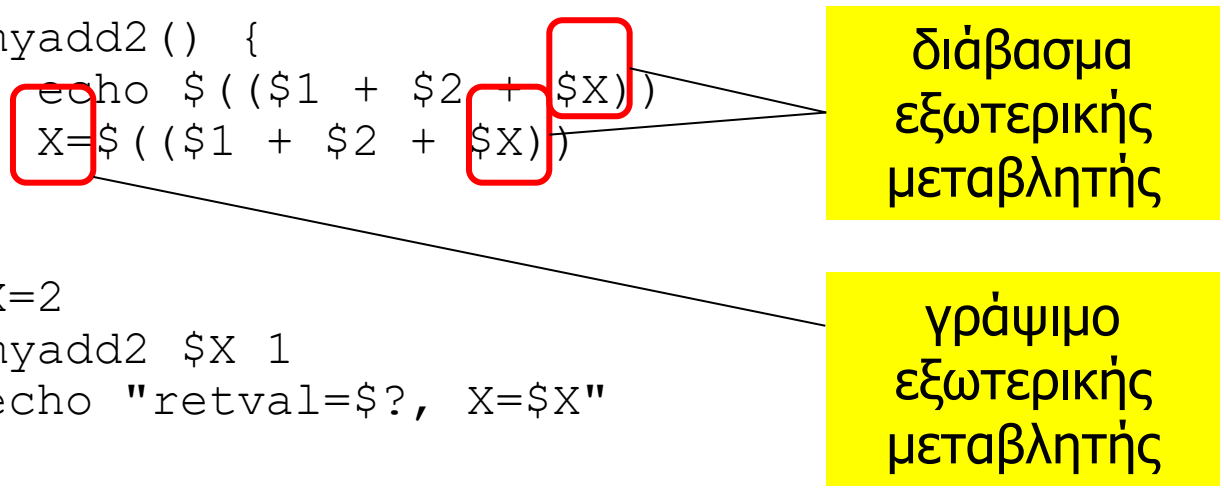
αποτίμηση έκφρασης

```
#!/bin/sh

myadd2() {
  echo $(($1 + $2 + $X))
  X=$(($1 + $2 + $X))
}

X=2
myadd2 $X 1
echo "retval=$?, X=$X"


Y=$(myadd2 $X 1)
echo "retval=$?, X=$X, Y=$Y"
```



```
#!/bin/sh

myadd3() {
  echo $(($1 + $2))
  return $(($1 + $2))
}

X=2
myadd3 $X 1
echo "retval=$?, X=$X"


Y=$(myadd3 $X 1)
echo "retval=$?, X=$X, Y=$Y"
```

ρητή επιστροφή τιμής (με όριο 255)