

# Λειτουργικά Συστήματα Ι

## Ασκήσεις Πράξης

Τμήμα Μηχανικών Πληροφορικής & Υπολογιστών  
Σχολή Μηχανικών



# Σενάρια Φλοιού (shell scripts)

# Σενάρια Φλοιού

Ένα σενάριο φλοιού (**shell script**) είναι μια ακολουθία εντολών φλοιού αποθηκευμένες σε ένα εκτελέσιμο αρχείο κειμένου του οποίου η κλήση έχει ως αποτέλεσμα την εκτέλεση των εντολών με τη σειρά που αυτές εμφανίζονται σ' αυτό.

Μπορεί να περιλαμβάνει δηλώσεις μεταβλητών, ορίσματα γραμμής εντολών, εντολές απόφασης και επανάληψης, διαλογική επικοινωνία με τον χρήστη κ.τ.λ., αποτελεί δηλαδή μία πλήρη διερμηνευόμενη γλώσσα προγραμματισμού. Τα σενάρια φλοιού έχουν ως βασικό σκοπό τον προγραμματισμό αυτοματοποιημένων εργασιών και είναι οι “καλύτεροι φίλοι” ενός διαχειριστή.

# Σενάρια Φλοιού

Ο προκαθορισμένος φλοιός κάθε χρήστη ορίζεται στο αρχείο **/etc/passwd** και συγκεκριμένα στο τελευταίο πεδίο κάθε εγγραφής:

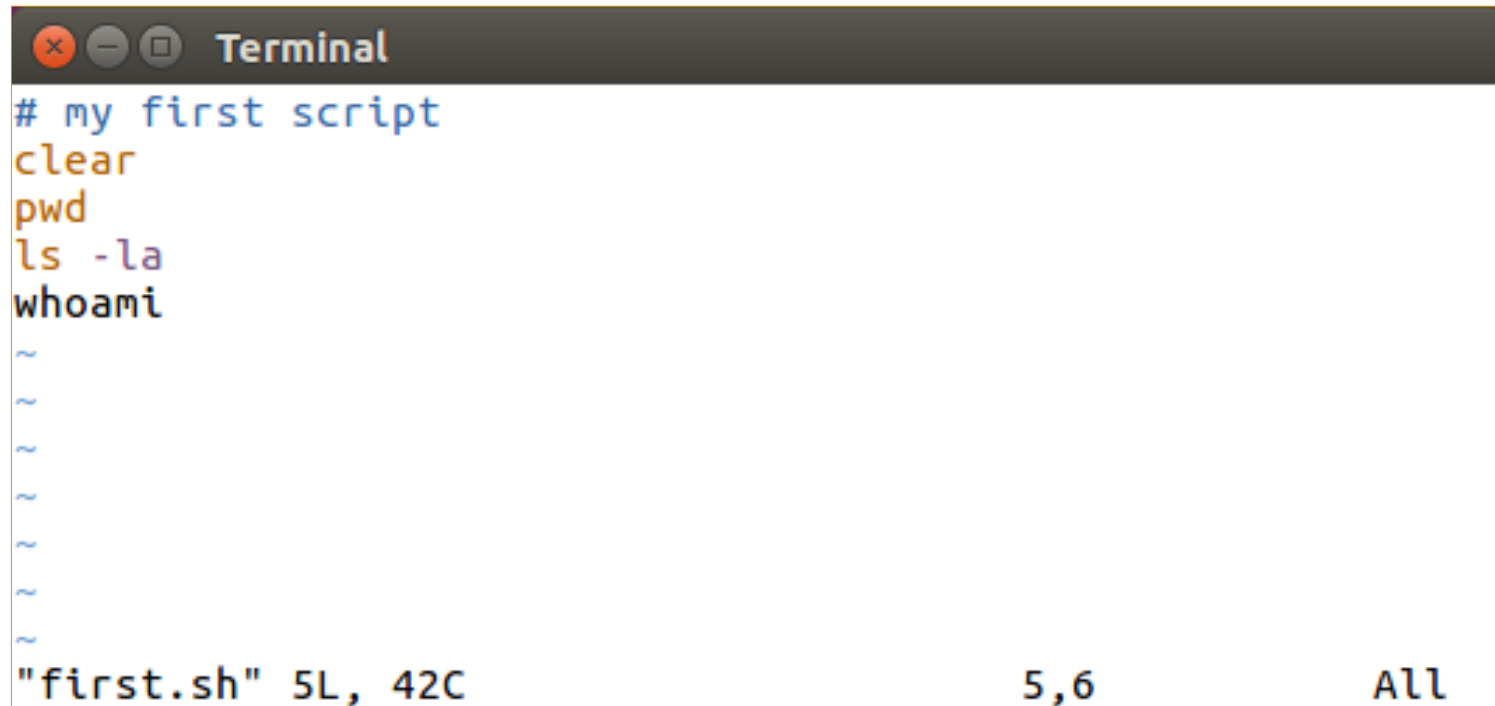
**nemo:x:1001:1001:Captain Nemo,,,:/home/nemo:/bin/bash**

Η κλήση ενός νέου φλοιού γίνεται καλώντας τον επιθυμητό φλοιό από τη γραμμή εντολών του τερματικού:

**\$ ksh** # execute Korn shell

Το τερματισμός ενός φλοιού και η επιστροφή στον προηγούμενο γίνεται με την εντολή **exit** ή με τον συνδυασμό πλήκτρων **[Ctrl]+[D]** που στέλνει ένα σήμα “end of file” σε οποιοδήποτε πρόγραμμα διαβάσει από την πρότυπη είσοδο.

# Το Πρώτο μας Σενάριο Φλοιού

A terminal window titled "Terminal" with standard macOS window controls (red, yellow, green buttons). The terminal displays the execution of a script named "first.sh". The script content is: "# my first script", "clear", "pwd", "ls -la", and "whoami". The output shows the current directory as "~" (home) on five separate lines. At the bottom of the terminal, a status bar indicates the file "first.sh" is 5 lines long and 42 characters, with line 5, column 6 highlighted in orange, and the word "All" is visible on the right.

```
Terminal
# my first script
clear
pwd
ls -la
whoami
~
~
~
~
~
~
~
"first.sh" 5L, 42C 5,6 All
```

Η επέκταση **sh** του αρχείου είναι απλά μία σύμβαση και παραπέμπει σε σενάρια φλοιού. Θα μπορούσε να μην υπάρχει ή να είναι οτιδήποτε άλλο.

# Εκτέλεση Σεναρίου

Η εκτέλεση του script γίνεται με την εντολή:

```
$ bash first.sh
```

Η εντολή αυτή καλεί ένα νέο κέλυφος (subshell) το οποίο εκτελεί το αρχείο εντολών που του δίνεται ως όρισμα. Οι εντολές εκτελούνται σειριακά.

Αν το αρχείο δεν βρίσκεται στον τρέχοντα κατάλογο θα πρέπει να προσδιορίσουμε την πλήρη (απόλυτη ή σχετική) διαδρομή προς αυτό:

```
$ bash /home/nemo/first.sh
```

# Εκτέλεση Σεναρίου

Ένας εναλλακτικός τρόπος εκτέλεσης είναι να δώσουμε στο αρχείο δικαιώματα εκτέλεσης και να το καλέσουμε με το όνομά του:

```
$ chmod u+x first.sh      # executable permission
```

```
$ ./first.sh              # execute script
```

ή πιο απλά χωρίς τον προσδιορισμό του (τρέχοντος) καταλόγου `./` εφόσον ο τρέχον κατάλογος έχει προστεθεί στη μεταβλητή περιβάλλοντος **PATH**:

```
$ first.sh
```

# Εκτέλεση Σεναρίου

Αν θέλουμε τα script που έχουν δικαίωμα εκτέλεσης να μην εκτελούνται από τον τρέχοντα φλοιό αλλά από έναν διαφορετικό φλοιό, πρέπει στην πρώτη γραμμή του αρχείου να προσθέσουμε μία οδηγία **shebang** που αποτελείται από την ακολουθία χαρακτήρων **#!** και την απόλυτη διαδρομή προς τον επιθυμητό φλοιό. Για παράδειγμα, για να εκτελούνται τα script από τον C shell, η πρώτη γραμμή του αρχείου θα πρέπει να είναι της μορφής:

**#!/bin/csh**

Η παραπάνω σύμβαση ισχύει και για διερμηνευόμενες γλώσσες προγραμματισμού, όπως είναι η Perl και η Python.



# Εκτέλεση Σεναρίου

Εάν δεν θέλουμε να εκτελεστεί το σενάριο από έναν καινούργιο φλοιό αλλά από τον τρέχοντα φλοιό, χρησιμοποιούμε την εντολή **source**:

```
$ source first.sh
```

ή ισοδύναμα

```
$ . first.sh                                # source
```

Και σ' αυτήν την περίπτωση το σενάριο δε χρειάζεται δικαιώματα εκτέλεσης. Η λειτουργία αυτή αναφέρεται ως "sourcing" και είναι πολύ χρήσιμη όταν διαπραγματευόμαστε μεταβλητές ή συναρτήσεις.

# Τιμή εξόδου ή επιστροφής

- Κάθε πρόγραμμα που εκτελείται από τον φλοιό επιστρέφει μία ακέραια τιμή σ' αυτόν, η οποία είναι γνωστή ως τιμή εξόδου ή τιμή επιστροφής (**exit/return code**).
- Στην περίπτωση των σεναρίων του φλοιού, η τιμή εξόδου είναι η τιμή της τελευταίας εντολής εφόσον δεν έχει καθοριστεί διαφορετικά. Η τιμή αυτή περιέχεται στην ειδική μεταβλητή `?` του φλοιού, την οποία μπορούμε να αποτιμήσουμε για να αποφασίσουμε τις περαιτέρω ενέργειες.
- Σύμφωνα με το πρότυπο POSIX τα προγράμματα επιστρέφουν την τιμή **0** για επιτυχή εκτέλεση και έναν ακέραιο από **1** μέχρι το **255** σε άλλη περίπτωση.

# Τιμή εξόδου ή επιστροφής

```
$ cal 13 2019
```

```
cal: 13 is neither a month number (1..12) nor a name
```

```
$ echo $?
```

```
64
```

Για να επιστρέψουμε μία συγκεκριμένη τιμή εξόδου από ένα σενάριο φλοιού χρησιμοποιούμε την εντολή **exit n**, όπου n ο αριθμός που θέλουμε να επιστρέψουμε. Στο επόμενο παράδειγμα επιστρέφουμε την τιμή 1 από το script *exit.sh* με περιεχόμενο:

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

```
exit 1
```

# Μεταβλητές Φλοιού

Η δημιουργία μεταβλητών στα σενάρια φλοιού γίνεται προφανώς με τον ίδιο τρόπο, την εκχώρηση δηλαδή μιας τιμής με τη χρήση του τελεστή ανάθεσης:

```
$ myvar=Hi
```

```
$ echo $myvar
```

```
Hi
```

Θα πρέπει να έχουμε πάντα υπόψη μας πώς οι μεταβλητές ενός φλοιού είναι **τοπικές**. Αυτό σημαίνει πώς αν καλέσουμε έναν νέο φλοιό (subshell) αυτός δε θα έχει πρόσβαση στις μεταβλητές του γονικού φλοιού. Προφανώς ισχύει και το αντίστροφο.

# Μεταβλητές Φλοιού

Για να κάνουμε διαθέσιμη μία μεταβλητή σε κάθε νέο φλοιό που θα καλέσουμε στη συνέχεια πρέπει να χρησιμοποιήσουμε την εντολή **export**:

```
$ export myvar
```

```
$ bash # subshell
```

```
$ echo $myvar
```

```
Hi # variable is exported in new process
```

# Παράμετροι Θέσης

- Οι παράμετροι θέσης (positional parameters) είναι μία σειρά ειδικών μεταβλητών που περιέχουν τα ορίσματα της γραμμής εντολών.
- Στις παραμέτρους αυτές, όπως και στις υπόλοιπες ειδικές μεταβλητές μπορούμε να κάνουμε μόνο αναφορά και όχι να αναθέσουμε κάποια τιμή. Τα ορίσματα εντολών ακολουθούν το όνομα του script κατά την κλήση του και επιτρέπουν να καλούμε ένα σενάριο φλοιού με διαφορετικές τιμές κάθε φορά.
- Η παράμετρος **0** αντιστοιχεί στο όνομα του script, ενώ οι **1, 2, 3 ... 9** αντιστοιχούν στο πρώτο, δεύτερο, τρίτο και ένατο όρισμα αντίστοιχα.

# Παράμετροι Θέσης

- Οι μεταβλητές \* και @ αντιπροσωπεύουν όλες τις παραμέτρους θέσης εκτός της 0. Η διαφορά τους έγκειται όταν περικλείονται σε διπλά εισαγωγικά, οπότε η μεταβλητή \* περιέχει όλα τα ορίσματα ομαδοποιημένα ως μία παράμετρο, ενώ η @ περιέχει όλα τα ορίσματα, ένα για κάθε παράμετρο.
- Τέλος, η μεταβλητή # περιέχει το πλήθος των ορισμάτων θέσης μη συμπεριλαμβανόμενης της 0. Η αναφορά σ' όλες τις παραπάνω μεταβλητές γίνεται με το σύμβολο \$ όπως και σε όλες τις μεταβλητές.

# Παράμετροι Θέσης - Παράδειγμα

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

```
# parameters.sh - positional parameters
```

```
echo There are $# arguments to $0: $*
```

```
echo First argument: $1
```

```
echo Second argument: $2
```

```
echo Third argument: $3
```

Ακολουθεί ένα παράδειγμα κλήσης του parameters.sh:

```
$ bash parameters.sh a b "1 2 3"
```

```
There are 3 arguments to parameters.sh: a b 1 2 3
```

```
First argument: a
```

```
Second argument: b
```

```
Third argument: 1 2 3
```



# shift

Η ενσωματωμένη εντολή **shift** του φλοιού μετατοπίζει τις τιμές παραμέτρων θέσης κατά μία θέση προς τα αριστερά μειώνοντας ταυτόχρονα και το πλήθος τους.

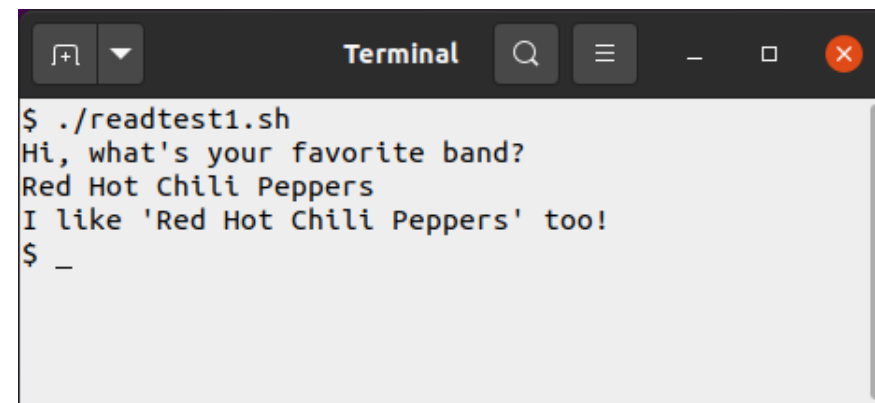
```
#!/bin/bash
# shifting.sh - shift
echo '$0:' $0
shift
echo '$1:' $1
shift
echo '$2:' $2
```

```
$ bash shifting.sh 1 2 3 4
$0: shifting.sh
$1: 2
$2: 4
```

# read

Η ενσωματωμένη εντολή **read** του φλοιού επιτρέπει το διάβασμα τιμών σε μία ή περισσότερες μεταβλητές από το πληκτρολόγιο (προκαθορισμένη είσοδο). Συνήθως προηγείται μία εντολή **echo** με την κατάλληλη προτροπή:

```
#!/bin/bash  
echo Hi, what\'s your favorite band?  
read band  
echo I like \'$band\' too!
```

A terminal window titled "Terminal" with standard window controls (minimize, maximize, close) and search, menu, and zoom icons. The terminal shows the execution of a script named "readtest1.sh". The script prompts the user with "Hi, what's your favorite band?". The user enters "Red Hot Chili Peppers". The script then outputs "I like 'Red Hot Chili Peppers' too!". The prompt "\$ \_" is visible at the bottom of the terminal.

```
$ ./readtest1.sh  
Hi, what's your favorite band?  
Red Hot Chili Peppers  
I like 'Red Hot Chili Peppers' too!  
$ _
```

# read

Η επιλογή **-p prompt** εμφανίζει ένα μήνυμα στην οθόνη (προκαθορισμένη έξοδο) χωρίς να γίνει αλλαγή γραμμής και η επιλογή **-s** (silent mode) αποτρέπει την αντήχηση στην οθόνη των χαρακτήρων που πληκτρολογούνται.

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

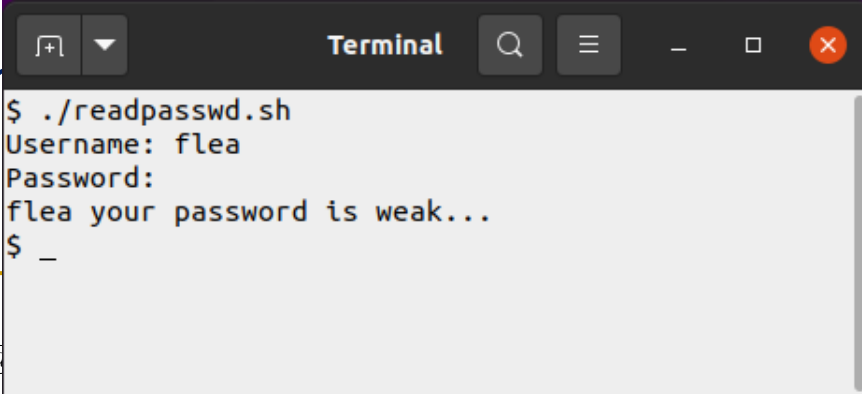
```
# readpasswd.sh - read password
```

```
read -p "Username: " uservar
```

```
read -sp "Password: " passvar
```

```
echo
```

```
echo $uservar your password
```

A terminal window titled "Terminal" with standard window controls. It shows the execution of a script named readpasswd.sh. The script prompts for a username, which is entered as "flea". It then prompts for a password in silent mode. After the password is entered, the script outputs "flea your password is weak..." and returns to the shell prompt.

```
$ ./readpasswd.sh
Username: flea
Password:
flea your password is weak...
$ _
```

# declare

Με την ενσωματωμένη εντολή **declare** μπορούμε να δηλώσουμε μεταβλητές και συναρτήσεις, να ορίσουμε τις ιδιότητές τους και να εμφανίσουμε τις τιμές τους.

Επιλογή	Λειτουργία
-a	Η μεταβλητή είναι πίνακας με αριθμοδείκτες (array)
-A	Η μεταβλητή είναι συσχετιζόμενος πίνακας (associative array)
-f	Τα ονόματα χειρίζονται ως ονόματα συναρτήσεων και όχι μεταβλητών (function)
-i	Ακέραια μεταβλητή (integer)
-p	Εμφάνιση πληροφοριών (print)
-r	Μεταβλητή μόνο για ανάγνωση (readonly)
-x	Η μεταβλητή εξάγεται (export)

# declare

```
$ declare -r var=1963          # readonly
$ echo $var
1963
$ var=1969
bash: var: readonly variable
$ declare -p                   # list all variables
declare -- BASH="/bin/bash"
declare -ir BASHPID
...
declare -r var="1963"
```

# declare

```
$ declare -a array      # explicit declaration
$ array[0]=Good         # first element index zero
$ array[1]=Morning
$ array[2]=Tux
$ array[7]=!            # not need to be contiguous
$ declare -p array      # print array
declare -a array='([0]="Good" [1]="Morning"
[2]="Tux" [7]="!")'
```

# declare

```
$ echo ${array[0]}           # first element
Good

$ echo ${array[-1]}         # last element
!

$ echo ${array[*]}          # * to display all elements
Good Morning Tux !

$ echo ${#array[@]}         # number of elements,
4                            # length

$ echo ${#array}            # ${#array[0]} length of
4                            # first element
```

# declare

```
$ declare -A map      # declare associative array
$ map[cat]=Katze
$ map[dog]=Hund
$ map["polar bear"]=Eisbär
$ echo ${map[dog]}
Hund
$ echo ${map[polar bear]}
Eisbär
$ echo "${!map[@]}"    # print all keys
dog cat polar bear
```



# declare

```
$ declare -A array          # simulate 2D array
$ array[0,0]=1
$ array[0,1]=2
$ array[1,0]=3
$ array[1,1]=4
$ echo ${array[0,0]} ${array[1,1]}
1 4
$ echo echo "${!arr[@]}"    # print all keys
1,0 1,1 0,1 0,0
```

# unset

```
$ var=1969
```

```
$ unset var
```

```
$ echo $var
```

```
# prints nothing
```

```
$ rgb=(red green blue)
```

```
# new array, no declare
```

```
$ unset rgb[1]
```

```
# delete second element
```

```
$ echo ${rgb[*]}
```

```
# print all elements
```

```
red blue
```

```
$ unset rgb
```

```
# destroy array rgb
```

# printf

```
$ printf "Hello %s\n" Alice
```

```
Hello Alice
```

```
$ printf "Hello %s\n" Alice Bob Carol
```

```
Hello Alice
```

```
Hello Bob
```

```
Hello Carol
```

```
$ printf "%10s\n" Alice
```

```
    Alice
```

```
$ printf "%s is %d years old\n" Bob 57
```

```
Bob is 57 years old
```

```
$ printf "%c\n" Alice      # only first character
```

```
A
```

# printf

```
$ printf "The price is %f\n" 4.99
```

```
The price is 4.990000
```

```
$ printf "π is %.4f\n" 3.14159265
```

```
π is 3.1416
```

```
$ printf "Current date/time is %s\n" "$(date)"
```

```
Current date/time is Sat Nov 18 21:41:25 CET 2023
```

```
$ printf "Linux version %s\n" "$(uname -r)"
```

```
Linux version 6.2.0-36-generic
```

```
$ printf -v name "%s" Alice          # output assigned
```

```
$ echo $name                          # to variable
```

```
Alice
```

# Αριθμητική Φλοιού

```
$ declare -i var=3           # integer declaration
$ var=var+1; echo $var      # no need for let
4
$ var=Hi; echo $var         # invalid value assignment
0
$ var="3*7"; echo $var      # string with arithmetic
21
$ var="28/5"; echo $var     # round down next lowest
5
```

# Αριθμητική Φλοιοῦ

Ο φλοιός Bash διαθέτει ενσωματωμένες δυνατότητες εκτέλεσης αριθμητικών πράξεων ακεραίων, οπότε δεν είναι απαραίτητο να χρησιμοποιούμε εξωτερικές εντολές. Αν για κάποιο λόγο πρέπει τα σενάρια να είναι συμβατά προς τα πίσω, μπορούμε να κάνουμε χρήση της εντολής **expr**.

```
$ expr 5 + 2
```

```
7
```

```
$ var=1
```

```
$ var=`expr $var + 1`           # increase var by 1
```

```
$ echo $var
```

```
2
```

# Αριθμητική Φλοιού

```
$ echo $(( 3 + 4 ))          # expression evaluation
```

```
7
```

```
$ (( var = 3 + 5 )); echo $var    # assignment
```

```
8
```

```
$ var=1
```

```
$ var=$(( $var + 2 )); echo $var
```

```
3
```

```
$ var=$(( var + 3 )); echo $var    # $ is optional
```

```
6
```

```
$ echo $(( var1 = 7 + $(( var2 = 4 + 5 )) ))
```

```
16
```

# Αριθμητική Φλοιοῦ

Τελεστής	Λειτουργία
++ --	Επιθεματική αύξηση ή μείωση της τιμής μιας μεταβλητής κατά 1
++ --	Προθεματική αύξηση ή μείωση της τιμής μιας μεταβλητής κατά 1
+ -	Μοναδιαίο συν / πλην
! ~	Λογική άρνηση / Συμπλήρωμα ως προς 1
**	Ύψωση στη δύναμη
* / %	Πολλαπλασιασμός / Διαίρεση / Υπόλοιπο ακέραιας διαίρεσης
+ -	Πρόσθεση / Αφαίρεση
<< >>	Ολίσθηση αριστερά / Ολίσθηση δεξιά
<= >= < >	Μικρότερο ή ίσο / Μεγαλύτερο ή ίσο / Μικρότερο / Μεγαλύτερο



# Αριθμητική Φλοιοῦ

Τελεστής	Λειτουργία
== !=	Ίσο / Όχι ίσο
&	Δυαδικό AND
^	Δυαδικό XOR
	Δυαδικό OR
&&	Λογικό AND
	Λογικό OR
?:	Τριαδική συνθήκη
= *= /= %= += -= <<= >>= &= ^=  =	Ανάθεση τιμής Πολλαπλασιασμός / Διαίρεση και ανάθεση Υπόλοιπο ακέραιας διαίρεσης και ανάθεση Πρόσθεση / Αφαίρεση και ανάθεση Αριστερή / Δεξιά Ολίσθηση και ανάθεση Δυαδικό And / Xor / Or και ανάθεση
,	διαχωριστής λίστας εκφράσεων

# Αριθμητική Φλοιοῦ

```
$ let var=var+1; echo $var      # no spaces around  
2                               # operator  
  
$ let "ram = 8 * 1024"; echo $ram  
8192  
  
$ let "rom = ram / 32"; echo $rom  
256  
  
$ let "var = 10 % 4"; echo $var  # modulo  
2  
  
$ let "var = 16 << 3"; echo $var # left shifting  
128
```

# Ερωτήσεις

