UOPR - Μάθημα 4ο

1. APT (Advanced Package Tool)
2. Διαμόρφωση δίσκου και κατατμήσεις δίσκου
3. Systemd
4. Εισαγωγή στο Bash scripting

---

1. APT (Advanced Package Tool)

Ενημερώσεις

- `sudo -i`
- `apt update` (προβάλλονται οι νέες ενημερώσεις)
- `apt list --upgradable` (βλέπουμε αναλυτικά τι θα εγκατασταθεί στο σύστημά μας)
- `apt dist-upgrade` (γίνεται λήψη και εγκατάσταση των νέων ενημερώσεων)

Σε άλλες διανομές η εντολή είναι μία αντί για 2, πχ (dnf update)

Εγκατάσταση ενός πακέτου (παράδειγμα με git)

- `apt list git`
- `apt search git`
- `apt install git`
- `apt remove git`

Πώς προσθέτουμε ένα ppa - Personal Package Archive (παράδειγμα με LibreOffice)

- `sudo add-apt-repository ppa:libreoffice/ppa`
- `sudo apt update`
- `sudo apt dist-upgrade`

`ls -la /etc/apt/sources.list.d` (για να δούμε από ποιες πηγές κάνουμε λήψη τα πακέτα)

---

2. Διαμόρφωση δίσκου και κατατμήσεις δίσκου

2α] Τι είναι το Σύστημα Αρχείων (Filesystem);

Ένα σύστημα αρχείων είναι μια δομή που διαχειρίζεται την αποθήκευση μέσα σε μια κατάτμηση, ανεξάρτητα από το αν αυτό βρίσκεται σε έναν εικονικό ή φυσικό δίσκο.

Τύποι συστήματος αρχείων	**Περιγραφή**
ext4	Το πιο διαδεδομένο και προεπιλεγμένο σύστημα αρχείων σε πολλές διανομές Linux. Σταθερό, γρήγορο και αξιόπιστο, αλλά με περιορισμένες προηγμένες δυνατότητες (π.χ. δεν υποστηρίζει snapshots ή ενσωματωμένο RAID).
XFS	Υψηλής απόδοσης σύστημα αρχείων, σχεδιασμένο για μεγάλους όγκους δεδομένων και ταχύτητα, ιδιαίτερα σε παράλληλες εγγραφές. Δεν υποστηρίζει snapshots εγγενώς και δεν είναι ιδανικό για μικρά αρχεία.
ZFS	Ισχυρό σύστημα αρχείων με ενσωματωμένο RAID, snapshots, έλεγχο ακεραιότητας δεδομένων και deduplication. Κατάλληλο για servers και αποθήκευση μεγάλων δεδομένων, αλλά πιο απαιτητικό σε μνήμη.
BTRFS	Σύστημα αρχείων next-gen με υποστήριξη για snapshots, ενσωματωμένο RAID, compression, subvolumes και έλεγχο ακεραιότητας. Ακόμα θεωρείται "πειραματικό".

Τύποι συστήματος αρχείων

2β] Ονόματα συσκευών

Τα ονόματα των συσκευών συνδέονται με το λειτουργικό σύστημα μέσω modules του πυρήνα και έχουν τις παρακάτω γενικές παραμέτρους:

/dev/xxyn, where:

- **xx** είναι ο τύπος της συσκευής (**sd** = scsi, **hd** = ide, **vd** = virtual disk)
- **y** είναι ένα γράμμα που υποδηλώνει τη σειρά ανακάλυψης από το σύστημα (**sda** = first, **sdb** = second, etc.)
- **n** είναι ο αριθμός της κατάτμησης (partition) (όταν λείπει εννοούμε ολόκληρο τον δίσκο, **1** = 1ο partition, **2** = 2ο partition, κτλ)

Όνομα συσκευής	**Τύπος συσκευής**
/dev/sda **/dev/sdb**	Συσκευή που χρησιμοποιεί τον SCSI driver (πχ, δίσκος SATA, USB, κτλ)
/dev/nvme0n1p1	NVMe SSD

Όνομα συσκευής	Τύπος συσκευής
/dev/vda **/dev/vdb**	Εικονικός δίσκος

Όνομα και τύπος συσκευών σε Linux

Όπως αναφέρθηκε, το όνομα του δίσκου αποδίδεται κατά την εκκίνηση του συστήματος. Το σύστημα συνήθως αποδίδει το ίδιο όνομα στην ίδια συσκευή, επειδή αυτή εντοπίζεται με την ίδια σειρά σε κάθε εκκίνηση. Ωστόσο, αν αλλάξει ο αριθμός των συσκευών, το όνομα μπορεί να διαφέρει.

Οι δίσκοι περιέχουν κατατμήσεις. Μια κατάτμηση (partition) είναι ένας διαχωρισμός ενός δίσκου σε λογικά ανεξάρτητα τμήματα.

Το τρέχον σύστημα κατάτμησης σε Linux ονομάζεται **GPT (GUID Partition Table)**. Το GPT υποστηρίζει έως και **128 κατατμήσεις ανά δίσκο**.

Οι κατατμήσεις μάς επιτρέπουν να χωρίσουμε τον αποθηκευτικό χώρο ενός δίσκου σε λογικά ξεχωριστά τμήματα. Έτσι μας δίνεται η δυνατότητα να εφαρμόσουμε ειδικές παραμέτρους σε επιμέρους περιοχές του δίσκου για λόγους ασφαλείας και απόδοσης.

2γ] Partition table, format σε ext4 & fstab

Ακολούθως, διαμορφώνουμε έναν νέο δίσκο μέσω τερματικού και στη συνέχεια τον προσθέτουμε στο αρχείο ρυθμίσεων "fstab" ώστε να προσαρτάται αυτόματα, κατά την εκκίνηση του Η/Υ.

Για να προβάλλουμε όλες τις συσκευές (δίσκους) στο σύστημά μας:
`sudo fdisk -l` ή `lsblk`. Έστω ότι ο δίσκος που θέλουμε να διαμορφώσουμε βρίσκεται στο `/dev/sdb`

Δημιουργούμε partition table σε GPT. Χρησιμοποιούμε το εργαλείο `gdisk`

- `sudo gdisk /dev/sdb`
- Πληκτρολογούμε `o` για partition table σε gpt
- και μετά `w` για να το γράψει
- Επιβεβαιώνουμε ότι ο δίσκος μας έχει GPT Partition Table με `sudo fdisk -l`

Κάνουμε το 1ο partition sdb1

- Τρέχουμε εκ νέου το `sudo gdisk /dev/sdb`
- Πληκτρολογούμε `n` για new partition
- Partition number: 1 (Σε GPT μπορούμε να κάνουμε μέχρι 128 partitions)

- First sector: Πατάμε απλώς enter, αν το partition μας θέλουμε να ξεκινάει από την αρχή του δίσκου.
- Last sector: Πληκτρολογούμε πόσο μεγάλο θέλουμε να είναι το partition, πχ για 20GB, πληκτρολογούμε: `+20G`
- Current type is 8300: Πατάμε απλώς enter
- Command: `w` για να το γράψει
- Τέλος, πατάμε `y` για την επιβεβαίωση
- Βλέπουμε ότι έχει δημιουργηθεί το partition sdb1 πληκτρολογώντας `lsblk` ή `fdisk -l`

Διαμορφώνουμε σε ext4 το partition sdb1
`sudo mkfs.ext4 /dev/sdb1`

Κάνουμε προσάρτηση (mount) το partition sdb1
`mkdir /mnt/disk1` (δημιουργούμε έναν κατάλογο με όνομα `disk1`)
`sudo mount /dev/sdb1 /mnt/disk1` (Κάνουμε mount το sdb1 στο /mnt/disk1)
Γενικά καλό είναι να κάνουμε mount έναν δίσκο ή usb stick σε έναν κενό κατάλογο.
Ελέγχουμε ότι είναι mounted, πληκτρολογώντας `df -Th`

`sudo umount /mnt/disk1` (για αποπροσάρτηση του δίσκου - unmount)

Πώς το βάζουμε στο fstab

- σε μια νέα καρτέλα του τερματικού, πληκτρολογούμε `sudo blkid /dev/sdb1` για να δούμε το UUID (Universally Unique Identifier του partition) του δίσκου. Αντιγράφουμε το UUID.
- `sudo nano /etc/fstab` και προσθέτουμε την γραμμή:
  `UUID=1234-ABCD-XXXXXXXX /mnt/disk1 ext4 defaults 0 2`
  μαζί με ένα σχόλιο από πάνω, αν επιθυμούμε
- Τέλος, από την επόμενη εκκίνηση του συστήματος το partitition θα γίνεται automounted.

---

3. Systemd

3α] Τι είναι το systemd;

Το systemd είναι μια σουίτα βασικών δομικών στοιχείων για ένα σύστημα Linux. Θα μπορούσαμε να πούμε ότι είναι η ραχοκοκαλιά του συστήματος.

Παρέχει έναν διαχειριστή συστήματος και υπηρεσιών που εκτελείται ως `PID 1`, ο οποίος με την σειρά του εκκινεί όλα τα υπόλοιπα στοιχεία (όπως υπηρεσίες) του συστήματος.

Προσφέρει δυνατότητες παράλληλης εκτέλεσης, εκκίνηση daemons κατόπιν ζήτησης, παρακολουθεί τις διεργασίες, διατηρεί σημεία προσάρτησης (mounts) και αυτόματης

προσάρτησης, και υλοποιεί μια πολυεπίπεδη λογική ελέγχου υπηρεσιών βασισμένη σε εξαρτήσεις.

Περιλαμβάνει επίσης έναν daemon καταγραφής, βοηθητικά προγράμματα για τον έλεγχο βασικής διαμόρφωσης του συστήματος, όπως το όνομα υπολογιστή (hostname), την ημερομηνία, το locale, τη διατήρηση μιας λίστας των συνδεδεμένων χρηστών και λογαριασμών συστήματος, τη βασική διαχείριση δικτύου, τον συγχρονισμό δικτυακού χρόνου (NTP), την διαχείριση καταγραφών και επίλυσης ονομάτων (resolv) και πολλά ακόμα.

3β] systemctl

Αποτελεί το κύριο εργαλείο/εντολή διαχείρισης, ενώ η σύνταξή του έχει ως εξής:
``systemctl [OPTIONS...] COMMAND ...

Βασική διαχείριση υπηρεσιών
Ένα `<unit>` μπορεί να αντιστοιχεί σε μια υπηρεσία, πχ την υπηρεσία συστήματος του cron

```
systemctl status
systemctl status <unit>
systemctl enable <unit>
systemctl disable <unit>
systemctl start <unit>
systemctl stop <unit>
systemctl restart <unit>
```

Πληροφορίες συστήματος

```
systemctl list-units
systemctl list-unit-files
```

Αποσφαλμάτωση
systemctl --state=failed

3γ] systemd components

Πέραν του systemctl υπάρχουν πρόσθετες εντολές για την διαχείριση του συστήματος. Μερικές από αυτές είναι οι εξής:

Εντολή	Πληροφορίες
systemd-path	Εμφανίζει τις διαδρομές που χρησιμοποιούνται από το systemd για την αναζήτηση αρχείων και υπηρεσιών.
systemd-sysusers	Δημιουργεί χρήστες και ομάδες σύμφωνα με τις ρυθμίσεις που καθορίζονται σε αρχεία συστήματος.

Εντολή	Πληροφορίες
systemd-resolve	Διαχειρίζεται την ανάλυση DNS και παρέχει πληροφορίες σχετικά με την κατάσταση των DNS.
systemd-tmpfiles	Δημιουργεί, διαγράφει ή καθαρίζει προσωρινά αρχεία και καταλόγους σύμφωνα με τις ρυθμίσεις που καθορίζονται σε αρχεία ρυθμίσεων.
systemd-ask-password	Ζητάει από τον χρήστη να εισάγει έναν κωδικό πρόσβασης, συνήθως για την προστασία ευαίσθητων πληροφοριών.
systemd-machine-id-setup	Δημιουργεί ή ρυθμίζει το μοναδικό αναγνωριστικό μηχανής (machine ID) για το σύστημα.
systemd-cat	Κατευθύνει την έξοδο από άλλες εντολές στο systemd journal, επιτρέποντας την καταγραφή τους.
systemd-mount	Διαχειρίζεται την προσάρτηση (mount) συσκευών και καταλόγων στο σύστημα.
systemd-umount	Διαχειρίζεται την αποσύνδεση (unmount) συσκευών και καταλόγων από το σύστημα.
systemd-analyze	Παρέχει πληροφορίες σχετικά με την εκκίνηση του συστήματος, όπως το χρόνο εκκίνησης και την κατάσταση των υπηρεσιών.
systemd-analyze blame	Εμφανίζει μια λίστα με τις υπηρεσίες που καθυστέρησαν την εκκίνηση του συστήματος, ταξινομημένες κατά τον χρόνο που χρειάστηκαν για να ξεκινήσουν.

3δ] journalctl

Είναι ένα εργαλείο που χρησιμοποιείται για την προβολή και την ανάλυση των καταγραφών (logs) του systemd journal. Επιτρέπει στους χρήστες να φιλτράρουν, να αναζητούν και να εμφανίζουν καταγραφές από διάφορες υπηρεσίες και χρονικές περιόδους, διευκολύνοντας την παρακολούθηση και την αντιμετώπιση προβλημάτων στο σύστημα.

Εμφάνιση όλων των μηνυμάτων που ταιριάζουν στο PATTERN:
`journalctl --grep=PATTERN`

Εμφάνιση όλων των μηνυμάτων από την τρέχουσα εκκίνηση του συστήματος:
`journalctl -b`

Ακολουθεί την καταγραφή για νέα μηνύματα:
`journalctl -f`

Εμφάνιση όλων των μηνυμάτων από την ημερομηνία (και προαιρετικά ώρα):
`journalctl --since="2025-05-15 18:17:16"`

Εμφάνιση όλων των μηνυμάτων από μία συγκεκριμένη διεργασία:
`journalctl _PID=1`

4. Εισαγωγική στο Bash scripting

Το Bash scripting είναι η διαδικασία δημιουργίας και εκτέλεσης scripts (προγραμμάτων) που γράφονται στη γλώσσα προγραμματισμού Bash.

Χαρακτηριστικά του Bash Scripting:

- **Αυτοματοποίηση**: Το Bash scripting επιτρέπει την αυτοματοποίηση επαναλαμβανόμενων εργασιών, όπως η διαχείριση αρχείων, η εκτέλεση εντολών συστήματος και η εκτέλεση ρουτινών συντήρησης.
- **Δομές Ελέγχου**: Υποστηρίζει δομές ελέγχου ροής, όπως if, for, while, case, που επιτρέπουν την εκτέλεση διαφορετικών εντολών ανάλογα με τις συνθήκες.
- **Μεταβλητές**: Μπορείτε να δηλώσετε και να χρησιμοποιήσετε μεταβλητές για να αποθηκεύσετε δεδομένα και να τα επαναχρησιμοποιήσετε στο script.
- **Συναρτήσεις**: Υποστηρίζει τη δημιουργία συναρτήσεων, που επιτρέπουν την οργάνωση του κώδικα σε επαναχρησιμοποιήσιμα μπλοκ.
- **Ενσωμάτωση Εντολών**: Μπορείτε να εκτελείτε εντολές του συστήματος και να χρησιμοποιείτε τις εξόδους τους μέσα στο script.

Χρήσεις του Bash Scripting:

- **Διαχείριση Συστήματος**: Αυτοματοποίηση εργασιών διαχείρισης συστήματος, όπως η δημιουργία αντιγράφων ασφαλείας, η παρακολούθηση πόρων και η εγκατάσταση λογισμικού.
- **Επεξεργασία Αρχείων**: Διαχείριση και επεξεργασία αρχείων κειμένου, όπως η αναζήτηση, η αντικατάσταση και η μορφοποίηση.
- **Δημιουργία Εργαλείων**: Δημιουργία εργαλείων και scripts για την εκτέλεση συγκεκριμένων εργασιών ή διαδικασιών.
- **Διαχείριση Δικτύου**: Αυτοματοποίηση εργασιών δικτύου, όπως η παρακολούθηση συνδέσεων και η διαχείριση διακομιστών.

4α] shebang

Είναι μια ειδική γραμμή που τοποθετείται στην αρχή ενός script και καθορίζει ποιο πρόγραμμα θα χρησιμοποιηθεί για την εκτέλεση του script, πχ

```
#!/bin/bash
#!/bin/sh
#!/usr/bin/env python
#!/usr/bin/python3
```

4β] echo

Είναι μια βασική εντολή στο Bash και σε άλλα shell που χρησιμοποιείται για την εκτύπωση κειμένου ή μεταβλητών

Παράδειγμα 1

```
#!/bin/bash
# A simple script
echo "Γεια!"
echo "Τα λέμε!"
```

4γ] Ορίσματα γραμμής εντολών (Command line arguments)

Συμβολίζονται με το `$` και έπειτα τον αριθμό του ορίσματος κατά σειρά, πχ
$1, $2, $3 ... κτλ
Η πρώτη, δεύτερη, τρίτη, κτλ

Τα ορίσματα διαχωρίζονται από το κενό χαρακτήρα.

Παράδειγμα 1

```
#!/bin/bash
# Ένα απλό script
#
echo "Γεια!"
echo "Έδωσες πρώτα την τιμή:" $1
echo "Έδωσες μετά την τιμή:" $2
echo "Τα λέμε!"
```

Εκτελούμε: `./myscript.sh 10 20`
Αποτέλεσμα:

```
Γεια!
Έδωσες πρώτα την τιμή: 10
Έδωσες μετά την τιμή: 20
Τα λέμε!
```

Παράδειγμα 2

```
#!/bin/bash
# Ένα απλό script αντιγραφής αρχείου σε νέο
cp $1 $2
# Εμφάνιση πληροφοριών για 2ο αρχείο
```

```
echo "Πληροφορίες αρχείου2:" $2
ls -lh $2
```

Εκτελούμε: `./mycopy.sh ./file1.data ./file2.data
Αποτέλεσμα:

```
Πληροφορίες αρχείου2: ./file2.data
-rw-rw-r-- 1 greeklug greeklug 36 Μαΐ  17 12:53 ./file2.data
```

4δ] Μεταβλητές

Συμβολίζονται με ένα `$ και, όπως σε άλλες γλώσσες προγραμματισμού, με ένα όνομα. Κατά τον ορισμό τους δεν χρησιμοποιούμε το` $`, πχ

```
myvar="hello"
```

Το bash έχει κάποιες προορισμένες **ειδικές μεταβλητές**:

Μεταβλητή	**Πληροφορίες**
$0	Περιέχει το όνομα του Bash script.
$1	$9 - Τα ορίσματα που είδαμε στην ενότητα 4γ.
$#	Ο αριθμός των ορισμάτων που δόθηκαν στο Bash script.
$@	Όλα τα ορίσματα ως ενιαία τιμή που δόθηκαν στο Bash script.
$?	Η κατάσταση εξόδου της διεργασίας εκτέλεσης (επιστρέφει 0 αν το script εκτελέστηκε επιτυχώς, οτιδήποτε διαφορετικό υποδηλώνει σφάλμα εκτέλεσης).
$$	Το PID του τρέχοντος script.
$USER	Το όνομα χρήστη που εκτελεί το script.
$HOSTNAME	Το hostname της συσκευής που τρέχει το script.
$SECONDS	Ο αριθμός των δευτερολέπτων που πέρασαν από την εκκίνηση εκτέλεσης του script.
$RANDOM	Επιστρέφει έναν διαφορετικό τυχαίο αριθμό κάθε φορά που καλείται.
$LINENO	Επιστρέφει τον αριθμό γραμμής που εκτελείται την εκάστοτε στιγμή στο Bash script.

Παράδειγμα 1

```
#!/bin/bash
# Ένα απλό script με μεταβλητές
```

```
myvariable=Hello
anothervar=Mike
echo $myvariable $anothervar
echo
sampledir=/etc
ls $sampledir | sort -h
```

Εκτελούμε: ./simplevariables1.sh
Αποτέλεσμα:

```
Hello Mike

adduser.conf
adjtime
alsa
alternatives
anacrontab
apache2
apg.conf
apm
apparmor
apparmor.d
apt
audit
...
```

Γενικό Παράδειγμα

```
#!/bin/bash
# Ένα απλό script
#
echo "Γεια σου $USER"
echo "Τρέχεις το script:" $0
echo "Το PID εκτέλεσης είναι:" $$
echo "Έδωσες πρώτα την τιμή:" $1
echo "Έδωσες μετά την τιμή:" $2
echo "Συνολικά έδωσες:" $# "τιμές"
echo "Τα λέμε!"
```

Εκτελούμε: ./simplevariables2.sh 20 40
Αποτέλεσμα:

```
Γεια σου mike
Τρέχεις το script: ./myscript.sh
Το PID εκτέλεσης είναι: 84713
Έδωσες πρώτα την τιμή: 20
Έδωσες μετά την τιμή: 40
Συνολικά έδωσες: 2 τιμές
Τα λέμε!
```

4ε] Εισαγωγικά

Όταν περιβάλλουμε το περιεχόμενό μας με εισαγωγικά, υποδεικνύουμε στο Bash ότι το περιεχόμενο θα πρέπει να θεωρείται ως ένα ενιαίο στοιχείο.

Μπορείτε να χρησιμοποιήσετε μονά εισαγωγικά (') ή διπλά εισαγωγικά (").

- Τα μονά εισαγωγικά θα θεωρήσουν κάθε χαρακτήρα κυριολεκτικά.
- Τα διπλά εισαγωγικά κάνουν υποκατάσταση μεταβλητών, εμφανίζουν δηλαδή το περιεχόμενο των μεταβλητών

Το ακόλουθο script:

```
#!/bin/sh
myvar=hello
echo "τα διπλά εισαγωγικά δίνουν $myvar"
echo 'τα μονά εισαγωγικά δίνουν $myvar'
```

θα επιστρέψει το παρακάτω αποτέλεσμα:

```
τα διπλά εισαγωγικά δίνουν hello
τα μονά εισαγωγικά δίνουν $myvar
```

4ζ] Υποκατάσταση εντολών (Command Substitution)

Μπορούμε να αποθηκεύσουμε την έξοδο μιας εντολής σε μια μεταβλητή με την ακόλουθη μορφή: `variable=$( command )`

πχ
`myvar=$( ls )`

Γενικό Παράδειγμα

```
#!/bin/bash
# Ένα απλό script
#
```

```
path=$( pwd )
echo "Γεια σου $USER"
echo "Βρίσκεσαι στην διαδρομή:" $path
echo "Τρέχεις το script:" $0
echo "Το PID εκτέλεσης είναι:" $$
echo "Έδωσες πρώτα την τιμή:" $1
echo "Έδωσες μετά την τιμή:" $2
echo "Συνολικά έδωσες:" $# "τιμές"
echo "Τα λέμε!"
```

Εκτελούμε: `./myscript.sh Maria Nikos
Αποτέλεσμα:

```
Γεια σου mike
Βρίσκεσαι στην διαδρομή: /home/mike
Τρέχεις το script: ./myscript.sh
Το PID εκτέλεσης είναι: 87672
Έδωσες πρώτα την τιμή: Maria
Έδωσες μετά την τιμή: Nikos
Συνολικά έδωσες: 2 τιμές
Τα λέμε!
```

4η] Αριθμητική επέκταση (Arithmetic Expansion)

Αν θέλουμε να κάνουμε μαθηματικές πράξεις με τις μεταβλητές θα πρέπει να τις ορίσουμε με την ακόλουθη μορφή: `variable=$(( ... ))`

πχ
sum=$((num1 + num2))

Γενικό Παράδειγμα

```
#!/bin/bash
# Ένα απλό script
#
path=$( pwd )
sum=$(($1 + $2))
echo "Γεια σου $USER"
echo "Βρίσκεσαι στην διαδρομή:" $path
echo "Τρέχεις το script:" $0
echo "Το PID εκτέλεσης είναι:" $$
echo "Έδωσες πρώτα την τιμή:" $1
echo "Έδωσες μετά την τιμή:" $2
```

```
echo "Συνολικά έδωσες:" $# "τιμές"
echo "Η πρόσθεση κάνει:" $sum
echo "Τα λέμε!"
```

Εκτελούμε: `./myscript.sh 12 33
Αποτέλεσμα:

```
Γεια σου mike
Βρίσκεσαι στην διαδρομή: /home/mike
Τρέχεις το script: ./myscript.sh
Το PID εκτέλεσης είναι: 88395
Έδωσες πρώτα την τιμή: 12
Έδωσες μετά την τιμή: 33
Συνολικά έδωσες: 2 τιμές
Η πρόσθεση κάνει: 45
Τα λέμε!
```

4θ] Εξαγωγή μεταβλητών

Αν θέλουμε οι μεταβλητές να είναι διαθέσιμες σε υποδιεργασίες/νέα script που θα εκτελεστούν από το κύριο script μας, μπορούμε να τις "εξάγουμε" με την ακόλουθη μορφή:

`export var1`

4ι] Εισαγωγή χρήστη (User Input)

Μπορούμε να ζητήσουμε την εισαγωγή δεδομένων από τον χρήστη κατά την εκτέλεση του script. Αυτό γίνεται με χρήση της εντολής:
`read var1`
Η εντολή έχει και μερικά βοηθητικά options, όπως:
`-s | δεν τυπώνεται με echo η είσοδος από το τερματικό`
`-p | εμφάνιση PROMPT χωρίς αλλαγή γραμμής πριν την ανάγνωση δεδομένων`

Παράδειγμα 1

```
#!/bin/bash
# Ρωτάμε το όνομα του χρήστη
echo "Γειά σου, πως σε λένε;"
read varname
echo "Χάρηκα για την γνωριμία" $varname
```

Παράδειγμα 2

```
#!/bin/bash
# Ρωτάμε τα στοιχεία πρόσβασης του χρήστη
read -p 'Username: ' uservar
read -sp 'Password: ' passvar
echo
echo "Ευχαριστούμε" $uservar "που μας έδωσες τα στοιχεία σου! :)"
```

Παράδειγμα 3

```
#!/bin/bash
# Ανάγνωση πολλαπλών εισόδων
echo "Τι χρώματα σου αρέσουν;"
read color1 color2 color3
echo "Το πρώτο χρώμα ήταν:" $color1
echo "Το δεύτερο χρώμα ήταν:" $color2
echo "Το τρίτο χρώμα ήταν:" $color3
```

Γενικό Παράδειγμα

```
#!/bin/bash
# Ένα απλό script
#
path=$( pwd )
#sum=$(($1 + $2))
echo "Γεια σου $USER"
echo "Βρίσκεσαι στην διαδρομή:" $path
echo "Τρέχεις το script:" $0
echo "Το PID εκτέλεσης είναι:" $$
echo "#############"
echo "Δώσε μου δύο αριθμούς:"
read num1 num2
sum=$(($num1 + $num2))
echo "#############"
#echo "Έδωσες πρώτα την τιμή:" $1
#echo "Έδωσες μετά την τιμή:" $2
#echo "Συνολικά έδωσες:" $# "τιμές"
echo "Η πρόσθεση κάνει:" $sum
echo "Τα λέμε!"
```

Εκτελούμε: `./myscript.sh GNU Linux
Εισάγουμε: 33 55
Αποτέλεσμα:

```
Γεια σου mike
Βρίσκεσαι στην διαδρομή: /home/mike
Τρέχεις το script: ./myscript.sh
Το PID εκτέλεσης είναι: 92127
#############
Δώσε μου δύο αριθμούς:
33 55
#############
Η πρόσθεση κάνει: 88
Τα λέμε!
```

4κ] Λογικές συνθήκες

Όπως σε άλλες γλώσσες προγραμματισμού υπάρχουν διαθέσιμοι λογικοί έλεγχοι για τον χειρισμό της ροής εκτέλεσης. Τυπικό παράδειγμα αποτελεί η συνθήκη `if`.

Σύνταξη:

```
if [ <some check> ]
then
<commands>
fi
```

Πολλαπλές συνθήκες:

```
if [ <some check> ]
then
<commands>
elif [ <some check> ]
then
<commands>
else
<commands>
fi
```

Οι έλεγχου περιλαμβάνουν τελεστές ελέγχου:

Τελεστής	**Πληροφορίες**
==	Επιστρέφει true αν τα strings είναι ίσα
!=	Επιστρέφει true αν τα strings δεν είναι ίσα

Τελεστής	Πληροφορίες
-n	Επιστρέφει true αν τα strings δεν ειναι κενά (null)
-z	Επιστρέφει true αν τα strings ειναι κενά (null)
! EXPRESSION	Η EXPRESSION είναι false.
STRING1 = STRING2	STRING1 είναι ίσο με STRING2
STRING1 != STRING2	STRING1 δεν είναι ισο με STRING2
-d FILE	το FILE υπάρχει και είναι directory
-e FILE	το FILE υπάρχει
-r FILE	το FILE υπάρχει και έχει το δικαίωμα ανάγνωσης (read)
-s FILE	το FILE υπάρχει και το μέγεθός του είναι μεγαλύτερο από 0 (δεν είναι κενό)
-w FILE	το FILE υπάρχει και έχει το δικαίωμα εγγραφής (write)
-x FILE	το FILE υπάρχει και έχει το δικαίωμα εκτέλεσης (execute)

Τελεστής	Πληροφορίες
-eq	Equal
-ge	Greater Than or Equal
-gt	Greater Than
-le	Less Than or Equal
-lt	Less Than
-ne	Not Equal

Συνδυασμός ελέγχων με:
ισχύει το A και το B: `and ή &&`
ισχύει το A ή το B: `or ή ||`

Παράδειγμα 1

```
#!/bin/bash
# Ένα τυπικό if
if [ $1 -gt 100 ]
then
echo "Αυτός είναι ένας μεγάλος αριθμός!"
fi
```

**Παράδειγμα 2

```
#!/bin/bash
# Ένα ανεπτυγμένο if
if [ $1 -lt 100 ]
then
echo "Αυτός είναι ένας μικρός αριθμός!"
elif [ $1 -gt 100 ] && [ $1 -lt 1000 ]
then
echo "Αυτός είναι ένας μέσος αριθμός!"
else
echo "Αυτός είναι ένας τεράστιος αριθμός!"
fi
```

Γενικό Παράδειγμα

```
#!/bin/bash
# Ένα απλό script
#
path=$( pwd )
#sum=$(($1 + $2))
echo "Γεια σου $USER"
echo "Βρίσκεσαι στην διαδρομή:" $path
echo "Τρέχεις το script:" $0
echo "Το PID εκτέλεσης είναι:" $$
echo "#############"
echo "Δώσε μου δύο αριθμούς:"
read num1 num2

if [[ -n "$num1" ] && [ -n "$num2" ]]
then
    sum=$(($num1 + $num2))
else
    echo "Error! Δεν έδωσες δύο αριθμούς!"
    exit 1
fi

echo "#############"
#echo "Έδωσες πρώτα την τιμή:" $1
#echo "Έδωσες μετά την τιμή:" $2
#echo "Συνολικά έδωσες:" $# "τιμές"
echo "Η πρόσθεση κάνει:" $sum
echo "Τα λέμε!"
```

Εκτελούμε: `./myscript.sh
Εισάγουμε: 44 66
Αποτέλεσμα:

```
Γεια σου mike
Βρίσκεσαι στην διαδρομή: /home/mike
Τρέχεις το script: ./myscript.sh
Το PID εκτέλεσης είναι: 98290
#############
Δώσε μου δύο αριθμούς:
44 66
#############
Η πρόσθεση κάνει: 110
Τα λέμε!
```

4λ] Case Statements

Είναι μια δομή ελέγχου που χρησιμοποιείται για να ελέγξει μια μεταβλητή και να εκτελέσει διαφορετικές εντολές ανάλογα με την τιμή της. Είναι χρήσιμες όταν θέλουμε να συγκρίνουμε μια μεταβλητή με πολλές πιθανές τιμές και να εκτελέσουμε διαφορετικές ενέργειες για κάθε περίπτωση.

Σύνταξη:

```
case <variable> in
<pattern 1>)
<commands>
;;
<pattern 2>)
<other commands>
;;
esac
```

```
#!/bin/bash
# Ένα παράδειγμα case
case $1 in
start)
echo "starting"
;;
stop)
echo "stoping"
;;
restart)
```

```
echo "restarting"
;;
*)
echo "no data"
;;
esac
```

Γενικό Παράδειγμα
cases: prosthesi afairesi pollaplasiasmos diairesi

```
#!/bin/bash
# Ένα απλό script
#
path=$( pwd )
#sum=$(($1 + $2))
echo "Γεια σου $USER"
echo "Βρίσκεσαι στην διαδρομή:" $path
echo "Τρέχεις το script:" $0
echo "Το PID εκτέλεσης είναι:" $$
echo "#############"
echo "Δώσε μου δύο αριθμούς:"
read num1 num2

if [ -n "$num1" ] && [ -n "$num2" ]
then
    case $1 in
    prosthesi)
    sum=$(($num1 + $num2))
    praxi="Η πρόσθεση"
    ;;
    afairesi)
    sum=$(($num1 - $num2))
    praxi="Η αφαίρεση"
    ;;
    pollaplasiasmos)
    sum=$(($num1 * $num2))
    praxi="Ο πολλαπλασιασμός"
    ;;
    diairesi)
    sum=$(($num1 / $num2))
    praxi="Η διαίρεση"
    ;;
    *)
```

```
    echo "Δεν έδωσες κάποιο όρισμα μαθηματικής πράξης!!!"
    exit 1
    esac
else
    echo "Error! Δεν έδωσες δύο αριθμούς!"
    exit 1
fi

echo "#############"
#echo "Έδωσες πρώτα την τιμή:" $1
#echo "Έδωσες μετά την τιμή:" $2
#echo "Συνολικά έδωσες:" $# "τιμές"
echo $praxi "κάνει:" $sum
echo "Τα λέμε!"
```

Εκτελούμε: `./myscript.sh pollaplasiasmos
Εισάγουμε: 22 99
Αποτέλεσμα:

```
Γεια σου mike
Βρίσκεσαι στην διαδρομή: /home/mike
Τρέχεις το script: ./myscript.sh
Το PID εκτέλεσης είναι: 99439
#############
Δώσε μου δύο αριθμούς:
22 99
#############
Ο πολλαπλασιασμός κάνει: 2178
Τα λέμε!
```

4μ] Επίλογος

Στις παραπάνω ενότητες περιγράψαμε μερικά εισαγωγικά και βασικά στοιχεία που μας εισάγουν στον προγραμματισμό Bash scripting.

Αποτελεί μια πλούσια και ευέλικτη γλώσσα, με πολλές δυνατότητες που δεν μπορούν να καλύψουμε σε αυτές τις σημειώσεις. Υπάρχουν πολλές ακόμα δομές ελέγχου, όπως οι επαναληπτικές δομές (`while` , `for` , `until`), καθώς και η χρήση συναρτήσεων που μπορούν να σας βοηθήσουν να δημιουργήσετε πιο σύνθετα scripts.

Αν ενδιαφέρεστε να εμβαθύνετε περαιτέρω στο Bash scripting, σας προτείνουμε να εξερευνήσετε περισσότερους πόρους, όπως βιβλία και διαδικτυακά μαθήματα. Η πρακτική είναι το κλειδί για την εκμάθηση, οπότε μην διστάσετε να πειραματιστείτε με τα scripts σας και να ανακαλύψετε τις δυνατότητες που προσφέρει το Bash.

Καλή εξερεύνηση του κόσμου του Bash scripting!