Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Μάθημα

Λειτουργικά Συστήματα

Λειτουργικό Σύστημα Linux

Σημειώσεις Εργαστηρίου

Α. Παυλίδης, Εργαστηριακό Διδακτικό Προσωπικό
Πανεπιστημίου Πειραιώς

Φθινόπωρο 2012

Περιεχόμενα

ΕΙΣΑΓΩΓΗ 5

1. Ιστορικό. 5
2. Χαρακτηριστικά. 5
2. Πρόσβαση στο σύστημα. 6
3. Ο φλοιός (κέλυφος). 7
4. Η μορφή των εντολών. 7

ΣΥΣΤΗΜΑ ΑΡΧΕΙΩΝ 11

1. Η δομή του συστήματος αρχείων. 11
2. Περιήγηση στους καταλόγους - διαδρομές. 12
3. Βασικές εντολές για λειτουργίες σε αρχεία. 14
4. Βασικές εντολές για λειτουργίες σε καταλόγους. 17
5. Μεταχαρακτήρες. 17
6. Τα περιεχόμενα των αρχείων. 18

Εργαστηριακές Ασκήσεις 21

ΕΞΟΥΣΙΟΔΟΤΗΣΕΙΣ 25

1. Εισαγωγή 25
2. Τύποι πρόσβασης. 25
3. Κατηγορίες πρόσβασης. 25
4. Αριθμητικός συμβολισμός εξουσιοδοτήσεων. 26
5. Αλλαγή των εξουσιοδοτήσεων 27
6. Προκαθορισμένες εξουσιοδοτήσεις (default permissions). 28
7. Επιλογή αναδρομής για εντολές αρχείων 29

Εργαστηριακές Ασκήσεις 31

ΦΙΛΤΡΑ – ΑΝΑΚΑΤΕΥΘΥΝΣΕΙΣ - ΣΩΛΗΝΩΣΕΙΣ 33

1. Εισαγωγή 33
2. Φίλτρα. 34
3. Ανακατευθύνσεις. 36
4. Σωληνώσεις. 38
5. Σύνθετοι Συνδυασμοί. 38

Εργαστηριακές Ασκήσεις 41

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΦΛΟΙΟΥ – SHELL SCRIPTS 43

1. Εισαγωγή 43
2. Τρόπος εκτέλεσης – Απλά παραδείγματα. 43
3. Προγράμματα αρχικοποίησης. 44
4. Μεταβλητές. 45
5. Υπολογισμοί και παραστάσεις. 49
6. Έλεγχος ροής προγράμματος. 52
7. Συναρτήσεις και άλλες δυνατότητες. 59

Ασκήσεις. 63

ΔΙΕΡΓΑΣΙΕΣ και ΜΝΗΜΗ 65

1. Εισαγωγή 65
2. Χρονοδρομολόγηση ή χρονοπρογραμματισμός. 65
3. Εντολές ελέγχου διεργασιών. 68
4. Προγράμματα στο προσκήνιο και παρασκήνιο. 71
5. Φυσική και εικονική μνήμη. 73

Εργαστηριακές Ασκήσεις 77

ΒΙΒΛΙΟΓΡΑΦΙΑ 83

Λειτουργικό Σύστημα Linux
Ενότητα Ι
ΕΙΣΑΓΩΓΗ

1. Ιστορικό.

Το λειτουργικό σύστημα Linux ανήκει στην οικογένεια των λειτουργικών συστημάτων Unix. Το Unix αναπτύχθηκε στα τέλη της δεκαετίας του 1960 και σήμερα έχει εξελιχθεί διαθέτοντας πολλές παραλλαγές ανάλογα με την αρχιτεκτονική για την οποία προορίζεται. Γνωστές παραλλαγές είναι τα SunOS και Solaris της Sun Microsystems, HP-UX της Hewlett Packard, AIX της IBM, IRIX της Silicon Graphics, Mac OS X της Apple.
Το Linux ανήκει στην κατηγορία του λογισμικού *ανοικτού κώδικα* (Open Source) και η χρήση του χ είναι ελεύθερη χωρίς την πληρωμή άδειας. Η εξέλιξή του υποστηρίζεται από χιλιάδες χρήστες σε όλον τον κόσμο που συνεχώς βελτιώνουν και κάνουν προσθήκες στον πηγαίο κώδικά του (σε γλώσσα C).
Υπάρχουν διάφορες διανομές Linux (π.χ. Debian, Fedora, Ubuntu, Knopix) που διαφοροποιούνται λόγω των διαφορετικών προγραμμάτων υποστήριξης (συμπεριλαμβανομένου και του γραφικού περιβάλλοντος) που ενσωματώνουν γύρω από το βασικό *πυρήνα* (**kernel**). Ένα καλό σημείο εκκίνησης για να βρει κανείς τη διανομή της αρεσκείας του είναι ο ιστότοπος http://www.linux.org.

2. Χαρακτηριστικά.

Το Linux, όπως και όλα τα Λ.Σ. της κατηγορίας Unix, προσφέρει μία σειρά από χαρακτηριστικά που το κάνουν ελκυστικό για εφαρμογές που απαιτούν αξιοπιστία και ασφάλεια. Μερικά από αυτά χαρακτηριστικά είναι :
α) *Πολυπρογραμματισμός* (**multiprogramming**), *πολυχρησία* (**multiuser**) και *πολυεπεξεργασία* (**multiprocessing**) : Σε έναν υπολογιστή μπορούν να τρέχουν πολλές εφαρμογές (διεργασίες) και επί πλέον να γίνεται χρήση του υπολογιστή από πολλούς χρήστες ταυτόχρονα. Εκτός από αυτά, μπορεί ο υπολογιστής να είναι βασισμένος σε πολλούς επεξεργαστές, αντί για έναν, και να γίνεται κατανομή του φορτίου σε όλους τους επεξεργαστές. Όλα αυτά είναι εφικτά με τη χρήση τεχνικών χρονοδρομολόγησης (scheduling) που είναι μέρος του πυρήνα του Λ.Σ.
β) Διαχείριση μνήμης – Εικονική μνήμη : Το Linux ενοποιεί την υπάρχουσα φυσική μνήμη (RAM) μαζί με ένα κομμάτι μνήμης του σκληρού δίσκου προκειμένου να φαίνεται στο σύστημα μία ενιαία μεγάλη *εικονική μνήμη* (**virtual memory**). Αυτό γίνεται με εναλλαγές σελίδων και υλοποιείται στον πυρήνα του Λ.Σ.
γ) Ασφάλεια συστήματος αρχείων : Ο κάθε χρήστης μπορεί να προστατεύει τα αρχεία του από μη εξουσιοδοτημένη πρόσβαση από άλλους χρήστες που χρησιμοποιούν το σύστημα μέσω εξουσιοδοτήσεων που υπάρχουν σε κάθε αρχείο.
δ) Ασφάλεια διεργασιών : Μία διεργασία ενός χρήστη δεν μπορεί να παρέμβει σε άλλη διεργασία κάποιου άλλου χρήστη.
ε) Ευκολία διαδικτύωσης με τη χρήση ενσωματωμένων στη διανομή των πρωτοκόλλων TCP/IP.
στ) Υλοποίηση σε πολλές αρχιτεκτονικές υπολογιστών (εκτός της Intel IA32), για παράδειγμα m68k της Motorola, Alpha της DEC, Sparc της SUN, Itanium (Intel 64 bits) αλλά και σε επεξεργαστές ενσωματωμένων συστημάτων.
ζ) Αλληλεπίδραση με το χρήστη μέσω παραθυρικού γραφικού περιβάλλοντος (πιθανώς διαφορετικό ανάλογα με τη διανομή).
η) Αλληλεπίδραση μέσω γραμμής εντολών για απεριόριστο έλεγχο του συστήματος από το διαχειριστή.

θ) Μεγάλες δυνατότητες που πηγάζουν από το γεγονός ότι η φιλοσοφία του Unix είναι τα διάφορα προγράμματα υποστήριξής του να έχουν τη δυνατότητα αλληλεπίδρασης μεταξύ τους (ανακατευθύνσεις - σωληνώσεις – φίλτρα).

2. Πρόσβαση στο σύστημα.

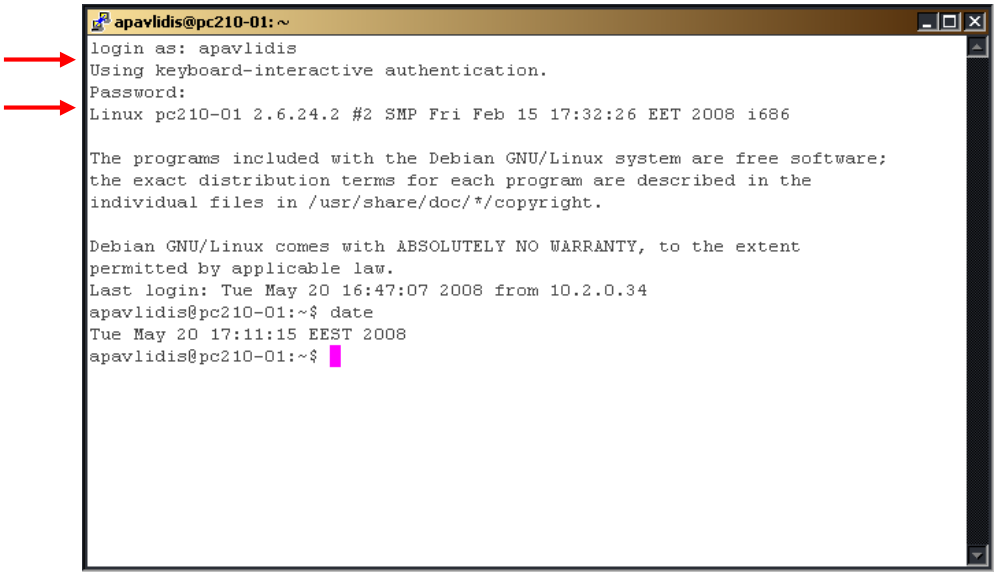
Για να αποκτήσει κάποιος πρόσβαση σε ένα σύστημα που τρέχει Λ.Σ. Linux πρέπει να κάνει *σύνδεση* (**login**) έχοντας διαθέσιμο ένα *λογαριασμό* (**account**) Linux. Ο κάθε λογαριασμός περιλαμβάνει ένα σύνολο από ιδιότητες που μεταξύ άλλων είναι :

- Όνομα *χρήστη* (**username**) – είναι μία λέξη, συνήθως μέχρι οκτώ χαρακτήρες, που επιτρέπει το διαχωρισμό από το σύστημα των διαφόρων χρηστών που μπορούν να έχουν πρόσβαση στο σύστημα. Εσωτερικά, το όνομα χρήστη αντιστοιχεί σε έναν αριθμό που λέγεται *ταυτότητα χρήστη* (**user id – uid**).
- *Συνθηματική λέξη* (**password**) – είναι μία μυστική λέξη που πρέπει να ξέρει μόνο ο χρήστης και επιβεβαιώνει τον ισχυρισμό κάποιου χρήστη προς το σύστημα ότι όντως είναι αυτός. Χρησιμοποιείται κατά τη σύνδεση στο σύστημα.
- *Ομάδα χρήστη* (**user group**) – κάθε χρήστης ανήκει σε μία ομάδα και οι χρήστες που ανήκουν σε μία κοινή ομάδα επιτρέπεται να έχουν κάποιες διαφοροποιήσεις σε θέματα ασφαλείας μεταξύ τους σε σχέση με το τι επιτρέπεται να κάνουν χρήστες που δεν ανήκουν στην ομάδα. Για παράδειγμα μπορεί να έχουν πρόσβαση σε κάποια αρχεία ενώ αυτοί που δεν ανήκουν στην ομάδα να μην έχουν. Και σε αυτή την ιδιότητα αντιστοιχεί ένας αριθμός *ταυτότητας ομάδας* (**group id – gid**).
- *Προσωπικός κατάλογος* (**home directory**) – μία περιοχή στο σύστημα αρχείων στην οποία επιτρέπεται να έχει απεριόριστη πρόσβαση ο χρήστης. Επίσης, είναι η περιοχή την οποία «βλέπει» αμέσως μετά από κάθε σύνδεση ο χρήστης.
- Είδος *φλοιού* ή *κελύφους* (**shell**) – ο τύπος της *διεπαφής γραμμής εντολών* (**command line interface**) μεταξύ χρήστη και συστήματος.

Οι παραπάνω ιδιότητες είναι αποθηκευμένες στις πιο πολλές περιπτώσεις (με πιθανή εξαίρεση τη συνθηματική λέξη) για κάθε χρήστη στο αρχείο `/etc/passwd`. Σε ένα σύστημα Linux υπάρχει πάντα ένας λογαριασμός *διαχειριστή* (**administrator**) με όνομα `root`. Αυτός έχει απεριόριστη εξουσιοδότηση πάνω στο σύστημα.

Η σύνδεση μπορεί να γίνει επί τόπου στο χώρο που βρίσκεται ο υπολογιστής, από ένα απομακρυσμένο τερματικό που είναι συνδεδεμένο με τον υπολογιστή με χρήση σειραϊκής γραμμής (π.χ. τερματικό τύπου VT ή πρόγραμμα Hyperterminal των Windows) ή δικτύου (π.χ. X-terminal), ή από άλλο υπολογιστή που είναι συνδεδεμένος μέσω δικτύου (π.χ. προγράμματα telnet, ssh).

Σε κάθε περίπτωση ο υπολογιστής περιμένει την πληκτρολόγηση του ονόματος χρήστη (μετά το προτρεπτικό μήνυμα `login` ή `login as`) και κατόπιν της συνθηματικής λέξης μετά το μήνυμα `password`. Το password κατά τη διάρκεια που πληκτρολογείται δεν εμφανίζεται (φαίνονται οι χαρακτήρες `*` ή ● ή τίποτα) για να προφυλαχθεί η μυστικότητά του από παράπλευρους θεατές. Η παρακάτω εικόνα δείχνει ένα παράδειγμα εισόδου μέσω δικτύου. Θα πρέπει να σημειωθεί ότι το Linux κάνει διάκριση μεταξύ πεζών και κεφαλαίων χαρακτήρων (case-sensitive)



Η έξοδος από το σύστημα γίνεται πληκτρολογώντας στη γραμμή εντολών την εντολή `logout` ή `exit` ή πατώντας συνδυασμένα τα πλήκτρα `Ctrl-D`. Σε παραθυρικό περιβάλλον η έξοδος από το σύστημα γίνεται επιλέγοντας τη σχετική εντολή από το μενού εντολών.
Ανάλογα με τον τρόπο που έγινε η σύνδεση, ο χρήστης εισάγεται σε γραφικό περιβάλλον (εφ' όσον έχει εγκατασταθεί) ή σε περιβάλλον γραμμής εντολών (βλέπε παραπάνω εικόνα). Οι σημειώσεις αυτές ασχολούνται μόνο με το περιβάλλον της γραμμής εντολών. Σε περίπτωση εισαγωγής σε παραθυρικό περιβάλλον μπορεί να ανοιχτεί ένα περιβάλλον γραμμής εντολών με επιλογή της σχετικής εντολής από το μενού των εντολών.

3. Ο φλοιός (κέλυφος).

Η αλληλεπίδραση του χρήστη με τη γραμμή εντολών γίνεται με τη βοήθεια ενός ερμηνευτή εντολών που λέγεται φλοιός ή κέλυφος (shell). Το πρόγραμμα αυτό ενεργοποιείται σε κάθε σύνδεση με το σύστημα και η κύρια λειτουργία του είναι να εκτελεί τις εντολές που δίνονται από το πληκτρολόγιο και να εμφανίζει τα αποτελέσματά τους στην οθόνη. Ο φλοιός εκτελεί κυκλικά τις εξής λειτουργίες : Εμφανίζει ένα *σύμβολο αναμονής* (**prompt**) και περιμένει είσοδο από το πληκτρολόγιο (`$` στην εικόνα), δέχεται τις πληκτρολογούμενες εντολές (**`date`** στο συγκεκριμένο παράδειγμα) μέχρι να πατηθεί το πλήκτρο `Enter`, ελέγχει συντακτικά τις δοθείσες εντολές, εκτελεί τα σχετικά προγράμματα, εμφανίζει τα αποτελέσματα στην οθόνη (`Tue May 20 17:11:15 EEST 2008`) και περιμένει πάλι νέα είσοδο.
Μία εγκατάσταση Linux συνήθως περιέχει πάνω από έναν φλοιούς και ο κάθε χρήστης μπορεί να επιλέξει τον φλοιό που του ταιριάζει. Ο αρχικός φλοιός ορίζεται όπως έχει αναφερθεί στο αρχείο `/etc/passwd`. Αυτό που διαφοροποιεί τους φλοιούς μεταξύ τους είναι κυρίως ο τρόπος προγραμματισμού τους και όχι τόσο οι εντολές που μπορούν να δεχθούν.
Κάποιοι από τους συνηθισμένους φλοιούς είναι οι : C-shell (`/bin/csh`), Bourne shell (`/bin/sh`), Korn shell (`/bin/ksh`), Bourne Again shell (`/bin/bash`), T-C shell (`/bin/tcsh`). Ο εξ ορισμού φλοιός στο Debian Linux είναι ο Bash και έχει σύμβολο αναμονής για τους απλούς χρήστες το `$` ενώ για τον διαχειριστή το σύμβολο αναμονής είναι το `#`. Πάντως, τα σύμβολα αναμονής είναι εύκολο να αλλάξουν.

4. Η μορφή των εντολών.

Οι εντολές που δέχεται ένας φλοιός διακρίνονται σε *εξωτερικές* και *εσωτερικές*.
Οι εξωτερικές εντολές είναι στην πραγματικότητα το όνομα ενός εκτελέσιμου αρχείου που βρίσκεται κάπου στο σύστημα αρχείων και μπορεί ο φλοιός να το αναζητήσει και να το τρέξει.
Η παρακάτω εξωτερική εντολή εμφανίζει την ώρα και ημερομηνία :

```
$ date
Tue Sep 18 16:45:14 EEST 2007
```

Αν το αρχείο-εντολή δεν είναι άμεσα προσπελάσιμο γιατί η περιοχή που βρίσκεται δεν είναι σε αυτές που έχει οριστεί να ψάχνει ο φλοιός, τότε μπορεί να δοθεί η πλήρης ή η σχετική διαδρομή όπως παρακάτω (η έννοια της διαδρομής θα οριστεί παρακάτω):

```
$ /bin/date
Tue Sep 18 16:45:14 EEST 2007
```

Οι εσωτερικές εντολές είναι ήδη ενσωματωμένες μέσα στο φλοιό. Ένα παράδειγμα εσωτερικής εντολής είναι η **history** που εμφανίζει τις τελευταίες εντολές που έχουν δοθεί στο φλοιό :

```
$ history
...
451  ls
452  postg1
453  cd postg1
454  ls
```

Ακολουθούν μερικά παράδειγμα απλών εντολών με πρώτη την εντολή με την οποία αλλάζει ένας χρήστης τη συνθηματική του λέξη. Στην αρχή ο φλοιός ζητάει το παλιό συνθηματικό και κατόπιν δύο φορές το νέο (επειδή δεν εμφανίζονται οι χαρακτήρες στην οθόνη υπάρχει η περίπτωση να γίνει κάποιο λάθος).

```
$ passwd
Changing password for username
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Οι επόμενες εντολές εμφανίζουν, διαφορετικές πληροφορίες για τους χρήστες που είναι συνδεδεμένοι στον υπολογιστή (μπορεί να είναι πολλοί αφού πρόκειται για πολυχρηστικό περιβάλλον). Φαίνονται πληροφορίες όπως πότε έγινε είσοδος στο σύστημα, από που, πόση χρήση του επεξεργαστή έχει γίνει, τι πρόγραμμα τρέχει εκείνη τη στιγμή ο φλοιός κ.α.

```
$ who
apavlidi pts/0        Sep 30 21:32 (10.2.0.6)
apavlidi pts/1        Sep 30 22:10 (localhost.localdomain)
$ w
  22:11:03 up 1 day, 14:27,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
apavlidi pts/0    10.2.0.6         21:32    0.00s  0.09s  0.02s ssh
apavlidi pts/1    localhost.locald 22:10    0.00s  0.03s  0.01s w
$ finger
Login     Name                 Tty      Idle  Login Time   Office     Office Phone
apavlidis Pavlidis Archimedes  pts/0          Sep 30 21:32 (10.2.0.6)
apavlidis Pavlidis Archimedes  pts/1          Sep 30 22:10 (localhost.localdomain)
```

Η **whoami** δείχνει ποιος έχει κάνει είσοδο στο σύστημα στον τρέχοντα φλοιό από τον οποίο δίνεται η εντολή, η **uname** δίνει το όνομα του λειτουργικού συστήματος και η **hostname** το όνομα του υπολογιστή.

```
$ whoami
apavlidis
$ uname
Linux
```

```
$ hostname
zeus
```

Μεγάλες δυνατότητες όμως δίνονται μέσω της χρήσης *επιλογών* ή *παραμέτρων* (**options**) και *ορισμάτων* (**arguments**) στις εντολές. Μέσω των παραμέτρων αλλάζει λίγο ως πολύ η συμπεριφορά της εντολής, ενώ μέσω των ορισμάτων γίνεται επεξεργασία πάνω στα ορίσματα τα οποία είναι συνήθως, αλλά όχι πάντα, αρχεία.
Η γενική μορφή σύνταξης μίας εντολής με μία ή παραπάνω επιλογές είναι :

```
command -a
command -a -b -c
command -abc
command --word_option
command --word_option1 --word_option2
```

όπου *command* είναι το όνομα της εντολής και *a*, *b*, *c*, *abc*, *word_option*, *word_option1*, *word_option2* είναι οι πιθανές επιλογές της.
Μπορεί δηλαδή να δίνονται μία η παραπάνω επιλογές με ένα χαρακτήρα του οποίου προηγείται η παύλα, ή αν η εντολή το υποστηρίζει να δίνεται μία ή παραπάνω λέξεις των οποίων προηγούνται δύο παύλες.
Ακολουθούν δύο παραδείγματα. Στο πρώτο η εντολή **uname** αλλάζει συμπεριφορά και δίνει πιο λεπτομερή αποτελέσματα για το λειτουργικό σύστημα που έχει εγκατασταθεί, όπως ποιος πυρήνας υπάρχει και πότε κατασκευάστηκε ή για ποιον τύπο επεξεργαστή είναι σχεδιασμένος. Στο δεύτερο παράδειγμα η **who** αντί να αναφέρει ποιοι είναι συνδεδεμένοι στο σύστημα, εμφανίζει πότε έγινε εκκίνηση (boot) του υπολογιστή.

```
$ uname -a
Linux node-2 2.4.26-om1 #1 SMP Thu Feb 15 12:59:44 EET 2007 i686
GNU/Linux
$ who -b
system boot  Sep 18 11:43
```

Σε σχέση με τα ορίσματα που μπορεί να δεχτεί μία εντολή η σύνταξη είναι :

```
command arg1
command arg1 arg2
```

όπου *command* είναι το όνομα της εντολής και *arg1*, *arg2* είναι τα πιθανά ορίσματά της.
Ένα παράδειγμα χρήσης εντολής με όρισμα είναι η αλλαγή της ημερομηνίας του συστήματος σε 15 Σεπτεμβρίου :

```
$ date 09152009
date: cannot set date: Operation not permitted
Tue Sep 15 20:09:00 EEST 2009
```

Η εντολή αποτυγχάνει επειδή μόνο ο διαχειριστής έχει δικαίωμα αλλαγής της ημερομηνίας.
Επειδή τις πιο πολλές φορές τα ορίσματα είναι ονόματα αρχείων, πληθώρα παραδειγμάτων θα φανεί στις επόμενες ενότητες.
Επίσης, μπορούν να συνδυαστούν επιλογές και ορίσματα, με τις επιλογές όμως να προηγούνται :

```
command -a -b -cde arg1 arg2
```

Τέλος, μπορούν σε μία γραμμή να συνδυαστούν παραπάνω από μία εντολές με χρήση του διαχωριστικού χαρακτήρα *;* όπως φαίνεται παρακάτω :

```
command1 -a -b ; command2 arg1
```

Σε περίπτωση που έχει δοθεί λάθος εντολή (π.χ. δεν υπάρχει ή έγινε κάποιο ορθογραφικό ή συντακτικό λάθος, ο φλοιός αποκρίνεται με σχετικό μήνυμα λάθους :

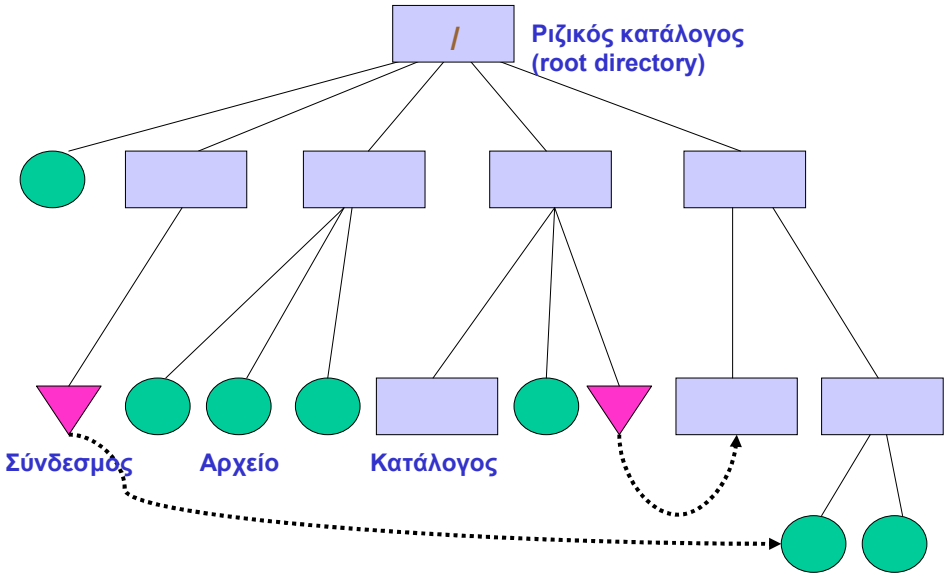
```
$ datte
-bash: datte: command not found
$ uname-a
-bash: uname-a: command not found
$ uname a
uname: extra operand `a'
Try `uname --help' for more information.
```

Πληροφορίες για τη χρήση της κάθε εντολής, τις επιλογές και τα ορίσματα που παίρνει δίνονται με την εντολή **man** (manual pages), για παράδειγμα η εντολή **man date** θα εμφανίσει στην οθόνη, σελίδα-σελίδα, όλα τα σχετικά για την εντολή **date**. Οι σελίδες αλλάζουν με το πλήκτρο του διαστήματος (Space Bar) και η έξοδος γίνεται με το πλήκτρο q.

Λειτουργικό Σύστημα Linux
Ενότητα ΙΙ
ΣΥΣΤΗΜΑ ΑΡΧΕΙΩΝ

1. Η δομή του συστήματος αρχείων.

Στο Λ.Σ. Linux (όπως και σε όλα τα Unix) υπάρχουν δύο βασικές οντότητες πάνω στις οποίες μπορεί να εφαρμοστούν διάφορες λειτουργίες, τα *αρχεία* (**files**) και οι *διεργασίες* (**processes**). Η έννοια του αρχείου στο Linux είναι γενικότερη αυτής που συναντάμε σε άλλα Λ.Σ., δηλαδή της αποθήκευσης οργανωμένης πληροφορίας σε ένα περιφερειακό μέσο όπως ο σκληρός δίσκος. Στο Linux σαν αρχείο μπορεί πέρα από τη συνηθισμένη σημασία να εννοείται και μία περιφερική συσκευή ή θύρα διασύνδεσης και λέγεται *αρχείο συσκευής* (**device file**) ή να είναι ένα ειδικό αρχείο που χρησιμοποιείται για *ενδοεπικοινωνία διεργασιών* (**interprocess communication**). Στην τελευταία περίπτωση υπάρχουν δύο διαφορετικοί τύποι ειδικών αρχείων που λέγονται **sockets** και **named pipes.**
Τα αρχεία στο Linux οργανώνονται σε μία ιεραρχική δομή για μεγαλύτερη ευκολία χειρισμού τους. Η ιεραρχική δομή έχει τη μορφή ανάποδου δέντρου όπως φαίνεται στο παρακάτω σχήμα. Τα φύλλα του δέντρου (κύκλοι στο σχήμα) είναι τα κανονικά αρχεία, ενώ οι εσωτερικοί κόμβοι (παραλληλόγραμμα) είναι ειδικά αρχεία που ονομάζονται κατάλογοι (directories) και η χρήση τους είναι να δημιουργούν αυτή τη δεντρική δομή. Έτσι, ένας κατάλογος μπορεί να περιέχει αρχεία και άλλους καταλόγους (που λέγονται *υποκατάλογοι* (**subdirectories**) του καταλόγου που τους περιέχει), οι οποίοι με τη σειρά τους έχουν άλλους καταλόγους κ.ο.κ.

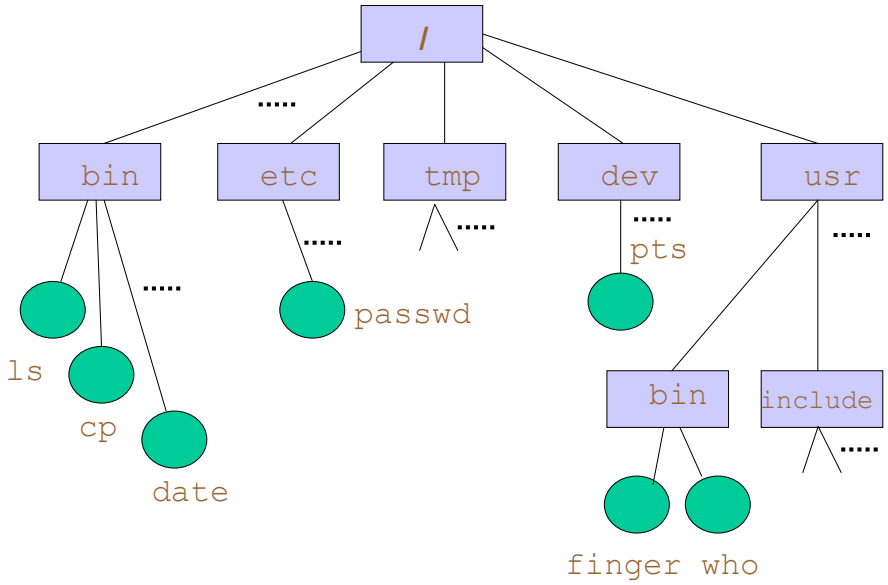


Ο κόμβος του καταλόγου που δεν έχει γονέα λέγεται *ριζικός κατάλογος* (**root directory**) ή *βασικός* ή *κύριος* ή *αρχικός*. Κάθε αρχείο και κατάλογος στο Linux έχει όνομα, το μήκος του οποίου μπορεί να είναι μέχρι 255 χαρακτήρες (στο ext2 filesystem). Οι χαρακτήρες των αρχείων και καταλόγων μπορούν να είναι σχεδόν οποιοιδήποτε (ο / εξαιρείται) και γίνεται διάκριση πεζών-κεφαλαίων. Τα αρχεία που περιέχονται σε έναν κατάλογο πρέπει να έχουν διαφορετικά ονόματα (μόνο αυτά που είναι ακριβώς κάτω από τον κατάλογο – όχι αυτά που περιέχονται σε υποκαταλόγους του). Αρχεία που βρίσκονται σε διαφορετικούς καταλόγους μπορούν να έχουν ίδια ονόματα. Αυτός είναι άλλωστε και ένας από τους κύριους σκοπούς της οργάνωσης σε δεντρικό σχήμα.

Μία άλλη ειδική μορφή αρχείου είναι ο *συμβολικός δεσμός* (**soft link**) που στο σχήμα εμφανίζεται ως τρίγωνο. Ουσιαστικά, κάθε αναφορά σε ένα τέτοιο αρχείο ισοδυναμεί με αναφορά σε ένα αντίστοιχο αρχείο ή κατάλογο προς τον οποίο παραπέμπει ο δεσμός. Οι δεσμοί χρησιμοποιούνται για την παράκαμψη της δενδρικής δομής όταν χρειάζεται από έναν κατάλογο να έχουμε συχνή πρόσβαση σε ένα απομακρυσμένο σημείο του δένδρου.

2. Περιήγηση στους καταλόγους - διαδρομές.

Πολλοί κατάλογοι και αρχεία του Linux δημιουργούνται κατά την εγκατάσταση του Λ.Σ. και έχουν συγκεκριμένη χρήση. Το παρακάτω σχήμα δείχνει ένα τμήμα από τη δομή του συστήματος αρχείων σε μία εγκατάσταση Linux. Στο ριζικό κατάλογο βρίσκονται οι υποκατάλογοι `bin` (περιέχει εκτελέσιμα αρχεία εξωτερικών εντολών (π.χ. αρχείο `date`), `etc` (περιέχει κυρίως αρχεία διαμόρφωσης του συστήματος (π.χ. αρχείο `passwd`), `tmp` που είναι περιοχή για τη δημιουργία προσωρινών αρχείων και καταλόγων από διάφορα προγράμματα, `dev` που περιέχει ειδικά αρχεία συσκευών, κ.τ.λ.



Για την προσπέλαση σε ένα ή περισσότερα αρχεία προκειμένου να γίνει πάνω τους μία λειτουργία (π.χ. ανάγνωση, διαγραφή, ενημέρωση) πρέπει να δοθεί σαν όρισμα στη σχετική εντολή το όνομά τους. Αυτό υπονοεί ότι ο χρήστης (ή η διεργασία η οποία προσπαθεί την προσπέλαση) έχει σαν σημείο αναφοράς το γονικό κατάλογο που περιέχει το αρχείο γιατί είναι πιθανό και σε άλλους καταλόγους να βρίσκονται αρχεία με το ίδιο όνομα. Ο κατάλογος που έχει κάθε στιγμή σα σημείο αναφοράς ένας χρήστης ή μία διεργασία λέγεται *τρέχων κατάλογος* ή *κατάλογος εργασίας* (**current directory, working directory**) και εμφανίζεται με την εντολή **`pwd`** (print working directory). Η αλλαγή τρέχοντος καταλόγου γίνεται με την εντολή **`cd`** *`newdir`* όπου *`newdir`* είναι ο νέος επιθυμητός τρέχων κατάλογος.

Υπάρχουν ειδικοί συμβολισμοί για καταλόγους όπως ο `/` (slash) που αναπαριστά το βασικό κατάλογο, ο συμβολισμός `..` (δύο τελείες) που αναπαριστά το γονικό κατάλογο ενός καταλόγου (δηλαδή έναν κατάλογο προς τα πάνω όπως είναι το σχήμα) και ο `.` (τελεία) που αναπαριστά τον κατάλογο εργασίας. Όπως έχει αναφερθεί ο κάθε χρήστης έχει έναν *προσωπικό κατάλογο* (**home directory**) όπου εκεί μπορεί να διατηρεί τα δικά του αρχεία και υπόκαταλογους. Οι κατάλογοι των χρηστών δεν φαίνονται στο παραπάνω σχήμα γιατί η τοποθεσία τους εξαρτάται από το διαχειριστή του συστήματος. Και για τους προσωπικούς καταλόγους υπάρχει ειδικός συμβολισμός που είναι για το κατάλογο του ίδιου του χρήστη η περισπωμένη `~` , ενώ για τους καταλόγους άλλων χρηστών *`~username`*.

Έχοντας αυτά υπ' όψη μπορούμε να κατανοήσουμε το παρακάτω παράδειγμα περιήγησης σε καταλόγους του Linux με τη χρήση των δύο εντολών **`pwd`** και **`cd`**.

```
$ cd /
$ pwd
/
$ cd bin
$ pwd
/bin
$ cd ..
$ pwd
/
$ cd usr
$ cd bin
$ pwd
/usr/bin
$ cd ~
$ pwd
/home/adp
```

Πρώτα γίνεται αλλαγή στο βασικό κατάλογο, κατόπιν τρέχων κατάλογος γίνεται ο `bin`, ύστερα αλλαγή σε έναν κατάλογο επάνω (γονικός) που στη συγκεκριμένη περίπτωση είναι ο βασικός, κ.ο.κ. Στο τέλος γίνεται αλλαγή τρέχοντος καταλόγου στον προσωπικό του χρήστη που εφαρμόζει τις εντολές.
Η εντολή που εμφανίζει τα αρχεία και τους καταλόγους του τρέχοντος καταλόγου είναι η **`ls`** (list) η οποία έχει και πληθώρα επιλογών. Τα αρχεία εμφανίζονται με λεξικογραφική σειρά κατά στήλες :

```
$ ls
index.txt  oslab
```

Στην απλή εκδοχή της η **`ls`** δεν εμφανίζει τα κρυφά αρχεία που είναι αρχεία τα οποία ξεκινούν με το χαρακτήρα (`.`) της τελείας.
Για να εμφανιστούν όλα τα αρχεία η σχετική επιλογή είναι `-a` ενώ για να εμφανιστούν και λεπτομέρειες σχετικά με κάθε αρχείο πρέπει να δοθεί η επιλογή `-l`.

```
$ ls -al
total 36
drwxr-xr-x  4 apavlidis apavlidis 4096 2007-09-30 21:42 .
drwxrwsr-x  4 root      staff     4096 2007-09-19 11:43 ..
-rw-------  1 apavlidis apavlidis 5291 2007-09-30 02:58 .bash_history
-rw-r--r--  1 apavlidis apavlidis  567 2007-09-19 11:43 .bash_profile
-rw-r--r--  1 apavlidis apavlidis 1834 2007-09-19 11:43 .bashrc
-rw-r--r--  1 apavlidis apavlidis   25 2007-09-30 21:42 index.txt
drwxr-xr-x  5 apavlidis apavlidis 4096 2007-09-30 02:42 oslab
drwx------  2 apavlidis apavlidis 4096 2007-09-19 22:09 .ssh
```

Σε αυτή την περίπτωση εμφανίζεται ένα αρχείο ή κατάλογος σε κάθε γραμμή και η κάθε στήλη δείχνει κάποιες από τις ιδιότητες του αρχείου. Η πρώτη στήλη δείχνει τις εξουσιοδοτήσεις που υπάρχουν σε κάθε αρχείο. Ο πρώτος χαρακτήρας της πρώτης στήλης αναπαριστά τον τύπο του αρχείου. Για κανονικά αρχεία είναι `-`, για καταλόγους είναι `d`. Οι υπόλοιποι εννέα χαρακτήρες που είναι `r`,`w`,`x` και `-` θα εξηγηθούν στην ενότητα για τις εξουσιοδοτήσεις. Η δεύτερη στήλη είναι οι «σκληροί» δεσμοί προς το αρχείο (hard links), αυτή η έννοια όμως δε θα αναλυθεί τώρα. Η τρίτη στήλη δείχνει ποιος είναι ο κάτοχος του αρχείου ενώ η τέταρτη σε ποια ομάδα ανήκει το αρχείο. Η πέμπτη στήλη είναι το μέγεθος του αρχείου σε bytes, και οι δύο επόμενες η ημερομηνία και η ώρα δημιουργίας ή ενημέρωσής του. Στην τελευταία στήλη εμφανίζεται το όνομα του αρχείου. Υπάρχουν πολλές επιλογές στον τρόπο εμφάνισης των αρχείων όπως να εμφανίζονται με σειρά χρόνου δημιουργίας, αντίστροφη σειρά, κ.α. που μπορούν να διερευνηθούν δίνοντας **`man ls`**. Επίσης, μπορεί να υποδειχθεί στην **`ls`** μέσω ορισμάτων ποια αρχεία θέλουμε να εμφανιστούν (εφ' όσον βέβαια υπάρχουν, αλλιώς εμφανίζεται σχετικό μήνυμα λάθους) :

```
$ ls -al .bashrc index.txt
-rw-r--r--  1 apavlidis apavlidis 1834 2007-09-19 11:43 .bashrc
-rw-r--r--  1 apavlidis apavlidis   25 2007-09-30 21:42 index.txt
```

Όπως φάνηκε από το προηγούμενο παράδειγμα οι διάφορες εντολές μπορούν να πάρουν σαν όρισμα ονόματα αρχείων αρκεί αυτά να περιέχονται στον εκάστοτε τρέχοντα κατάλογο. Αυτός ο περιορισμός μπορεί να αρθεί εισάγοντας την έννοια των *ονομάτων απόλυτων και σχετικών διαδρομών* (**absolute pathname, relative pathname**).
Η απόλυτη διαδρομή είναι μία ακολουθία που περιλαμβάνει στη σειρά όλους τους καταλόγους ξεκινώντας από τον βασικό μέχρι το αρχείο-στόχο ή τον κατάλογο-στόχο στον οποίο πρέπει να γίνει η αναφορά, έχοντας σα διαχωριστικό των ονομάτων τον χαρακτήρα `/`. Για παράδειγμα το αρχείο `finger` (είναι το αρχείο που καλείται όταν εκτελείται η αντίστοιχη εντολή) έχει απόλυτη διαδρομή `/usr/bin/finger`, όπως φαίνεται από το προηγούμενο σχήμα που δείχνει μερικούς καταλόγους και αρχεία του Linux.
Η σχετική διαδρομή είναι η ακολουθία από τον τρέχοντα κατάλογο μέχρι τον κατάλογο ή το αρχείο που πρέπει να γίνει η αναφορά. Έτσι αν τρέχων κατάλογός μας είναι ο `/usr/include` για να αναφερθούμε στο αρχείο `/usr/bin/finger` μία σχετική διαδρομή είναι η `../bin/finger`. Ενώ κάθε αρχείο έχει μία μοναδική απόλυτη διαδρομή, οι σχετικές διαδρομές μπορεί να είναι περισσότερες ανάλογα με το ποια διαδρομή ακολουθείται από τον τρέχοντα κατάλογο μέχρι το αρχείο-στόχο. Επομένως μία εναλλακτική σχετική διαδρομή για το αρχείο `/usr/bin/finger` θα ήταν η `../../dev/../usr/bin/finger`. Φυσικά συνηθίζεται να χρησιμοποιούμε την μικρότερη σχετική διαδρομή.
Στο παρακάτω παράδειγμα ο χρήστης βρίσκεται στον προσωπικό του κατάλογο και ελέγχει με δύο διαφορετικούς τρόπους τα αρχεία που βρίσκονται στον κατάλογο `/etc`

```
$ pwd
/home/adp
$ ls /etc
adduser.conf        fstab               lvmtab              rc2.d
adjtime             groff               magic               rc3.d
aliases             group               mailcap             rc4.d
…
$ ls ../../etc
adduser.conf        fstab               lvmtab              rc2.d
adjtime             groff               magic               rc3.d
aliases             group               mailcap             rc4.d
…
```

Στις σχετικές διαδρομές μπορούν να χρησιμοποιηθούν και οι χαρακτήρες `.` και `~` των οποίων η σημασία έχει αναφερθεί προηγουμένως.

3. Βασικές εντολές για λειτουργίες σε αρχεία.

Έχοντας κατανοήσει τη δομή του συστήματος των αρχείων θα περιγραφούν τώρα κάποιες εντολές για βασικές λειτουργίες πάνω σε αρχεία, όπως η αντιγραφή, η μετακίνηση, η αλλαγή ονόματος, η διαγραφή και η δημιουργία αρχείου μηδενικού περιεχομένου. Για κάθε λειτουργία που έχει ως αποτέλεσμα τη δημιουργία κάποιου αρχείου σε κάποιο συγκεκριμένο σημείο (π.χ. αντιγραφή) ή τη διαγραφή (π.χ. μετακίνηση ή διαγραφή) θα πρέπει να υπάρχουν από το χρήστη οι σχετικές εξουσιοδοτήσεις που επιτρέπουν τη διαγραφή. Λεπτομέρειες για το θέμα των εξουσιοδοτήσεων θα αναφερθούν στην επόμενη ενότητα.

- **Δημιουργία αρχείου/αρχείων**

Ένα αρχείο μπορεί να δημιουργηθεί μέσα από μία εφαρμογή κάνοντας αποθήκευση από τη σχετική επιλογή του προγράμματος. Για ευκολία πειραματισμών όμως θα περιγραφεί η εντολή **`touch`** που ο βασικός της σκοπός είναι να αλλάξει την ημερομηνία ενημέρωσης ενός ήδη υπάρχοντος αρχείου εφ' όσον υπάρχει. Αν όμως δεν υπάρχει τότε, δημιουργεί ένα κενό αρχείο (0 bytes).

Η σύνταξη είναι **touch** *file* ή **touch** *file1 file2 file3 ...*
Δηλαδή μπορεί να πάρει ένα ή περισσότερα ορίσματα. Τα ορίσματα μπορεί να είναι ονόματα αρχείων στον τρέχοντα κατάλογο ή απόλυτες διαδρομές ή σχετικές διαδρομές αρχείων. Η δυνατότητα χρήσης διαδρομών ισχύει για όλες τις εντολές που επεξεργάζονται αρχεία, έτσι παρακάτω όπου αναφέρεται αρχείο εννοείται και διαδρομή αρχείου. Αντίστοιχα ισχύουν και ια τους καταλόγους.

- **Αντιγραφή αρχείου/αρχείων**

Έχει τρεις μορφές σύνταξης :
cp *src_file dest_file*
cp *src_file dest_dir*
cp *src_file1 src_file2 dest_dir*
Η πρώτη μορφή είναι η πιο απλή και αντιγράφει το αρχείο *src_file* σε ένα νέο αρχείο με όνομα *dest_file*.
Η δεύτερη μορφή αντιγράφει το αρχείο *src_file* στον κατάλογο *dest_dir* με όνομα αρχείου ίδιο με αυτό του αρχικού.
Και στις δύο περιπτώσεις αν τα αρχεία προορισμού υπάρχουν τότε επικαλύπτονται από το νέο αρχείο προς αντιγραφή.
Η τρίτη μορφή δέχεται πάνω από δύο ορίσματα και αντιγράφει όλα τα αρχεία *src_file1*, *src_file2* που δίνονται στον κατάλογο *dest_dir* διατηρώντας το αρχικό τους όνομα. Έτσι, σε περίπτωση που δίνονται πάνω από δύο ορίσματα τότε, το τελευταίο θα πρέπει υποχρεωτικά να αναφέρεται σε κατάλογο και όχι σε απλό αρχείο.
Επειδή υπάρχει ο κίνδυνος να χαθεί ένα ήδη υπάρχον αρχείο σε περίπτωση που έχει ίδιο όνομα με κάποιο αρχείο προορισμού, υπάρχει η επιλογή **cp -i** που ζητάει επιβεβαίωση.

Παράδειγμα :
```
$ pwd
/home/adp/lab
$ ls
$ touch f1 f2 f3
$ ls
f1  f2  f3
$ cp f3 f4
$ ls
f1  f2  f3  f4
$ cp f2 f6 f4 /tmp
cp: cannot stat `f6': No such file or directory
$ ls /tmp/f2 /tmp/f4
/tmp/f2  /tmp/f4
```

Το αρχείο f6 δεν υπήρχε γι' αυτό εμφανίστηκε μήνυμα σφάλματος. Παρ' όλα αυτά η εντολή συνέχισε αντιγράφοντας το f4 χωρίς πρόβλημα.

- **Διαγραφή αρχείου/αρχείων**

Η σύνταξη είναι **rm** *file* ή **rm** *file1 file2 file3 ...*
Τα αρχεία διαγράφονται χωρίς να ζητηθεί επιβεβαίωση από το χρήστη, γι' αυτό υπάρχει και η επιλογή **rm -i** που μπορεί να χρησιμοποιηθεί σε περίπτωση που ζητείται μαζική διαγραφή.
Αν κάποιο αρχείο δεν υπάρχει τότε, εμφανίζεται μήνυμα λάθους και η εντολή συνεχίζει με τα υπόλοιπα αρχεία.

Συνέχεια προηγούμενου παραδείγματος:
```
$ cd /tmp
$ rm f2 f4
$ ls /tmp/f2 /tmp/f4
ls: /tmp/f2: No such file or directory
```

```
ls: /tmp/f4: No such file or directory
$ cd ~/lab
$ pwd
/home/adp/lab
$ ls
f1  f2  f3  f4
$ rm f3
$ ls
f1  f2  f4
```

- **Μετακίνηση και μετονομασία αρχείου/αρχείων**

Έχει διάφορες μορφές σύνταξεις ανάλογα με το αν το επιθυμητό αποτέλεσμα είναι η μετακίνση ή η μετονομασία.

```
mv old_file new_file
mv old_file dest_dir
mv old_file1 old_file2 dest_dir
```

Η πρώτη σύνταξη κάνει σίγουρα μετονομασία του *old_file* στο *newfile* εφ' όσον η διαδρομή *newfile* αφορά αρχείο. Αν η διαδρομή προορισμού περιέχει κατάλογο διαφορετικό από τη διαδρομή πηγής τότε, γίνεται και μετακίνηση.
Στη δεύτερη μορφή σύνταξης η διαδρομή προορισμού αφορά κατάλογο και γίνεται μετακίνηση στον κατάλογο διατηρώντας το αρχικό όνομα.
Η τρίτη μορφή επιτρέπει πάνω από δύο ορίσματα με το τελευταίο να αφορά υποχρεωτικά κατάλογο. Μετακινούνται δηλαδή όλα τα αρχεία με όνομα *old_file1*, *old_file2* στον κατάλογο *dest_dir* διατηρώντας το αρχικό τους όνομα.
Κατ' αναλογία με την **cp** υπάρχει και η επιλογή **mv -i** για επιβεβαίωση σε περίπτωση εγγραφής σε ήδη υπάρχον αρχείο.

Συνέχεια παραδείγματος:

```
$ pwd
/home/adp/lab
$ ls
f1  f2  f3  f4
$ rm f3
$ ls
f1  f2  f4
$ mv f1 /tmp
$ mv f2 /tmp/fnew
$ ls
f4
$ ls /tmp/f1 /tmp/fnew
/tmp/f1  /tmp/fnew
```

- **Δημιουργία συμβολικών δεσμών.**

Το αρχείο προς το οποίο μπορεί να γίνει η σύνδεση μπορεί να είναι απλό αρχείο ή κατάλογος και είναι το πρώτο όρισμα της εντολής. Το δεύτερο είναι το όνομα του συμβολικού δεσμού. Η σύνταξη είναι επομένως :

```
$ln -s real_file link_file
```

Παράδειγμα:

```
$ ln -s /tmp/fnew f5
$ ln -s /tmp dir6
$ ls -l
total 0
lrwxrwxrwx 1 adp adp 4 2008-05-25 22:12 dir6 -> /tmp
-rw-r--r-- 1 adp adp 0 2008-05-23 18:27 f4
lrwxrwxrwx 1 adp adp 9 2008-05-25 22:12 f5 -> /tmp/fnew
```

4. Βασικές εντολές για λειτουργίες σε καταλόγους.

Για το χειρισμό των καταλόγων υπάρχουν ειδικές εντολές που αναφέρονται παρακάτω :

- **Δημιουργία νέου καταλόγου/καταλόγων.**

Η σύνταξη είναι **mkdir** *dir* ή mkdir *dir1 dir2 dir3 ...* ανάλογα με το πόσοι κατάλογοι πρόκειται να δημιουργηθούν. Υπενθυμίζεται ότι *dir, dir1, dir2, dir3* μπορεί να είναι διαδρομή (απόλυτη ή σχετική) εκτός από το να είναι ένας κατάλογος κάτω από τον τρεχοντα. Επίσης, πρέπει να υπάρχουν οι σχετικές εξουσιοδοτήσεις για τη δημιουργία του καταλόγου.

- **Διαγραφή καταλόγου/καταλόγων.**

Απαραίτητη προϋπόθεση για να μπορέσει να γίνει η διαγραφή είναι ο κατάλογος προς διαγραφή να είναι κενός, δηλαδή θα πρέπει να έχει προηγηθεί η διαγραφή των αρχείων και καταλόγων που τυχόν περιέχει. Η σύνταξη της σχετικής εντολής είναι : **rmdir** *dir* ή **rmdir** *dir1 dir2 dir3 ...* ανάλογα με το πόσοι κατάλογοι πρόκειται να διαγραφούν.

- **Μετακίνηση καταλόγου/καταλόγων.**

Με αυτή την εντολή μετακινείται ο κατάλογος μαζί με τα περιεχόμενά του (αρχεία και υποκαταλόγους). Η εντολή είναι ίδια με αυτήν της μετακίνησης αρχείων και η σύνταξή της είναι :

mv *src_dir dest_dir*
mv *src_dir1 src_dir2 dest_dir*

ανάλογα με το πόσους καταλόγους θέλουμε να μετακινήσουμε. Ο κατάλογος προορισμού (εκεί που πρόκειται να μεταφερθεί ο κατάλογος) είναι πάντα το τελευταίο όρισμα.

5. Μεταχαρακτήρες.

Υπάρχει πολλές φορές η ανάγκη για επεξεργασία πολλών αρχείων που έχουν κάποια κοινή ιδιότητα στο όνομά τους με τις εντολές που προαναφέρθηκαν. Για παράδειγμα η αντιγραφή όλων των αρχείων που το όνομά τους αρχίζει με τους χαρακτήρες `proj`, πέμπτος χαρακτήρας είναι οποιοσδήποτε από τους αριθμούς `1-5` και το όνομα του αρχείου τελιεώνει σε `.doc`. Ο φλοιός του Linux υποστηρίζει τέτοιες λειτουργίες διαθέτοντας ειδικούς χαρακτήρες που λέγονται *μεταχαρακτήρες* (**wildcards**) και οι οποίοι όταν μπαίνουν σε ένα όρισμα που αναπαριστά διαδρομή ή όνομα αρχείου ερμηνεύονται με ειδικό τρόπο.

- **Μεταχαρακτήρας ?**

Ταιριάζει με οποιονδήποτε (αλλά μόνο έναν) χαρακτήρα οπουδήποτε εμφανίζεται σε ένα όνομα αρχείου, καταλόγου ή διαδρομής. Δηλαδή το όρισμα `d?ase.t?t` ταιριάζει μεταξύ άλλων με `dbase.txt drase.ttt`. Έτσι, η εντολή **`cp d?ase.t?t dir2`** θα αντιγράψει όσα αρχεία του τρέχοντος καταλόγου ταιριάζουν με το πρώτο όρισμα στον κατάλογο `dir2`.
Οι μεταχαρακτήρες μπορούν να είναι και τμήμα μίας διαδρομής όπως στην εντολή **`rm dir?/?.txt`** η οποία θα διαγράψει όλα τα αρχεία που ξεκινούν από οποιονδήποτε χαρακτήρα, αμέσως μετά καταλήγουν σε `.txt` και βρίσκονται σε καταλόγους με όνομα τεσσάρων χαρακτήρων με τους τρεις πρώτους να είναι οι χαρακτήρες `dir`.

- **Μεταχαρακτήρας ***

Ταιριάζει με οποιαδήποτε ακολουθία χαρακτήρων (οποιουδήποτε μήκους, ακόμα και μηδενικού). Έτσι, το όρισμα `exam*result` ταιριάζει με τα εξής : `exam12result exam2r6t2result examresult`. Επίσης το `*` ταιριάζει με όλα τα αρχεία (εκτός από αυτά που ξεκινούν με `.` δηλαδή τα κρυφά).

- **Μεταχαρακτήρες [,], -,!**

`[]` : Ταιριάζει με οποιονδήποτε (αλλά μόνο έναν) χαρακτήρα από αυτούς που είναι ανάμεσα στις αγκύλες. Επομένως η ακολουθία `exam[12]result` ταιριάζει μόνο με `exam1result` και `exam2result`

Μέσα στις αγκύλες μπορεί να μπει και περιοχή χαρακτήρων (σύμφωνα με την ASCII διάταξη) εκτός από την παράθεσή τους έναν προς έναν. Για παράδειγμα `[B-F]` σημαίνει οποιοσδήποτε χαρακτήρας από τους `B`, `C`, `D`, `E`, `F`.
Επί πλέον μπορεί να οριστεί και άρνηση χαρακτήρων όπως στο παράδειγμα `[!B-F]` που σημαίνει οποιοσδήποτε χαρακτήρας εκτός από τους `B`, `C`, `D`, `E`, `F`.

- **Χαρακτήρας διαφυγής **

Η λειτουργία του είναι να ερμηνεύει κυριολεκτικά τον χαρακτήρα που τον ακολουθεί. Χρησιμεύει όταν θέλουμε να χρησιμοποιήσουμε κάποιον από τους ειδικούς χαρακτήρες μέσα σε ένα όνομα αρχείου. Στο παρακάτω παράδειγμα, στην πρώτη χρήση της **`rm`** ο χαρακτήρας `*` ερμηνεύεται κυριολεκτικά με αποτέλεσμα τη διαγραφή του αρχείου με όνομα `test*`. Στη δεύτερη χρήση της εντολής ο χαρακτήρας `*` ερμηνεύεται σαν μεταχαρακτήρας με αποτέλεσμα τη διαγραφή των αρχείων `test1`, `test2`, `test3`.

```
$ ls
test*  test1  test2  test3
$ rm test\*
$ ls
test1  test2  test3
$ rm test*
$ ls
$
```

- **Απλά εισαγωγικά ' '**

Έχουν παρόμοια λειτουργία με τον χαρακτήρα διαφυγής. Οτιδήποτε βρίσκεται ανάμεσα στα απλά εισαγωγικά ερμηνεύεται κυριολεκτικά – τυχόν ειδικοί χαρακτήρες χάνουν τη σημασία τους. Η εντολή **`cp 'new**Job!' oldJob?`** θα αντιγράψει το αρχείο με όνομα `new**Job!` Σε ένα νέο αρχείο με όνομα `oldJob?` .

6. Τα περιεχόμενα των αρχείων.

Τα αρχεία στο Linux μπορούν να έχουν οποιοδήποτε περιεχόμενο όπως αρχεία κειμένου, εκτελέσιμα δυαδικά αρχεία γλώσσας μηχανής, εικόνες, αρχεία ήχου και βίντεο κ.α. Ο τύπος ενός αρχείου δε δηλώνεται απαραίτητα από την *κατάληξή* (**extension**) τους όπως σε λειτουργικά συστήματα (π.χ. DOS, Windows). Ούτε και είναι υποχρεωτικό ένα αρχείο να έχει κατάληξη. Για αυτό το λόγο υπάρχει ειδική εντολή που κάνει εκτίμηση για το περιεχόμενο ενός αρχείου με σύνταξη :
`file` *`filename1 filename2 ...`*
Μερικά από τα πιθανά αποτελέσματα και τη σημασία του φαίνονται στον παρακάτω πίνακα :

Απόκριση εντολής **`file`**	Σημασία
`ASCII text`	Κείμενο
`ASCII English text`	Αγγλικό κείμενο
`ASCII C program text`	Πηγαίος κώδικας C
`XXX` `shell script text executable`	Πρόγραμμα φλοιού
`Directory`	Κατάλογος
`ELF 32-bit LSB shared object`	Βιβλιοθήκες
`ELF 32-bit LSB executable`	Εκτελέσιμο προγραμμα
`Data`	Αταξινόμητο

Στην περίπτωση που ένα αρχείο είναι κείμενο (είτε απλό, είτε πηγαίος κώδικας, κτ.λ.) μπορεί να εμφανιστεί στην οθόνη με μία πληθώρα εντολών που περιγράφονται πιο κάτω :
`cat` *`filename`* : Εμφανίζει ολόκληρο το αρχείο.
`more` *`filename`* : Εμφανίζει ολόκληρο το αρχείο, αλλά μόλις γεμίσει η οθόνη σταματάει μέχρι να πατηθεί το πλήκτρο του διαστήματος (Space Bar).
`head` *`filename`* : Εμφανίζει τις πρώτες 10 γραμμές του αρχείου. Υπάρχει και επιλογή ως :
`head -`*`n filename`* : Εμφανίζει τις πρώτες n γραμμές του αρχείου.
`tail` *`filename`* : Εμφανίζει τις τελευταίες 10 γραμμές του αρχείου. Υπάρχει και επιλογή ως :

`tail -`*`n filename`* : Εμφανίζει τις τελευταίες n γραμμές του αρχείου.

Για καλύτερη επισκόπηση ενός αρχείου κειμένου (μετακίνηση μπρος – πίσω), πιθανές αλλαγές και διορθώσεις, θα πρέπει να χρησιμοποιηθεί ένας από τους διορθωτές κειμένου (text editor). Σε περιβάλλον γραμμής εντολών ο διορθωτής που υπάρχει σίγουρα σε όλα τα συστήματα είναι ο vi (καλείται μέσω της εντολής **vi**), όμως είναι σχετικά δύσχρηστος. Πιο απλοί σε χρήση διορθωτές είναι οι **pico** και **nano**. Σε γραφικό περιβάλλον υπάρχουν άλλοι διορθωτές που μπορούν να κληθούν σε ξεχωριστό παράθυρο απ' ευθείας.

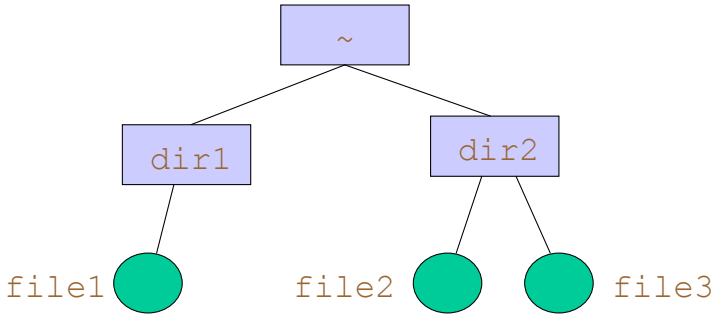
Εργαστηριακές Ασκήσεις

Άσκηση 1.

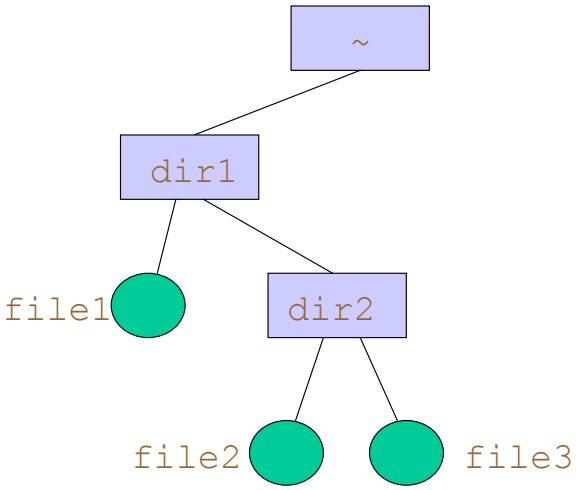
1.Δημιουργείστε ένα αρχείο με όνομα `xyz` στον προσωπικό σας κατάλογο.
2.Ελέγξτε ότι δημιουργήθηκε με την εντολή **`ls`**.
3.Αντιγράψτε το στον κατάλογο `/tmp`
4.Αλλάξτε το όνομα του αρχείου που αντιγράψατε στον `/tmp` σε `.xyz`
5.Ελέξτε ότι πράγματι άλλαξε το όνομα (κρυφό αρχείο).
6.Προσπαθήστε να το αντιγράψετε το στον κατάλογο `/etc`
7.Μετακινηθείτε από τον προσωπικό σας κατάλογο στον `/etc`
8.Ελέγξτε ότι πράγματι μεταφερθήκατε εκεί.
9.Βεβαιωθείτε ότι εκεί υπάρχει το αρχείο `passwd` και αντιγράψτε το στον αρχικό σας κατάλογο.
10.Βεβαιωθείτε ότι υπάρχει το αρχείο `inittab` στον κατάλογο `/etc` και προσπαθήστε να το μετακινήσετε στον αρχικό σας κατάλογο.
11.Διαγράψτε το αρχείο που βάλατε στον κατάλογο `/tmp`
12.Δημιουργήστε στον προσωπικό σας κατάλογο ένα δεσμό προς το αρχείο `/etc/services`
13. Επιστρέψτε στον προσωπικό σας κατάλογο.

Άσκηση 2.

Δημιουργείστε την παρακάτω δομή κάτω από τον προσωπικό σας κατάλογο :



Κατόπιν με μία εντολή δημιουργήστε το παρακάτω δένδρο :

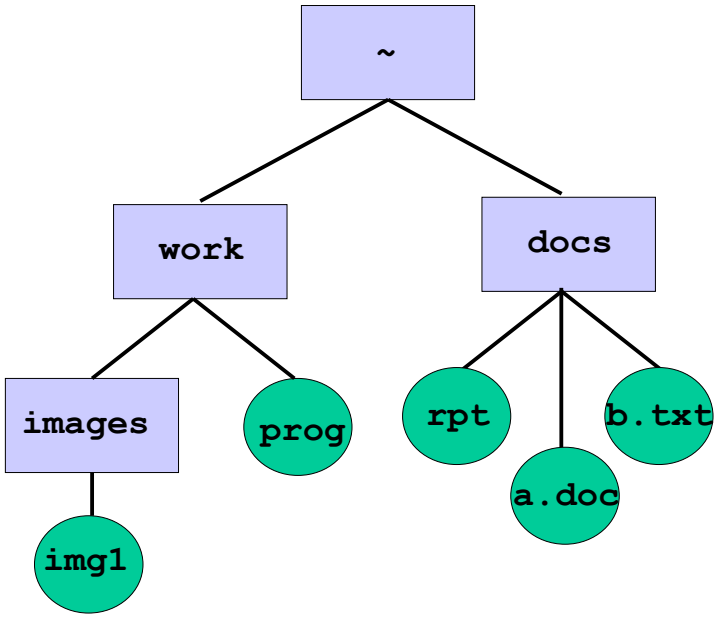


Άσκηση 3.

1.Δημιουργείστε έναν κατάλογο με όνομα `wild` μέσα στον προσωπικό σας κατάλογο.
2.Αντιγράψτε στον κατάλογο `wild` όλα τα αρχεία που υπάρχουν στον κατάλογο `/etc` και το όνομά τους αρχίζει από `p`
3.Αντιγράψτε στον κατάλογο `wild` όλα τα αρχεία που υπάρχουν στον κατάλογο `/etc` και περιέχουν στο όνομά τους την ακολουθία `net`
4.Αντιγράψτε στον κατάλογο `wild` όλα τα αρχεία που υπάρχουν στον κατάλογο `/etc` και το όνομά τους τελειώνει σε `conf`
5.Αντιγράψτε στον κατάλογο `wild` όλα τα αρχεία που υπάρχουν στον κατάλογο `/usr/lib` , το όνομά τους περιέχει τους αριθμούς `1`,`2`,`3` και τελειώνει σε `.so`
6.Με μία εντολή διαγράψτε όλα τα αρχεία που βάλατε στον κατάλογο `wild`

Επί πλέον ασκήσεις.

4. Ενώ βρίσκεστε στον προσωπικό σας κατάλογο (home directory) και χωρίς να μετακινηθείτε καθόλου από εκεί (δηλαδή σε κάθε βήμα ο τρέχων κατάλογός σας να είναι ο προσωπικός σας) δημιουργήστε την παρακάτω δομή αρχείων και καταλόγων κάτω από τον αρχικό σας. (Για τη δημιουργία των αρχείων χρησιμοποιήστε την εντολή **`touch`** και για τον έλεγχο ότι όλα δημιουργήθηκαν όπως πρέπει χρησιμοποιήστε την εντολή **`ls`**). Κατόπιν μετακινηθείτε στον κατάλογο `docs` και διαγράψτε ότι βρίσκεται κάτω από τον κατάλογο `work` καθώς και τον ίδιο τον κατάλογο `work`. Κατά τη διάρκεια των διαγραφών θα πρέπει να μην αλλάζετε τρέχοντα κατάλογο, αλλά να είστε μόνιμα στον `docs`. Ύστερα μετακινηθείτε στο ριζικό κατάλογο (root directory) και διαγράψτε τον κατάλογο `docs` .

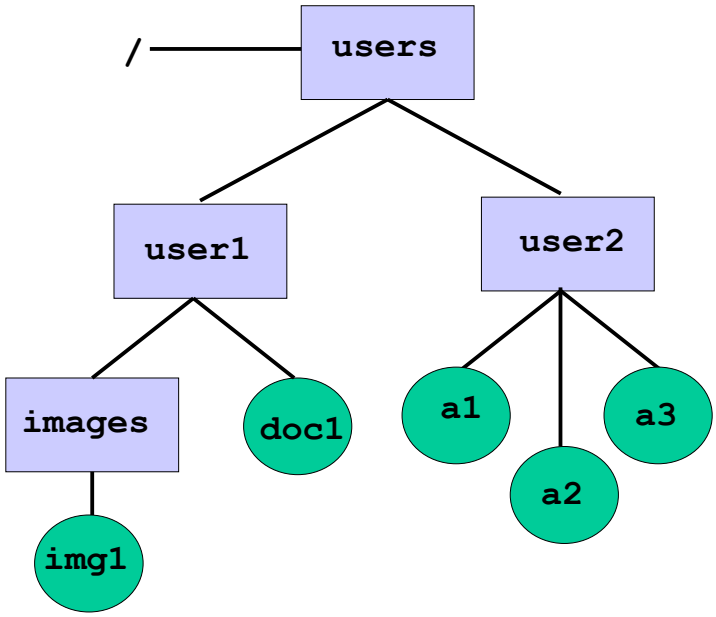


5. Υποθέστε ότι βρίσκεστε στον προσωπικό σας κατάλογο (home directory) ο οποίος περιέχει ένα μόνο αρχείο (εκτός από τα πιθανά κρυφά) με όνομα `class.txt` και θέλετε να το αντιγράψετε στον κατάλογο `/tmp`
α) Με ποια εντολή μπορεί να γίνει αυτό ;
β) Ποιο θα είναι το αποτέλεσμα των εντολών :

```
cp class.txt / tmp
cp class.txt tmp
cp class.txt tmp/
```

```
cp class.txt tmp /
cp class.txt /tmp/class.txt
cp class.txt /tmp/class.dat
cp ./class.txt ../class.txt
```

6. Υποθέστε ότι σε ένα σύστημα αρχείων υπάρχει η δομή που φαίνεται στο παρακάτω σχήμα (ο κατάλογος `users` βρίσκεται κάτω από τον ριζικό) και ότι μεταξύ άλλων υπάρχουν δύο χρήστες με ονόματα `user1` και `user2` με τα home directories τους να είναι τα `user1` και `user2` αντίστοιχα. Αν έχετε κάνει είσοδο (login) στο σύστημα ως χρήστης `user1` και βρίσκεστε (τρέχων κατάλογος – working directory) στον κατάλογο `images` ποιες είναι έγκυρες διαδρομές για να αναφερθείτε στα αρχεία α) `doc1` και β) `a1`. Δώστε όσο το δυνατόν πιο πολλές διαδρομές για κάθε περίπτωση.



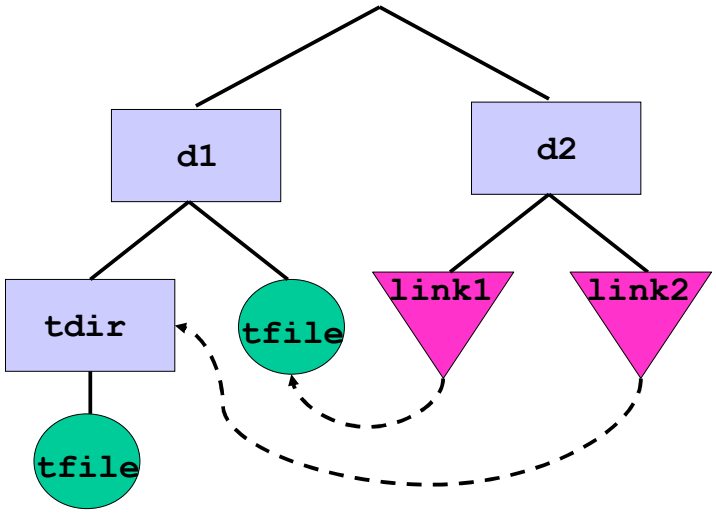
7. Με ποια εντολή μπορεί να αντιγραφεί το αρχείο `passwd` που βρίσκεται στον κατάλογο `/etc` στον κατάλογο `/tmp` ανεξάρτητα από το σε ποιον κατάλογο βρισκόμαστε ; Με ποια μπορεί να μετακινηθεί στον `/tmp` και επί πλέον να μετονομαστεί σε `passwords` (υποθέστε ότι έχετε τα σχετικά δικαιώματα);

8. Ποιο θα είναι το αποτέλεσμα της εντολής **`cp f1 f2 f3`** , υποθέτωντας ότι στον κατάλογο εργασίας (working directory) υπάρχουν μόνο τα αρχεία `f1` και `f2` και τίποτε άλλο ;

9. Με ποια παράμετρο η εντολή **`cp`** διατηρεί την ώρα-ημερομηνία αλλαγής των αρχικών αρχείων στα νέα αρχεία που θα δημιουργήσει;

10. α) Με ποια παράμετρο η εντολή **`ls -l`** εμφανίζει το χρόνο (ημερομηνία και ώρα) προσπέλασης ενός αρχείου αντί για τον χρόνο δημιουργίας/αλλαγής που εμφανίζεται κανονικά;
    β) Με ποιες παραμέτρους παρουσιάζει τα περιεχόμενα ενός καταλόγου ταξινομημένα κατά : μέγεθος, χρόνο μεταβολής, χρόνο προσπέλασης, αλφαβητική επέκταση αρχείου (`.xyz`).
    γ) Με ποια επί πλέον παράμετρο αναστρέφεται η σειρά (αύξουσα αντί φθίνουσα) ;
    δ) Με ποιες ενέργειες πάνω σε έναν κατάλογο αλλάζει ο χρόνος προσπέλασής του και με ποιες ο χρόνος αλλαγής του ; Με ποιες ενέργειες δεν αλλάζει κανένα από αυτά τα χαρακτηριστικά;

11. Δημιουργήστε την παρακάτω δομή αρχείων και καταλόγων.



Διαπιστώστε, ενεργώντας πάνω στους δεσμούς `link1` και `link2`, ποιες από τις εντολές **`ls cp mv ln rm rmdir mkdir`** επιδρούν πάνω στα αρχεία-δεσμούς και ποιες πάνω στα πραγματικά αρχεία ή καταλόγους στα οποία δείχνουν οι δεσμοί.

12. Ποιο θα είναι το αποτέλεσμα των παρακάτω εντολών ;
α) **`cp [A-Z]?? dest`**
β) **`cp *.[!0-9][!0-9][!0-9] dest`**
γ) **`ls ~/prog[x-z]/feb??.c`**
δ) **`mv ??? dest`**
ε) **`cp [a-zA-Z][!a-zA-Z]* dest`**

13. Υποθέστε ότι βρίσκεστε σε έναν κατάλογο ο οποίος περιέχει τα αρχεία `a1`, `a2`, `b1`, `b2` και τους καταλόγους `x`, `y` και `z`. Ποιο θα είναι το αποτέλεσμα των παρακάτω εντολών; (κάθε μία εφαρμόζεται ανεξάρτητα στην παραπάνω δομή).
α) **`cp *`**
β) **`mv *`**
γ) **`cp */`**
δ) **`mv */`**
ε) **`cp a? [x-y]`**
στ) **`mv *1 b*`**
ζ) **`cp *1 b*`**

14. α) Σε ποιές περιπτώσεις χρήσης ειδικών χαρακτήρων σε ονόματα αρχείων πρέπει να χρησιμοποιηθεί ο χαρακτήρας διαφυγής (**`\`**); (Π.χ. μία περίπτωση είναι οι χαρακτήρας **`*`** στο όνομα αρχείου `blahblah*`).
β) Σε έναν κατάλογο δημιουργήστε αρχεία με τα εξής ονόματα :
`?????` `*??????` `*????` `**????**` `**?**` `*****`
Συντάξτε την εντολή **`ls`** με τέτοιο τρόπο ώστε να εμφανίσει μόνο τα αρχεία :
i) `*??????` και `*????`
ii) `*????` και `**????**`

Λειτουργικό Σύστημα Linux
Ενότητα ΙΙΙ
ΕΞΟΥΣΙΟΔΟΤΗΣΕΙΣ

1. Εισαγωγή.

Ένα από τα ισχυρά σημεία του Linux είναι το σύστημα ασφάλειας που διαθέτει για τα αρχεία έναντι μη εξουσιοδοτημένων προσπελάσεων. Για την υλοποίησή του αντιστοιχίζεται σε κάθε αρχείο που υπάρχει στο σύστημα (κανονικό ή μη) ένα σύνολο από ιδιότητες που λέγονται *εξουσιοδοτήσεις* ή *άδειες πρόσβασης* (**access permissions**). Επίσης κάθε αρχείο ανήκει σε έναν χρήστη-ιδιοκτήτη (**owner**) και μία ομάδα-ιδιοκτήτη (**group**). Ο συνδυασμός αυτών των στοιχείων καθορίζει για το ποιος έχει τι είδους πρόσβαση σε κάθε αρχείο.

2. Τύποι πρόσβασης.

Οι άδειες πρόσβασης χωρίζονται σε τρεις κύριους *τύπους* (**access types**) που συμβολίζονται με τα γράμματα `r`,`w`,`x`. Υπάρχουν και άλλοι τύποι (`s`,`t`,`X`) αλλά δε θα αναλυθούν. Οι τρεις τύποι πρόσβασης που μπορεί να έχει (ή να μην έχει) κάποιος σε κάποιο αρχείο ή κατάλογο έχουν την εξής ερμηνεία ανάλογα με το άν αφορούν σε αρχείο ή κατάλογο:

Τύπος / Είδος Αρχείου	`r` (**r**ead)	`w` (**w**rite)	`x` (e**x**ecute)
Κανονικό Αρχείο	Ανάγνωση	Αλλαγή, Προσθήκη, Διαγραφή	Εκτέλεση (πρόγραμμα)
Κατάλογος	Εμφάνιση περιεχομένων (**`ls`**)	Προσθήκη, διαγραφή, μετονομασία αρχείων	Είσοδος (**`cd`**) Πρόσβαση στα αρχεία

Αν δεν υπάρχει άδεια πρόσβασης για κάποιον τύπο αυτό συμβολίζεται με μία παύλα (`-`). Συνολικά τρεις χαρακτήρες συμβολίζουν την πρόσβαση που έχει κάποιος σε ένα αρχείο, ο πρώτος για το αν έχει πρόσβαση ανάγνωσης (`r` ή `-`), ο δεύτερος αν έχει άδεια για εγγραφή (`w` ή `-`) και ο τρίτος χαρακτήρας δείχνει αν υπάρχει άδεια εκτέλεσης (`x` ή `-`). Για παράδειγμα κάποιος που έχει πρόσβαση `r-x` σε ένα αρχείο σημαίνει ότι μπορεί να το διαβάσει και να το εκτελέσει, αλλά όχι να το μεταβάλει ή να το διαγράψει.

3. Κατηγορίες πρόσβασης.

Κάθε αρχείο ή κατάλογος κατατάσει τις εξουσιοδοτήσεις σε τρεις κατηγορίες ή επίπεδα χρηστών : Τις εξουσιοδοτήσεις που έχει ο ιδιοκτήτης (**u**ser,owner) του αρχείου πάνω στο αρχείο, τις εξουσιοδοτήσεις που έχουν οι χρήστες που ανήκουν στην ίδια ομάδα ιδιοκτησίας αρχείου (**g**roup) και τις εξουσιοδοτήσεις που έχουν όλοι οι υπόλοιποι χρήστες (**o**thers) του συστήματος πάνω στο αρχείο. Συνήθως ιδιοκτήτης του αρχείου είναι αυτός που το δημιούργησε (αν και αργότερα μπορεί να αλλάξει η ιδιοκτησία από τον διαχειριστή), και συνήθως ομάδα ιδιοκτησίας του αρχείου είναι η ομάδα στην οποία ανήκει ο ιδιοκτήτης (επίσης είναι δυνατή η εκ των υστέρων αλλαγή).

Επομένως δημιουργείται μία εννιάδα από εξουσιοδοτήσεις (μέσα από τον συνδυασμό των τριών τύπων και των τριών κατηγοριών) που δείχνει ποια κατηγορία χρήστη έχει ποια δικαιώματα πάνω στο αρχείο λαμβάνοντας υπ' όψη την ιδιοκτησία του αρχείου.
Πάλι αυτή η εννιάδα συμβολίζεται με τρεις τριάδες χαρακτήρων μέσα από τους r,w,x,- ανάλογα με το αν υπάρχει ή δεν υπάρχει το σχετικό δικαίωμα. Η εννιάδα αυτή είναι το πρώτο πεδίο που μας δίνει η εντολή **ls -l** (αν εξαιρεθεί ο πρώτος χαρακτήρας που όπως έχει αναφερθεί στη σχετική ενότητα δείχνει τον τύπο του αρχείου). Η σειρά εμφάνισης των τριάδων είναι : πρώτα η τριάδα για τον ιδιοκτήτη, ύστερα για την ομάδα και τέλος η τριάδα που δείχνει τις εξουσιοδοτήσεις για τους υπόλοιπους χρήστες.
Συμπερασματικά, για να αποφανθούμε για τις εξουσιοδοτήσεις που υπάρχουν σε ένα αρχείο πρέπει να ξέρουμε την εννιάδα των αδειών μαζί με το ποιος είναι ιδιοκτήτης και ποια είναι η ομάδα της ιδιοκτησίας. Εδώ πρέπει να αναφερθεί ότι ο διαχειριστής του συστήματος (root) μπορεί να παρακάμψει όλες τις εξουσιοδοτήσεις.
Ακολουθεί ένα παράδειγμα μέσα από την εκτέλεση μίας εντολής **ls -l**

```
$ ls -l
total 4
-rw-r--r-- 1 adp adp    0 2008-05-27 15:15 f1
-rwxr--r-- 1 adp adp    0 2008-05-27 15:15 f2
-r-xr-xr-x 1 adp adp    0 2008-05-27 15:15 f3
-rw-rw-r-- 1 adp adp    0 2008-05-27 15:15 f4
---------- 1 adp adp    0 2008-05-27 15:15 f5
drwxrw---x 2 adp adp 4096 2008-05-27 15:16 f6
```

Το αρχείο f1 μπορεί να διαβαστεί από όλους αλλά μόνο ο ιδιοκτήτης του (adp) μπορεί να το διαγράψει. Επίσης δεν υπάρχουν δικαιώματα εκτέλεσης από κανέναν. Το αρχείο f2 μπορεί να διαβαστεί από όλο τον κόσμο, αλλά μόνο ο ιδιοκτήτης του μπορεί να το διαγράψει και να το εκτελέσει (εφ' όσον εκτελείται). Το αρχείο f3 δίνει εξουσιοδότηση ανάγνωσης και εκτέλεσης προς όλους, αλλά κανένας δεν μπορεί να το διαγράψει. Το αρχείο f4 μπορεί να διαβαστεί από όλους, να διαγραφεί μόνο από τον ιδιοκτήτη ή την ομάδα του και δεν μπορεί να εκτελεστεί. Το αρχείο f5 δεν παρέχει καμμία εξουσιοδότηση προς κανέναν (εκτός του διαχειριστή). Το αρχείο f6 είναι κατάλογος και παρέχει πλήρη δικαιώματα προς τον ιδιοκτήτη, μόνο ανάγνωση και εγγραφή προς την ομάδα (δηλαδή μπορούν να εμφανιστούν τα αρχεία που περιέχει χωρίς όμως κάποιος από την ομάδα να μπορεί να μπει μέσα στον κατάλογο ή να έχει πρόσβαση στα αρχεία που περιέχει. Ο υπόλοιπος κόσμος μπορεί απλώς να μπει μέσα στον κατάλογο χωρίς όμως να μπορεί να κάνει τίποτα άλλο, ούτε καν να δει τι αρχεία περιέχει.
Από τα παραπάνω γίνεται σαφές ότι για να υπάρχει πρόσβαση σε ένα αρχείο πρέπει να υπάρχει πρόσβαση και σε όλους τους γονικούς του καταλόγους μέχρι τον βασικό. Ή αντίστροφα μπορεί αν αποτραπεί η πρόσβαση σε ολόκληρα σύνολα αρχείων αρκεί να αλλαχθούν με τον κατάλληλο τρόπο οι προσβάσεις στους καταλογους που τα περιέχουν.
Κατά τον έλεγχο των δικαιωμάτων ακολουθείται η εξής προτεραιότητα : Πρώτα ελέγχεται αν αυτός που προσπαθεί την προσπέλαση είναι ο ιδιοκτήτης. Αν είναι τότε, ελέγχονται η σχετικοί τύποι πρόσβασης για αυτήν την κατηγορία και λαμβάνεται η απόφαση αν θα επιτραπεί η προσπέλαση. Αν δεν υπάρχουν τα σχετικά δικαιώματα τότε, δεν ελέγχονται οι υπόλοιπες κατηγορίες (ομάδα και υπόλοιποι). Αν αυτός που προσπαθεί την προσπέλαση δεν είναι ο ιδιοκτήτητης τότε, γίνεται έλεγχος στην επόμενη κατηγορία, δηλαδή την ομάδα κ.τ.λ.

4. Αριθμητικός συμβολισμός εξουσιοδοτήσεων.

Υπάρχει και ένας εναλλακτικός αριθμητικός συμβολισμός των εξουσιοδοτήσεων που παρέχει ένα αρχείο και χρησιμοποιείται κυρίως στην αλλαγή των εξουσιοδοτήσεων με την εντολή **chmod** που θα αναφερεθεί στην επόμενη παράγραφο. Κάθε κατηγορία εξουσιοδότησης χαρακτηρίζεται στην ουσία από τρία bits που παίρνουν την τιμή 0 (δεν υπάρχει εξουσιοδότηση) ή 1 (υπάρχει εξουσιοδότηση) – ένα bit για τον κάθε τύπο r, w και x με το bit του r να είναι το πιο σημαντικό

(MSB). Έτσι σε κάθε κατηγορία εξουσιοδότησης αντιστοιχεί ένα δεκαδικό ψηφίο με τιμή 0-7 που προκύπτει μετατρέποντας τον δυαδικό αριθμό σε δεκαδικό. Η μετατροπή μπορεί να γίνει εύκολα κάνοντας την αντιστοιχία `r`→4, `w`→2, `x`→1, όπου υπάρχει απουσία εξουσιοδότησης → 0 και αθροίζοντας. Συνδυάζοντας στην σειρά (owner.group,others) τα τρία δεκαδικά ψηφία παίρνουμε την αριθμητική αναπαράσταση εξουσιοδοτήσεων που στην ορολογία του Linux λέγεται mode.Αυτή η διαδικασία αναπαρίσταται στον παρακάτω πίνακα.
Για να γίνει πιο κατανοητό, αναφερόμενοι στο προηγούμενο παράδειγμα, βρίσκουμε τα αρχεία `f1` ως `f6` έχουν modes εξουσιοδοτήσεων 644,744,555,664,000,761, αντίστοιχα.

Κατηγορίες / **Τύποι**	**u**ser	**g**roup	**o**thers
read (msb)	`r → 4 , - → 0`	`r → 4 , - → 0`	`r → 4 , - → 0`
write	`w → 2 , - → 0`	`w → 2 , - → 0`	`w → 2 , - → 0`
E**x**ecute (lsb)	`x → 1 , - → 0`	`x → 1 , - → 0`	`x → 1 , - → 0`
mode	1ο ψηφίο (κάθετο άθροισμα)	2ο ψηφίο (κάθετο άθροισμα)	3ο ψηφίο (κάθετο άθροισμα)

5. Αλλαγή των εξουσιοδοτήσεων.

Η αλλαγή των εξουσιοδοτήσεων που έχει κάποιο αρχείο μπορεί να γίνει μόνο από τον ιδιοκτήτη του (ή από τον διαχειριστή) με την εντολή **`chmod`** (change mode). Η εντολή προσφέρει δύο μορφές σύνταξης, μία με συμβολικό τρόπο και μία με τον αριθμητικό τρόπο (modes).

- Συμβολική σύνταξη

`chmod` *`smode [,smode] file1 ...`*

όπου *`smode=`*`[`**`ugoa`**`]{`**`+|-|=`**`}[`**`rwx`**`]` είναι τα συμβολικά δικαιώματα που θα υπάρχουν πάνω στα αρχεία *`file1 ...`*

`[`**`ugoa`**`]` σημαίνει ένας ή περισσότεροι από τους χαρακτήρες u,g,o,a με τον κάθε χαρακτήρα να αντιστοιχεί στην κατηγορία ιδιοκτήτης, ομάδα, υπόλοιποι, όλες οι κατηγορίες, αντίστοιχα.

`{`**`+|-|=`**`}` σημαίνει μόνο έναν από τους χαρακτήρες +,-,=, με τον κάθε ένα να σημαίνει προσθήκη, αφαίρεση και ακριβής προσδιορισμός του τύπου εξουσιοδότησης που ακολουθεί.

`[`**`rwx`**`]` σημαίνει ένας ή περισσότεροι από τους χαρακτήρες r,w,x με τον κάθε ένα να ορίζει τον αντίστοιχο τύπο δικαιώματος (ανάγνωση, εγγραφή, εκτέλεση).

Για παράδειγμα η εντολή **`chmod u+wx,g=r,o-rwx file1`** θα προσθέσει εξουσιοδότηση εγγραφής/διαγραφής και εκτέλεσης στο αρχείο **`file1`** για τον ιδιοκτήτη (η εξουσιοδότηση ανάγνωσης δεν επηρρεάζεται), οι εξουσιοδοτήσεις για την ομάδα τίθενται να είναι μόνο ανάγνωσης ανεξάρτητα από το ποιες ήταν πριν, και αφαιρούνται όλες οι εξουσιοδοτήσεις που τυχόν είχαν όλοι οι υπόλοιποι.

Στην εντολή **`chmod a-w file2 file3`** αφαιρούνται από όλους (όλες οι κατηγορίες) τα δικαιώματα εγγραφής/διαγραφής που τυχόν υπήρχαν στα αρχεία **`file2`** και **`file3`**. Οι υπόλοποι τύποι δικαιωμάτων μένουν ως είχαν.

- Αριθμητική σύνταξη

`chmod` *`mode file1 ...`*

όπου *`mode`* είναι τρία δεκαδικά ψηφία και καθορίζουν τις εξουσιοδοτήσεις που θα έχουν τα αρχεία *`file1 ...`* σύμφωνα με τον αριθμητικό συμβολισμό δικαιωμάτων που έχει αναλυθεί πιο πριν. Αυτή η σύνταξη είναι ισοδύναμε μόνο με την περίπτωση χρήσης = στη συμβολική σύνταξη.

Έτσι, ή εντολή **chmod 755 file4** δίνει για το αρχείο **file4** στον ιδιοκτήτη όλα τα δικαιώματα (7) και στην ομάδα και τον υπόλοιπο κόσμο δικαιώματα ανάγνωσης και εκτέλεσης (55).
Και ή εντολή **chmod 400 file5 file6** δίνει για τα αρχεία **file5** και **file6** στον ιδιοκτήτη δικαίωμα ανάγνωσης (4) και στην ομάδα και όλους τους υπόλοιπους δε δίνει κανένα διακαίωμα (00).

Ένα ακόμα παράδειγμα που συνοψίζει και τους δύο τρόπους σύνταξης είναι το παρακάτω απόσπασμα από εντολές και αποκρίσεις του φλοιού που δείχνουν πως αλλάζουν διαδοχικά τα δικαιώματα ενός αρχείου:

```
$ touch file1
$ ls -l file1
-rw-r--r--  1 apavlidis apavlidis 0 2007-09-27 02:13 file1
$ chmod u-w file1
$ ls -l file1
-r--r--r--  1 apavlidis apavlidis 0 2007-09-27 02:13 file1
$ chmod u+x,g=rw,o-r file1
$ ls -l file1
-r-xrw----  1 apavlidis apavlidis 0 2007-09-27 02:13 file1
$ chmod 754 file1
$ ls -l file1
-rwxr-xr--  1 apavlidis apavlidis 0 2007-09-27 02:13 file1
$ chmod 000 file1
$ ls -l file1
----------  1 apavlidis apavlidis 0 2007-09-27 02:13 file1
```

6. Προκαθορισμένες εξουσιοδοτήσεις (default permissions).

Κάθε φορά που δημιουργείται ένα αρχείο από μία εφαρμογή (text editor, αντιγραφή από άλλο αρχείο), δίνει και εξουσιοδοτήσεις προς τις διάφορες κατηγορίες χρηστών. Π.χ. στο προηγούμενο παράδειγμα στο αρχείο `file1` που δημιουργήθηκε με την εντολή **touch** τέθηκαν οι εξουσιοδοτήσεις `rw-r--r–` ή σε αριθμητική μορφή «644».
Το είδος της προκαθορισμένης εξουσιοδότησης (μάσκα δικαιωμάτων) που θα έχουν τα αρχεία που δημιουργούνται καθορίζεται κατά την είσοδο του χρήστη στο σύστημα με βάση κάποια αρχεία αρχικοποίησης περιβάλλοντος. Αυτό όμως μπορεί να αλλάξει χρησιμοποιώντας την εντολή **umask** για όλη τη διάρκεια που ο χρήστης θα είναι συνδεδεμένος στο φλοιό του και μέχρι να αποσυνδεθεί. Αυτή η αλλαγή θα ισχύει για οποιοδήποτε σημείο του συστήματος αρχείων (δηλαδή οπουδήποτε δημιουργηθεί αρχείο) αλλά μόνο για το συγκεκριμένο φλοιό στον οποίο δόθηκε η εντολή. Αν ο χρήστης ανοίξει και άλλο φλοιό (τερματικό) θα ισχύουν οι παλιές προκαθορισμένες εξουσιοδοτήσεις.
Η σύνταξη της εντολής είναι : **umask** *ugo* όπου *ugo* είναι ένας τριψήφιος δεκαδικός αριθμός κατ΄αναλογία με την παράμετρο *mode* στην αριθμητική σύνταξη της εντολής **chmod**. Δηλαδή ο αριθμός *u* αφορά τα δικαιώματα του ιδιοκτήτη, ο *g* της ομάδας και ο *o* των υπολοίπων χρηστών. Οι αριθμοί έχουν τιμές 0..7 αλλά η σημασία τους είναι η συμπληρωματική της σημασίας που έχουν στην εντολή **chmod**. Δηλαδή δείχνουν ποια δικαιώματα δε θα υπάρχουν και όχι τι δικαιώματα θα υπάρχουν. Επομένως αν για παράδειγμα θέλουμε σε όλα τα αρχεία που δημιουργούνται από εδώ και στο εξής να υπάρχει εξουσιοδότηση της μορφής `rw-r-----` θα πρέπει να δοθεί η εντολή **umask 026**. (Ο αριθμός 2 αφαιρεί από την ομάδα το δικαίωμα εγγραφής (w) και ο 6 από τους υπόλοιπους το δικαίωμα ανάγνωσης και εγγραφής (rw). Να σημειωθεί ότι ανεξάρτητα από το αν η **umask** δώσει δικαιώματα σε κάποια κατηγορία για εκτέλεση (x), αυτά δε θα ισχύουν για κανονικά αρχεία παρά μόνο για καταλόγους. Τέλος, η τρέχουσα μάσκα εξουσιοδοτήσεων εμφανίζεται δίνοντας απλώς **umask** χωρίς κανένα όρισμα.

Ο παρακάτω πίνακας έχει μερικά ακόμα παραδείγματα παραμέτρων **umask** και τις εξουσιοδοτήσεις που γεννιούνται:

umask	Αρχεία	Κατάλογοι
000	rw-rw-rw-	rwxrwxrwx
001	rw-rw-rw-	rwxrwxrwx
002	rw-rw-r--	rwxrwxr-x
022	rw-r--r--	rwxr-xr-x
666	---------	--x--x--x
777	---------	---------

7. Επιλογή αναδρομής για εντολές αρχείων.

Έχοντας καλύψει και την εντολή αλλαγής εξουσιοδοτήσεων (**chmod**) θα περιγραφεί μία επιλογή που έχει αυτή η εντολή καθώς και οι **ls**, **cp**, **rm**, με την οποία επιδρούν σε ένα ολόκληρο υποδένδρο του συστήματος αρχείων, δηλαδή σε έναν κατάλογο και όλους τους υποκαταλόγος που υπάρχουν από κάτω. Η επιλογή αυτή είναι κοινή και για τις τέσσερις εντολές και είναι **-R**.

Η εντολή **ls -R** *dir* θα εμφανίσει τη λιστα καταλόγων και αρχείων όλου του υποδέντρου κάτω από τον κατάλογο *dir*.

Η εντολή **cp -R** *src_dir dest_dir* θα αντιγράψει, διατηρώντας τη δομή των υποκαταλόγων, όλων του περιεχομένων (υποκατάλογοι και αρχεία) του καταλόγου *src_dir* μέσα στον κατάλογο *dest_dir*.

Η εντολή **rm -R** *dir* θα διαγράψει όλους τους υποκαταλόγους και αρχεία που βρίσκονται μέσα στον κατάλογο *dir*.

Η εντολή **chmod -R** *mode dir* θα αλάξει τις εξουσιοδοτήσεις σε όλα τα περιεχόμενα του καταλόγου.

Εργαστηριακές Ασκήσεις

Άσκηση 1.

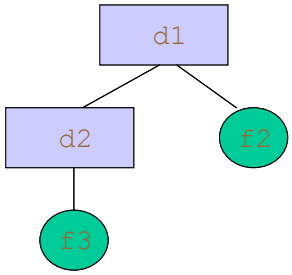
1.Ελέγξτε τα δικαιώματα πρόσβασης που έχετε πάνω στο αρχείο `/etc/timezone` και προσπαθήστε να το διαγράψετε.
2.Αντιγράψτε το προηγούμενο αρχείο στον προσωπικό σας κατάλογο, και ξαναελέγξτε τα δικαιώματά σας. Είναι ίδια με αυτά που έχετε στο προηγούμενο αρχείο;
3.Δώστε πλήρη δικαιώματα στον εαυτό σας και στους άλλους χρησιμοποιώντας τον αριθμητικό τρόπο σύνταξης της **`chmod`** και ελέγξτε το αποτέλεσμα με την εντολή **`ls`** .
4.Αφαιρέστε όλα τα δικαιώματα πρόσβασης από τον εαυτό σας, αλλά όχι από την ομάδα και τους άλλους, χρησιμοποιώντας τη συμβολική σύνταξη της **`chmod`**
5.Δοκιμάστε να διαβάσετε τα περιεχόμενα του αρχείου με **`cat`** ή **`more`**
6.Δοκιμάστε να δημιουργήσετε ένα αντίγραφο του αρχείου με όνομα `timezone2`
7.Δοκιμάστε να αλλάξετε το όνομα του αρχείου σε `timezone3`
8.Δοκιμάστε να διαγράψετε ότι αρχείο δημιουργήθηκε .

Άσκηση 2.

1.Δημιουργήστε στον προσωπικό σας κατάλογο έναν κατάλογο με όνομα `level1` και ελέγξτε τα δικαιώματα πρόσβασης σε αυτόν.
2.Αντιγράψτε μέσα στον κατάλογο `level1` το αρχείο `/etc/crontab`
3.Ενώ είστε στον προσωπικό σας κατάλογο δοκιμάστε τις εντολές **`ls level1`** , **`ls -l level1`** και **`ls level1/crontab`**
4.Αλλάξτε τα δικαιώματα του καταλόγου `level1` σε `r--------` και δοκιμάστε τις προηγούμενες εντολές.
5.Αλλάξτε τα δικαιώματα του καταλόγου `level1` σε `--x------` και δοκιμάστε τις προηγούμενες εντολές καθώς και την εντολή **`more level1/crontab`**.
6.Αντιγράψτε το αρχείο `level1/crontab` στον προσωπικό σας κατάλογο.
7.Αλλάξτε τα δικαιώματα του `level1` σε `-w-------` και δοκιμάστε να αντιγράψετε στον κατάλογο το αρχείο `/etc/services` και να σβήσετε το αρχείο `level1/crontab`
8.Αλλάξτε τα δικαιώματα του `level1` σε `-wx------` και δοκιμάστε να αντιγράψετε στον κατάλογο το αρχείο `/etc/services` και να σβήσετε το αρχείο `level1/crontab`

Άσκηση 3.

1.Βεβαιωθείτε ότι έχετε μάσκα δικαιωμάτων `022`
2.Δημιουργήστε το ακόλουθο υποδέντρο :



3.Αλλάξτε τη μάσκα δικαιωμάτων σε `777`
4.Αντιγράψτε όλο το υποδέντρο `d1` σε έναν κατάλογο με όνομα `cpdir`
4.Ελέγξτε το αποτέλεσμα
5.Προσπαθήστε να κάνετε αναδρομική αλλαγή δικαιωμάτων των αρχείων του υποδέντρου σε `rw-r--r--`
6.Προσπαθήστε να διαγράψετε αναδρομικά το υποδέντρο `cpdir`

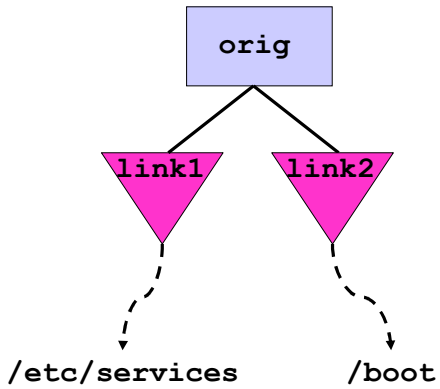
Επί πλέον ασκήσεις.

4. Σε ένα αρχείο υπάρχουν δικαιώματα πρόσβασης `rwxrwxr-x`
   α) Με ποιούς τρόπους μπορεί να αλλάξει σε `rwxr-xr-x`;
   β) Με ποιούς τρόπους μπορεί να αλλάξει σε `r-xr-xr-x`;

5. Θέστε την μάσκα δικαιωμάτων σε τέτοια τιμή ώστε οτιδήποτε αρχείο δημιουργείτε να δίνει πλήρη δικαιώματα στην ομάδα (group) και τους υπόλοιπους (others), ενώ να μη δίνει κανένα δικαίωμα στον ιδιοκτήτη (user). Δημιουργείστε ένα αρχείο κειμένου με κάποιον editor (π.χ. vi ή pico) που θα εκκινήσετε από το ίδιο τερματικό στο οποίο δώσατε την εντολή αλλαγής μάσκας. Προσπαθήστε να διαβάσετε το αρχείο με κάποια εντολή και ύστερα να το σβήσετε, ώστε να διαπιστώσετε αν παίζουν κάποιο ρόλο τα δικαιώματα που έχουν οι υπόλοιποι (others) έναντι του κατόχου (owner) του αρχείου.

6. Διαπιστώστε κατά πόσο παίζουν ρόλο στη διαγραφή ενός καταλόγου με την εντολή **`rmdir`** τα δικαιώματα πρόσβασης που έχει ο ίδιος ο κατάλογος. Αν δεν παίζουν ρόλο με ποιους τρόπους μπορεί να προστατευθεί από διαγραφή ;

7. Δημιουργήστε την παρακάτω δομή (Τα αρχεία `link1` και `link2` είναι δεσμοί προς τις απόλυτες διαδρομές που φαίνονται στο σχήμα).



   Κάντε αναδρομική αντιγραφή του καταλόγου `orig` και των δύο δεσμών του σε ένα νέο κατάλογο `copy`.
   α) Διαπιστώστε αν τα αρχεία `copy/link1` και `copy/link2` είναι δεσμοί οι οποίοι δείχνουν όπου έδειχναν και οι αρχικοί δεσμοί.
   β) Βρείτε με ποια παράμετρο η εντολή **`cp`** θα κάνει την ίδια λειτουργία, αντιγράφοντας τα αρχεία (και τους καταλόγους) στα οποία δείχνουν οι δεσμοί αντί για τους ίδιους τους δεσμούς και επιβεβαιώστε το.
   γ) Παρατηρήστε τις ώρες δημιουργίας των αρχείων κάτω από τον κατάλογο `copy` και βρείτε με ποια παράμετρο θα διατηρηθούν οι ημερομηνίες/ώρες δημιουργίας των αρχείων προς αντιγραφή στα νέα αρχεία.

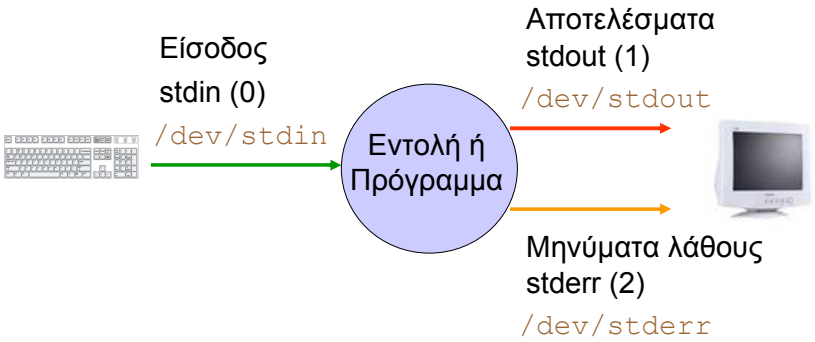
8. Βρείτε με ποια παράμετρο επιλογή η εντολή **`chmod -R`** θα δώσει δικαιώματα εκτέλεσης για καταλόγους αλλά όχι και σε αρχεία.

9. Τι εξουσιοδοτήσεις θα δώσετε σε έναν κατάλογο ώστε διάφοροι χρήστες να μπορούν να τοποθετήσουν σε αυτόν αρχεία χωρίς ο καθένας να γνωρίζει τι τοποθετούν οι υπόλοιποι;

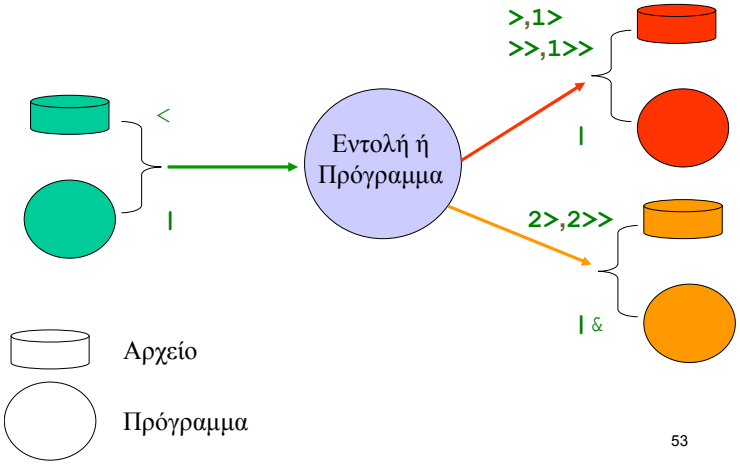
Λειτουργικό Σύστημα Linux
Ενότητα IV
ΦΙΛΤΡΑ – ΑΝΑΚΑΤΕΥΘΥΝΣΕΙΣ - ΣΩΛΗΝΩΣΕΙΣ

1. Εισαγωγή.

Όλες οι εντολές του Linux παρουσιάζουν τα αποτελέσματά τους στην οθόνη. Η οθόνη είναι γνωστή με τον όρο *καθιερωμένη έξοδος* (**standard output – stdout**). Ο κωδικός της καθιερωμένης εξόδου είναι ο αριθμός 1. Στην οθόνη επίσης παρουσιάζονται και τα μηνύματα σφάλματος (π.χ. δεν υπάρχει εξουσιοδότηση να διαγραφεί ένα αρχείο ή κάποιο αρχείο δεν βρέθηκε). Αυτή η έξοδο όμως θεωρείται ξεχωριστή ροή δεδομένων και λέγεται *καθιερωμένη έξοδος σφαλμάτων* (**standard error –stderr**) και μπορεί να δεχθεί διαφορετικό χειρισμό από την καθιερωμένη έξοδο όπως θα φανεί αργότερα. Ο κωδικός της καθιερωμένης εξόδου σφαλμάτων είναι 2.
Για όσες εντολές δέχονται είσοδο από το πληκτρολόγιο (π.χ. τα φίλτρα που θα αναλυθούν παρακάτω) ο όρος για τη ροή δεδομένων εισόδου είναι *καθιερωμένη είσοδος* (**standard input – stdin**) και έχει κωδικό 0.



Βασικό μέρος της φιλοσοφίας του Linux είναι η δυνατότητα συνεργασίας των διαφόρων εντολών και πρόσθετων προγραμμάτων μεταξύ τους . Αυτή η δυνατότητα δεν είναι ενδογενής στα προγράμματα, αλλά υποστηρίζεται από τον φλοιό του Linux.
Αυτό γίνεται με τη δυνατότητα που έχει ο φλοιός να παρουσιάζει την έξοδο ενός προγράμματος που κανονικά θα παρουσιαζόταν στην οθόνη, σαν είσοδο σε ένα άλλο πρόγραμμα, είσοδος η οποία θα δινόταν κανονικά από το πληκτρολόγιο σε αυτό το δεύτερο πρόγραμμα.
Επί πλέον παρέχεται και η δυνατότητα η έξοδος να οδεύει σε ένα αρχείο αντί στην οθόνη και η είσοδος να λαμβάνεται από ένα αρχείο αντί από το πληκτρολόγιο.
Η πρώτη δυνατότητα είναι γνωστή με τον όρο *σωλήνωση* ή *διοχέτευση* (**piping**), ενώ η δεύτερη δυνατότητα λέγεται *ανακατεύθυνση εισόδου-εξόδου* (**input/output redirection**). Η σύνδεση των εντολών μεταξύ τους και με αρχεία γίνεται με τη χρήση ειδικών μεταχαρακτήρων του φλοιού όπως : |, >, <, **&**.
Επιπροσθέτως, ειδικά προγράμματα που λέγονται *φίλτρα* (**filters**) μετασχηματίζουν την είσοδο που παίρνουν από το πληκτρολόγιο σε άλλη μορφή που παρουσιάζεται στην οθόνη. Ο συνδυασμός των τριών παραπάνω δυνατοτήτων προσφέρει εκπληκτικές δυνατότητες στον χρήστη ενός Linux συστήματος, χωρίς να τον αναγκάζει να καταφύγει σε προγραμματικές λύσεις. Πρέπει να τονιστεί ότι αυτές οι δυνατότητες παρέχονται για την περίπτωση δεδομένων μορφής κειμένου (ASCII χαρακτήρες).
Παρακάτω παρουσιάζονται σχηματικά οι έννοιες που προαναφέρθηκαν :



2. Φίλτρα.

Μερικά φίλτρα έχουν ήδη αναφερθεί (`cat`,`head`,`tail`), αλλά δεν είχαν ειδωθεί από αυτή τη σκοπιά. Για παράδειγμα η εντολή `cat` χωρίς ορίσματα είναι το ταυτοτικό φίλτρο, δηλαδή αντιγράφει τα δεδομένα του πληκτρολογίου στην οθόνη, μέχρι να πατηθεί ο συνδυασμός πλήκτρων **`Ctrl-D`**.

```
$ cat
1234
1234
test
test
^D
$
```

Ακολουθούν κάποια μη τετριμμένα φίλτρα που είναι αρκετά χρήσιμα, μαζί με κάποια παραδείγματα. Σε όλα τα φίλτρα ο τερματισμός γίνεται με **`Ctrl-D`**.

- **Το φίλτρο `grep`**

Αυτό το φίλτρο, όπως και όλα τα φίλτρα, παίρνει πολλές επιλογές και παραμέτρους. Στην απλή του μορφή όμως, με μία παράμετρο, βγάζει στην έξοδό του (οθόνη) τις γραμμές που παίρνει από την είσοδο και περιέχουν την παράμετρο που του δίνεται :

```
$ grep patt1
1234
dfg efrf pat
wej  patt1 weff
wej  patt1 weff
wedwedi wedin wpej pppatt1wefnlk
wedwedi wedin wpej pppatt1wefnlk
dw
^D
$
```

Όπως φαίνεται, μόνο οι γραμμές που περιέχουν την ακολουθία `patt1` αντιγράφονται στην έξοδο.

- **Το φίλτρο `sort`**

Δέχεται όλες τις γραμμές από την είσοδο (στην περίπτωση εισόδου από το πληκτρολόγιο μέχρι να δεχτεί τον χαρακτήρα **`Ctrl-D`** και κατόπιν τις παρουσιάζει στην έξοδο, ταξινομημένες κατά την λεξικογραφική τους σειρά (σύμφωνα με τη διάταξη ASCII).

```
$ sort
fgh
abc
xyz
efg
^D
abc
efg
fgh
xyz
$
```

- **Το φίλτρο `cut`**

Επιλέγει από την είσοδο στήλες χαρακτήρων και εμφανίζει μόνο αυτές στην έξοδο. Επίσης, μπορεί να επιλέξει πεδία με το κάθε πεδίο να είναι μία λέξη που χωρίζεται με κενά από τις υπόλοιπες (ή με άλλο χαρακτήρα διαχωρισμού εκτός του κενού που δίνεται σαν επιλογή).

```
$ cut -c3,5
1234567890 ervf wtg wrt
35
abcdefeghijklmnop xyz jkl a
ce
^D
$
```

- **Το φίλτρο `tr` (translate)**

Κάνει αντικατάσταση των χαρακτήρων της εισόδου σε άλλους στην έξοδο με βάση μία αντιστοιχία «1-1» που δίνεται σαν επιλογή κατά την κλήση. Στο επόμενο παράδειγμα οι χαρακτήρες `a,b,c` αντικαθίστανται στην έξοδο από τους `x,y,z`, αντίστοιχα.

```
$ tr "abc" "xyz"
afgrt eker;4340
xfgrt eker;4340
bcoe3dp 34pxyz
yzoe3dp 34pxyz
^D
$
```

- **Το φίλτρο `wc` (word count)**

Μετράει τον αριθμό των γραμμών, των λέξεων και των χαρακτήρων της εισόδου και εμφανίζει το αποτέλεσμα της μέτρησης στην έξοδο.

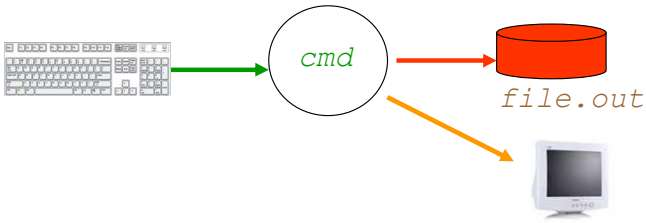
```
$ wc
wffrromj pro fwfr
rferf erff
egtptptg tgt gtpgt gtp
^D
       3       9      52
$
```

Χρησιμοποιώντας την εντολή βοήθειας **`man`** μπορούν να βρεθούν διάφορες επιλογές που υποστηρίζονται από τα παραπάνω φίλτρα.

3. Ανακατευθύνσεις.

Με την ανακατεύθυνση της καθιερωμένης εξόδου ή της εξόδου σφαλμάτων ή και των δύο εξόδων επιτυγχάνουμε να γραφούν τα αποτελέσματα της εντολής σε ένα αρχείο. Για κάθε περίπτωση δίνεται σχηματικά η ροή των δεδομένων, η γενική σύνταξη και κάποιο παράδειγμα. Στους συμβολισμούς της σύνταξης που εμφανίζονται στα σχήματα *cmd*, cmd1, cmd2 είναι κάποια εντολή του Linux, ενώ *file*, *file.in*, *file.out*, *file.err* είναι ονόματα αρχείων. Το πλάγιο (πορτοκαλί) βέλος αφορά την καθιερωμένη έξοδο σφαλμάτων.

α) Ανακατεύθυνση καθιερωμένης εξόδου.

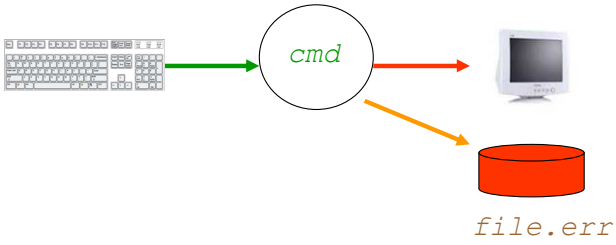


cmd **>** *file.out*
cmd **>>** *file.out*

Με το συμβολισμό >, αν το αρχείο *file.out* δεν υπάρχει τότε, δημιουργείται. Αν υπάρχει τότε, το αποτέλεσμα της εντολής γράφεται πάνω στα παλιά περιεχόμενα. Αν θέλουμε να γίνει προσθήκη στο τέλος των δεδομένων που ήδη υπάρχουν τότε, χρησιμοποιείται ο συμβολισμός >>. Τα πιθανά σφάλματα οδηγούνται στην οθόνη.

```
$ date > file1
$ cat file1
Fri May 30 15:46:27 EEST 2008
$ uname -a > file1
$ cat file1
Linux erasmus 2.6.15-26-server #1 SMP Fri Sep 8 21:00:37 UTC
2006 i686 GNU/Linux
$ who >> file1
$ cat file1
Linux erasmus 2.6.15-26-server #1 SMP Fri Sep 8 21:00:37 UTC
2006 i686 GNU/Linux
adp      pts/0        2008-05-30 13:57 (lab20909.cs.unipi.gr)
```

β) Ανακατεύθυνση εξόδου σφαλμάτων.



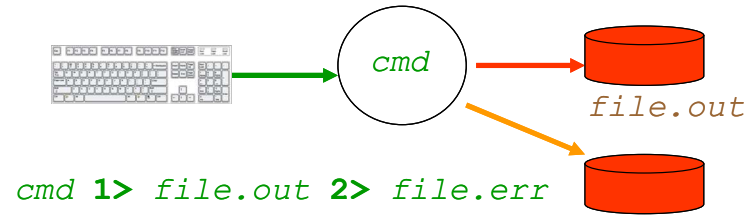
cmd **2>** *file.err*
cmd **2>>** *file.err*

Με το συμβολισμό 2> δίνεται η οδηγία για ανακατεύθυνση της εξόδου σφαλμάτων (κωδικός 2) προς αρχείο.

```
$ ls
f1  f2  f3  f4  f5  f6  file1
$ ls f1 f2 f3 f10
ls: f10: No such file or directory
f1  f2  f3
$ ls f1 f2 f3 f10 2> ls.err
f1  f2  f3
```

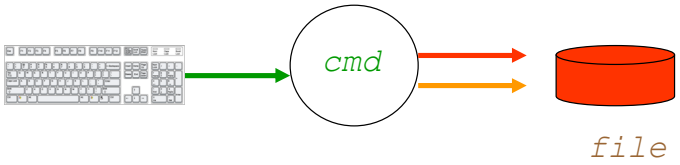
```
$ cat ls.err
ls: f10: No such file or directory
```

γ) Ανακατεύθυνση καθιερωμένης εξόδου και σφαλμάτων προς δύο ξεχωριστά αρχεία.



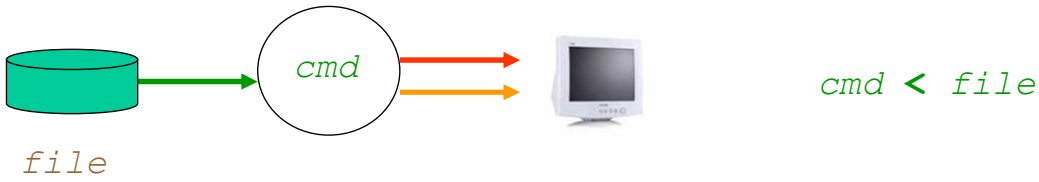
Η καθιερωμένη έξοδος οδεύει προς το αρχείο `file.out` και τα πιθανά σφάλματα τη εντολής στο αρχείο `file.err`. Μπορεί να χρησιμοποιηθεί όπως και πριν ο συμβολισμός `1>>` ή/και `2>>` για προσθήκη στα αρχεία.

δ) Ανακατεύθυνση καθιερωμένης εξόδου και σφαλμάτων προς ένα αρχείο.



Και οι δύο έξοδοι οδεύουν προς το αρχείο `file`. Η δεύτερη σύνταξη (>>) είναι για την περίπτωση της προσθήκης.

ε) Ανακατεύθυνση εισόδου από αρχείο.

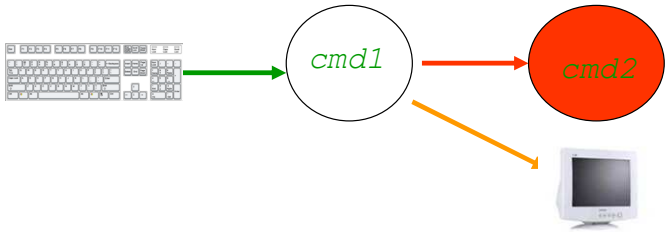


Το αρχείο `file` τροφοδοτεί με δεδομένα εισόδου την εντολή. Στο παρακάτω παράδειγμα ταξινομούνται οι γραμμές του αρχείου `file.to.sort` και το αποτέλεσμα εμφανίζεται στην οθόνη.

```
$ cat file.to.sort
123
890
yud
567
adfe ferf
wxc
c56
$ sort < file.to.sort
123
567
890
adfe ferf
c56
wxc
yud
```

4. Σωληνώσεις.

α) Απλή σωλήνωση.

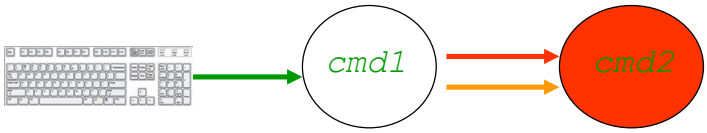


cmd1 | cmd2

Η εντολή *cmd1* παίρνει πιθανώς είσοδο από το πληκτρολόγιο, και στέλνει τα αποτελέσματά της για επί πλέον επεξεργασία στην εντολή *cmd2*. Τα πιθανά σφάλματα όμως της εντολής *cmd1* εμφανίζονται στην οθόνη.
Το πιο κάτω παράδειγμα εμφανίζει μόνο τα αρχεία του καταλόγου `/etc` που περιέχουν την ακολουθία χαρακτήρων `user`.

```
$ ls /etc | grep user
adduser.conf
deluser.conf
```

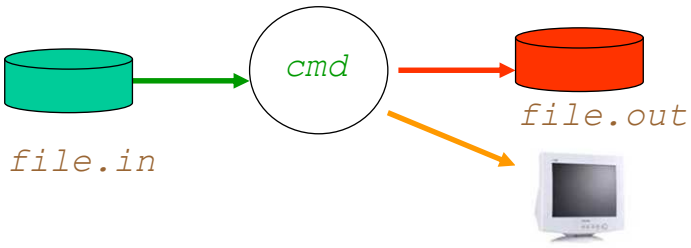
β) Σωλήνωση εξόδου και σφαλμάτων.



cmd1 |& cmd2

5. Σύνθετοι Συνδυασμοί.

α) Ανακατεύθυνση εισόδου και εξόδου.



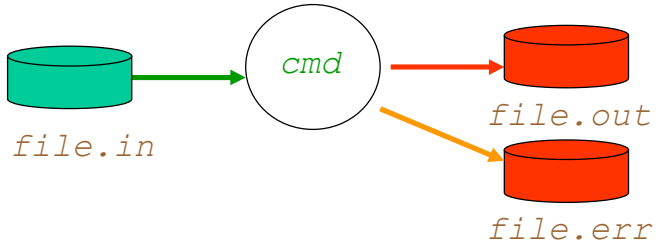
cmd < file.in > file.out

Χρησιμοποιώντας το αρχείο `file.to.sort` από προηγούμενο παράδειγμα με τον παρακάτω συνδυασμό εντολών δημιουργείται το αρχείο `file.sorted` που περιέχει τις γραμμές του πρώτου αρχείου ταξινομημένες. Στην οθόνη δεν εμφανίζεται τίποτα, εκτός από πιθανά σφάλματα.

```
$ sort < file.to.sort > file.sorted
$ cat file.sorted
123
567
890
adfe ferf
c56
```

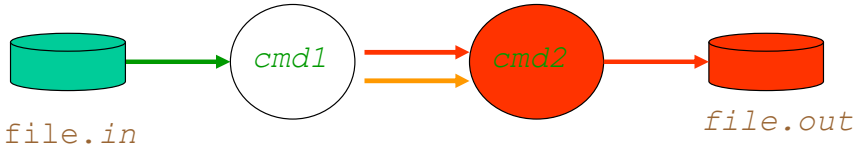
```
wxc
yud
```

β) Ανακατεύθυνση εισόδου, εξόδου και σφαλμάτων.




```
cmd < file.in > file.out 2> file.err
```

γ) Ανακατεύθυνση εισόδου, εξόδου και σωλήνωση.

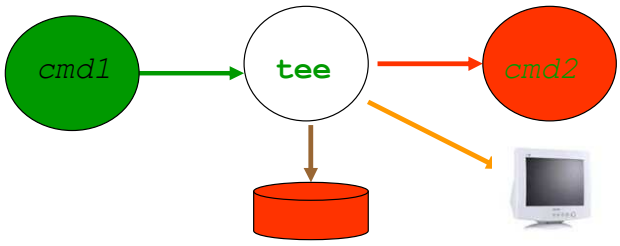



```
cmd < file.in |& cmd2 > file.out
```

Η εντολή *cmd1* παίρνει δεδομένα από το αρχείο *file.in* προωθεί τα αποτελέσματά της στην εντολή *cmd2* και η εντολή *cmd2* γράφει τα δικά της στο αρχείο *file.out*.

δ) Σωλήνωση και ταυτόχρονη καταγραφή σε αρχείο.

Υπάρχουν περιπτώσεις που είναι επιθυμητή η προώθηση δεδομένων από ένα πρόγραμμα προς ένα άλλο, αλλά ταυτόχρονα να γίνεται καταγραφή των δεδομένων αυτών και σε αρχείο. Αυτό επιτυγχάνεται με την εντολή **tee** *file* η οποία στην ουσία κάνει ότι και η **cat** (αντιγράφει την είσοδο στην έξοδο) αλλά ταυτόχρονα καταγράφει και την είσοδο στο αρχείο *file*.
Έτσι, μπορεί να παρεμβληθεί ανάμεσα σε δύο εντολές για να πραγματοποιηθεί η σωλήνωση όπως περιγράφεται στο σχήμα.




```
cmd1 | tee file | cmd2
```

Οι συνδυασμοί δεν περιορίζονται στους παραπάνω, με γνώση των βασικών λειτουργιών μπορεί να γίνουν επί πλέον συνδυασμοί όπως η σωλήνωση ανάμεσα σε περισσότερες από δύο εντολές κ.τ.λ.

Εργαστηριακές Ασκήσεις

Άσκηση 1.

1. Μετρήστε πόσα αρχεία υπάρχουν στον κατάλογο `/etc` χρησιμοποιώντας την εντολή **`wc`** (`|`)
2. Δημιουργήστε δύο αρχεία με όνομα `f1` και `f2` . Δοκιμάστε το αποτέλεσμα της εντολής **`ls -l f1 f2 f3`** .
3. Κάνετε επανακατεύθυνση εξόδου της προηγούμενης εντολής στο αρχείο `ls.out` (`>`)
4. Κάνετε επανακατεύθυνση εξόδου της προηγούμενης εντολής με προσθήκη (append) στο αρχείο `ls.out` (`>>`)
5. Κάνετε επανακατεύθυνση σφάλματος στο αρχείο `ls.err` (`2>`)
6. Κάνετε επανακατεύθυνση σε δύο ξεχωριστά αρχεία `ls.out` και `ls.err` (`1> 2>`)
7. Δημιουργήστε ένα μικρό αρχείου χρησιμοποιώντας την εντολή **`cat`** (ανακατεύθυνση σε έξοδο).
8. Δημιουργήστε με έναν editor ένα αρχείο 15 γραμμών. Κατόπιν χρησιμοποιήστε ανακατευθύνσεις και το φίλτρο **`sort`** για να παράγετε ένα αρχείο που έχει διαδοχικά τις ίδιες γραμμές ταξινομημένες λεξικογραφικά (`<` και `>` ή `>` και `>`).

Επί πλέον ασκήσεις.

2. Με ποιο συνδυασμό φίλτρων και σωληνώσεων εμφανίζονται οι γραμμές m εώς n ενός αρχείου ;

3. Να βρεθεί συνδυασμός εντολών που αποθηκεύει σε αρχείο όλους τους καταλόγους του συστήματος αρχείων που έχουν παρέχουν εξουσιοδότηση `rwxrwxrwx`.

4. Το αρχείο `/etc/passwd` περιέχει τους λογαριασμούς των χρηστών σε ένα σύστημα Linux/Unix. Κάθε γραμμή του αρχείου χωρίζεται σε πεδία με διαχωριστικό το `:` . Τα πεδία είναι κατα σειρά : username, password, uid, gid, περιγραφή χρήστη, home directory, shell. Οι σειρές είναι ταξινομημένες κατά αύξουσα σειρά uid. Με μία ακολουθία εντολών δημιουργήστε ένα αρχείο που θα περιέχει σε κάθε γραμμή μόνο τα username ταξινομημένα κατά αύξουσα σειρά.

5. Το κρυπτογράφημα του Καίσαρα είναι μία αρχαία και απλή μέθοδος κρυπτογράφησης κειμένου που συνίσταται στην αντικατάσταση κάθε γράμματος του αλφαβήτου μέ ένα διαφορετικό, με την αντιστοιχιση να είναι «1-1» και επί, έτσι ώστε να είναι δυνατή η αποκρυπτογράφηση. Η αντιστοίχιση λέγεται κλειδί της κρυπτογράφησης. Η αποκρυπτογράφηση γίνεται με την αντίστροφη διαδικασία.
   Χρησιμοποιώντας σωλήνωση και ανακατεύθυνση, κρυπτογραφήστε ένα αρχείο που περιέχει μόνο αριθμούς σε μορφή ASCII σε ένα άλλο αρχείο της ίδιας μορφής. Κατόπιν με την ίδια μέθοδο αποκρυπτογραφήστε το σε ένα τρίτο αρχείο. Ελέξτε ότι το αρχικό αρχείο είναι ίδιο με το τελικό χρησιμοποιώντας την εντολή **`diff`**.

Λειτουργικό Σύστημα Linux
Ενότητα V
ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΦΛΟΙΟΥ – SHELL SCRIPTS

1. Εισαγωγή.

Ο όρος *προγραμματισμός φλοιού* (*shell programming*) σημαίνει την συγγραφή από ένα χρήστη και εκτέλεση από τον φλοιό ειδικών *προγραμμάτων φλοιού* (**shells scripts**), δηλαδή αρχείων κειμένου που περιέχουν κάποιας μορφής γλώσσα προγραμματισμού την οποία μπορεί να ερμηνεύσει απ' ευθείας ο φλοιός. Ο προγραμματισμός φλοιού επαυξάνει κατά πολύ τις δυνατότητες που έχει το Λ.Σ. Linux γιατί ουσιαστικά με λίγο κόπο μπορούν να δημιουργηθούν νέες εντολές, χωρίς να χρειάζεται κανείς να καταφύγει σε πολύπλοκες γλώσσες προγραμματισμού όπως ή C, ούτε και να γνωρίζει λεπτομέρειες για την οργάνωση του Linux ή να μαθαίνει διάφορες κλήσεις συστήματος (system calls) όπως π.χ. πως ανοίγει, διαβάζεται και κλείνει ένα αρχείο σε γλώσσα C.
Τα προγράμματα φλοιού περιέχουν εντολές του Linux, κλήσεις άλλων εκτελέσιμων αρχείων ή προγραμμάτων φλοιού, μεταχαρακτήρες, μεταβλητές και φυσικά δομές ελέγχου ροής (όπως if-then-else,case,for και while loops). Το πρόγραμμα τρέχει απ' ευθείας από τον φλοιό στη μορφή κειμένου που ήδη βρίσκεται, χωρίς να μεσολαβήσει καμμία μεταγλώτιση, δηλαδή η γλώσσα προγραμματισμού εκτελείται μέσω ενός *διερμηνευτή* (**interpreter**).
Προγράμματα φλοιού υπάρχουν ήδη σε μία εγκατάσταση Linux, αφού κάποιες εντολές είναι τέτοιας μορφής και όχι δυαδικός κώδικας μηχανής. Επίσης, προγράμματα φλοιού είναι και κάποια αρχεία που αρχικοποιούν το περιβάλλον του χρήστη όποτε αυτός συνδέεται στο σύστημα. Επομένως, είναι απαραίτητο να ξέρει κανείς τα στοιχειώδη για τον προγραμματισμό φλοιού, ώστε να είναι τουλάχιστον σε θέση να αλλάξει το περιβάλλον του σε περίπτωση που αυτό απαιτηθεί, όπως όταν γίνεται η εγκατάσταση μίας εφαρμογής που συνήθως απαιτεί τον ορισμό κάποιων μεταβλητών σε αυτά τα αρχεία.
Επειδή το Linux προσφέρει διάφορους φλοιούς (sh, csh, bash, ksh, tcsh κ.α.) που έχουν ελαφρές διαφοροποιήσεις στο συντακτικό τους, θα πρέπει να είναι γνωστός ο φλοιός που χρησιμοποιεί ο χρήστης (φαίνεται στο αρχείο `/etc/passwd`) και η συγγραφή να γίνεται για αυτόν τον τύπο φλοιού. Υπάρχει τρόπος παράκαμψης που θα περιγραφεί αργότερα, ώστε το πρόγραμμα να τρέξει σε διαφορετικό φλοιό.
Έτσι, αυτά που θα περιγραφούν από εδώ και στο εξής θα αφορούν το φλοιό Bourne Again Shell (bash) που συνήθως είναι ο εξ΄ ορισμού φλοιός, τουλάχιστον για τους χρήστες του Debian Linux. Λεπτομέρειες για όλες τις δυνατότητες του bash φλοιού μπορούν να βρεθούν με **`man bash`**.

2.Τρόπος εκτέλεσης – Απλά παραδείγματα.

Υπάρχουν δύο τρόποι εκτέλεσης για ένα πρόγραμα φλοιού. Ας δοκιμάσουμε ένα απλό πρόγραμμα που ουσιαστικά δεν έχει προγραμματιστικές δομές, αλλά είναι παράθεση απλών εντολών Linux. Το πρόγραμμα δίνεται παρακάτω και υποθέτουμε ότι είναι ένα αρχείο κειμένου με όνομα `test1`. Θα πρέπει να δημιουργηθεί με τη χρήση κάποιου διορθωτή κειμένου π.χ. **`pico`** και να αποθηκευτεί σε έναν κατάλογο που έχει πρόσβαση ο χρήστης, π.χ. τον προσωπικό του.

```
#!/bin/bash
date
uname -a
whoami
```

Η πρώτη γραμμή δηλώνει ποιος φλοιός θα πρέπει να ερμηνεύσει το πρόγραμμα. Αν δεν υπάρχει τότε, η ερμηνεία γίνεται από τον τρέχοντα φλοιό που χρησιμοποιεί ο χρήστης. Οι υπόλοιπες τρεις γραμμές είναι γνωστές εντολές που θα εκτελεστούν με τη σειρά.
Το αρχείο θα πρέπει να παρέχει δικαιώματα εκτέλεσης (x) προς αυτόν ο οποίος θα προσπαθήσει να το τρέξει. Ένας σίγουρος τρόπος είναι να δοθούν δικαιώματα εκτέλεσης (και ανάγνωσης φυσικά) πρός όλες τις κατηγορίες, δηλαδή **chmod 755 test**.
Κατόπιν το αρχείο καλείται είτε δίνοντας την πλήρη διαδρομή του (π.χ. **~/test** αν βρίσκεται στον προσωπικό κατάλογο) είτε με σχετική διαδρομή, έστω και αν ο τρέχων κατάλογος είναι ο κατάλογος στον οποίο βρίσκεται το αρχείο, δηλαδή **./test**. Κανονικά θα περίμενε κανείς εφ' όσον βρίσκεται στο κατάλογο που περιέχει το αρχείο να αρκεί η απλή αναφορά του αρχείου, αυτό όμως δε συμβαίνει όταν πρόκειται για εκτέλεση αρχείου για λόγους ασφαλείας. Υπάρχει τρόπος αλλάζοντας μία μεταβλητή που περιγράφει τους καταλόγους στους οποίους γίνεται αναζήτηση εκτελέσιμων αρχείων (PATH), η εκτέλεση να πετύχει με απλή αναφορά του ονόματος.

```
$ ls -l test
-rw-r--r-- 1 adp adp 21 2008-06-01 01:40 test
$ ./test
-bash: ./test: Permission denied
$ chmod 755 test
$ ./test
Sun Jun  1 02:06:29 EEST 2008
Linux erasmus 2.6.15-26-server #1 SMP Fri Sep 8 21:00:37 UTC 2006
i686 GNU/Linux
adp
$ test
$
```

Ο δεύτερος τρόπος εκτέλεσης είναι με την εντολή **source test**. Στη δεύτερη περίπτωση δε χρειάζεται στο αρχείο να υπάρχουν δικαιώματα για εκτέλεση.

3. Προγράμματα αρχικοποίησης.

Όταν αρχίζει μία σύνδεση σε ένα Linux σύστημα γίνεται μία αρχικοποίηση της σύνδεσης με βάση κάποια αρχεία τα οποία είναι στην ουσία προγράμματα φλοιού. Η συνδέσεις που είναι *αλληλεπιδραστικές* (**interactive**) μπορεί να είναι δύο ειδών : login και non-login. Μία κανονική σύνδεση είναι τύπου login. Παραδείγματα non-login συνδέσεων είναι μία σύνδεση ftp ή μία ενεργοποίηση υποφλοιού (subshell).
Γενικά, μία φυσιολογική σύνδεση είναι login σύνδεση. Σε αυτή την περίπτωση τα αρχεία που διαβάζονται είναι με τη σειρά τα εξής :
α) /etc/profile : Όπως φαίνεται και από τη διαδρομή του είναι ένα αρχείο το οποίο είναι κοινό για όλους του χρήστες (system-wide) και μπορεί να αλλάξει μόνο από το διαχειριστή.
β) ~/.bash_profile : Είναι κρυφό αρχείο (πρώτος χαρακτήρας η τελεία) και πρέπει να υπάρχει στον προσωπικό κατάλογο του κάθε χρήστη. Εκεί μπορεί να επεμβαίνει ο χρήστης και να αλλάζει σύμφωνα με τις επιθυμίες του το περιβάλλον του.
γ) ~/.bash_login : Αυτό το αρχείο διαβάζεται εφ' όσον δεν υπάρχει το ~/.bash_profile.
δ) ~/.profile : Αυτό το αρχείο διαβάζεται εφ' όσον δεν υπάρχει το ~/.bash_login.
Η αντίστοιχη σειρά για την περίπτωση των non-login shells είναι : /etc/bash.bashrc και ~/.bashrc.
Κατόπιν εμφανίζεται το σήμα προτροπής (prompt) και ο χρήστης μπορεί να χρησιμοποιήσει το σύστημα.
Υπάρχει η δυνατότητα ακριβώς πριν να γίνει έξοδος από το σύστημα να τρέξει ένα επί πλέον πρόγραμμα φλοιού το οποίο πρέπει να βρίσκεται στον προσωπικό κατάλογο με όνομα ~/.bash_logout. Π.χ. να γίνεται καθαρισμός της οθόνης με την εντολή **clear** έτσι ώστε ο

επόμενος χρήστης του τερματικού να μην δει τις εντολές που χρησιμοποιήθηκαν πριν από αυτόν ή να γίνονται λειτουργίες κατασκευής εφεδρικών αντιγράφων (backup).

4. Μεταβλητές.

Χωρίς μεταβλητές είναι αδύνατον να γίνει έστω και στοιχειώδης προγραμματιστική λειτουργία. Ο φλοιός προσφέρει δύο ειδών μεταβλητές για χρήση, τις *επιλογές* (**options**) και τις κανονικές μεταβλητές. (**variables**). Οι επιλογές είναι μεταβλητές που παίρνουν μόνο δύο τιμές, `on` και `off`, δηλαδή είναι κάτι σαν boolean μεταβλητές. Οι κανονικές μεταβλητές, που από εδώ και στο εξής θα αναφέρονται απλώς μεταβλητές, μπορούν να είναι τύπου συμβολοσειράς (string) ή ακεραίου. Ο τύπος της μεταβλητής δεν δηλώνεται αλλά αποφασίζεται ο τύπος της με βάση τους τελεστές που εφαρμόζονται πάνω στην μεταβλητή. Έτσι μία μεταβλητή μπορεί σε ένα σημείο του προγράμματος να είναι συμβολοσειρά και σε άλλο να είναι ακέραιος.
Η επιλογές είναι προκαθορισμένες και επηρεάζουν ορισμένες λειτουργίες του περιβάλλοντος. Π.χ. η επιλογή `history` καθορίζει αν θα κρατείται ιστορικό εντολών για να το δείχνει κατόπιν η εντολή **`history`**. Η εμφάνιση όλων των επιλογών μπορεί να γίνει με την εντολή **`set -o`**. Για να παρουσιαστεί μία συγκεκριμένη επιλογή θα πρέπει να γίνει συνδυασμός με το φίλτρο **`grep`**, π.χ. **`set | grep history`**. Μία επιλογή με όνομα *`option`* τίθεται σε κατάσταση `on` με **`set -o`** *`option`* και σε κατάσταση off με **`set +`** *`option`*.
Οι μεταβλητές εμφανίζονται όλες με την εντολή **`set`**. Η ανάγνωση μίας μεταβλητής γίνεται με απλή αναφορά του ονόματός της βάζοντας το σύμβολο **`$`** πριν το όνομα, ενώ η εκχώρηση τιμής σε αυτήν γίνεται βάζοντας το **`=`**, χωρίς κενά, ανάμεσα σε αυτήν και την νέα τιμή που θέλουμε. Το παράδειγμα δείχνει το συνδυασμό αυτό χρησιμοποιώντας την εντολή **`echo`** η οποία εμφανίζει στην οθόνη ότι την ακολουθεί. (Είναι το αντίστοιχο της printf της γλώσσας C).

```
$ set | grep newVar
$ echo $newVar

$ newVar=12345
$ echo $newVar
12345
$ newVar=abcd
$ echo $newVar
abcd
```

Στην αρχή η μεταβλητή `newVar` δεν είναι ορισμένη όπως φαίνεται από τις δύο πρώτες εντολές, αλλά αρκεί μόνο η κατάχωρηση, χωρίς δήλωση, για να οριστεί.

Ενσωματωμένες μεταβλητές.

Οι *ενσωματωμένες μεταβλητές* (**built-in variables**) είναι ήδη ορισμένες από το σύστημα, αλλά μπορούν να αλλαχθούν με επέμβαση στα αρχεία αρχικοποίησης. Είναι αυτές που κυρίως επηρεάζουν το περιβάλλον του χρήστη. Μία από τις πιο σημαντικές ενσωματωμένες μεταβλητές είναι η `PATH` η οποία καθορίζει τους καταλόγους στους οποίους γίνεται αναζήτηση εκτελέσιμων προγραμμάτων. Αν δοθεί μία λέξη προς το σύμβολο αναμονής και αυτή δεν είναι εσωτερική εντολή, ο φλοιός θεωρεί ότι είναι εντολή η οποία θα πρέπει να αναζητηθεί με τη σειρά στους καταλόγους που περιγράφονται στην `PATH`. (Το σύμβολο `:` είναι το διαχωριστικό των διαδρομών).

```
$ echo $PATH
/home/adp/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sb
in:/bin:/usr/bin/X11:/usr/games
$ PATH=
$ date
-bash: date: No such file or directory
```

```
$ who
-bash: who: No such file or directory
```

Όπως φαίνεται παραπάνω, αρχικά η PATH έχει σαν περιεχόμενο μία σειρά από απόλυτες διαδρομές καταλόγων που είναι γνωστό ότι περιέχονται εντολές (π.χ. /bin). Κατόπιν η PATH τίθεται στην τιμή του μηδενικού string με αποτέλεσμα πολύ κοινές εντολές να μην μπορούν να βρεθούν και να έχουμε μηνύματα λάθους.
Ακολουθεί ένας πίνακας που περιγράφει μερικές σημαντικές ενσωματωμένες μεταβλητές.

Μεταβλητή	Περιγραφή
PATH	Οι κατάλογοι που ψάχνει για εντολές ο φλοιός
MAIL	Το αρχείο των εισερχόμενων μηνυμάτων
MAILCHECK	Χρονική περίοδος ελέγχου νέων εισερχόμενων μηνυμάτων
MANPATH	Κατάλογοι που περιέχουν αρχεία βοήθειας εντολών (man pages)
PS1	Μορφή συμβόλου αναμονής (prompt)
CDPATH	Αντίστοιχη της PATH, αλλά για αναζήτηση καταλόγων (**cd**)
HOME	Ο προσωπικός κατάλογος χρήστη
LOGNAME	Το όνομα χρήστη που τρέχει τον τρέχοντα φλοιό
PWD	Ο τρέχων κατάλογος
BASH	Ο τρέχων φλοιός
SHELL	Ο εξ΄ορισμού φλοιός (δηλαδή ποιος είναι ορισμένος στο /etc/passwd)
RANDOM	Τυχαίος αριθμός (0-32767)

Μεταβλητές χρήστη (user variables).

Αυτές μπορούν να ορισθούν εξ' αρχής στον φλοιό ή μέσα σε ένα πρόγραμμα φλοιού. Η εμβέλειά τους είναι για τον τρέχοντα φλοιό ή για το πρόγραμμα φλοιού μόνο. Όταν η εμβέλεια επεκτείνεται και σε άλλα προγράμματα που καλούνται από το πρόγραμμα φλοιού τότε λέγονται μεταβλητές περιβάλλοντος (environment variables). Αυτό μπορεί να γίνει ορίζοντάς τις με **$***var*=*value* ; **export** *var* ή κατ΄ ευθείαν με export **$***var*=*value*.
Για να είναι ορατές στον φλοιό όταν τελειώσει το πρόγραμμα θα πρέπει αυτό να έχει κληθεί με την εντολή **source** ή οι μεταβλητές να έχουν οριστεί στο αρχείο ~/.bashrc. Όλες οι μεταβλητές περιβάλλοντος εμφανίζονται με την εντολή **env**. Σε αυτές συμπεριλαμβάνονται και οι ενσωματωμένες.

Τρόποι εκχώρησης μίας τιμής σε μία μεταβλητή.

Εκτός από την εκχώρηση μίας σταθεράς σε μία μεταβλητή υπάρχουν και άλλο τρόποι που αυξάνουν σημαντικά την ευελιξία προγραμματισμού.

α) Εκχώρηση μίας μεταβλητής σε μία άλλη

var2=$var1

Εκχωρείται η τιμή της var1 στην var2.

β) Συνένωση τιμών μεταβλητών και σταθερών

```
var3=$var1$var2abcdef
```

Η var3 θα έχει τη συνένωση (concatenation) των συμβολοσειρών των var1, var2 και της σταθεράς abcdef. Εδώ υποτίθεται ότι δεν έχουν οριστεί μεταβλητές με ονόματα var2a, var2ab κτ.λ.

γ) Εκχώρηση από τερματικό με αλληλεπίδραση με τον χρήστη.

read var4

Το πρόγραμμα διακόπτεται και αναμένει εισαγωγή δεδομένων από το πληκτρολόγιο που θα περαστούν στην var4.

δ) Εκχώρηση από τις παραμέτρους του προγράμματος φλοιού.

```
var5=$1
var6=$2
```

Το πρόγραμμα φλοιού μπορεί να κληθεί μαζί με παραμέτρους, όπως ακριβώς και μία εντολή του Linux. Η κάθε παράμετρος (χωρισμένη από τις άλλες με ένα κενό) μπορεί να εκχωρηθεί σε κάποια μεταβλητή. Η αναφορά στις παραμέτρους του προγράμματος φλοιού γίνεται με το συμβολισμό `$n`, όπου *n* είναι ένας ακέραιος που δηλώνει τη σειρά της παραμέτρου. Αν υποτεθεί ότι οι δύο παραπάνω γραμμές εκχωρήσεων βρίσκονταν σε ένα πρόγραμμα φλοιού με όνομα `script1` τότε η κλήση του προγράμματος με **`script1 apple orange`** θα είχε ως αποτέλεσμα η μεταβλητή `var5` να έχει την τιμή `apple` και η `var6` την τιμή `orange`.

ε) Εκχώρηση με υποκατάσταση εξόδου εντολής.

Μία σημαντική δυνατότητα που υπάρχει είναι αυτή της εκχώρησης σε μία μεταβλητή του αποτελέσματος που παράγει στην καθιερωμένη έξοδό της μία εντολή. Αυτό λέγεται *υποκατάσταση εξόδου εντολής* (**command substitution**). Επιτυγχάνεται με ειδική σύνταξη περικλείοντας την επιθυμητή εντολή με τις πιθανές επιλογές και ορίσματά της σε ανάποδα εισαγωγικά.

```
var7=`ls`
```

Η μεταβλητή θα περιέχει τα ονόματα των αρχείων του τρέχοντος καταλόγου χωρισμένα με κενά.

Τα παρακάτω παραδείγματα συνοψίζουν το θέμα των εκχωρήσεων σε μεταβλητές :

Πρόγραμμα `tree2file`

```
#!/bin/bash
dumpDir=$1
now=`date ´+%d%m%Y´`
fileName=$dumpDir$now
ls -lR $dumpDir > $fileName
```

Το πρόγραμμα `tree2file` καλείται με μία παράμετρο που πρέπει να είναι ένας κατάλογος ή μία διαδρομή σε έναν κατάλογο. Η διαδρομή καταχωρείται στη μεταβλητή `dumpDir`. Η μεταβλητή `now` περιέχει την τρέχουσα ημερομηνία σε μορφή ΗΗΜΜΕΕ (π.χ. 01062008). Η μεταβλητή `filename` είναι η συνένωση του καταλόγου που δώθηκε και της τρέχουσας ημερομηνίας. Τελικά σε ένα αρχείο που θα γραφεί στον κατάλογο από τον οποίο κλήθηκε η `tree2file` και με όνομα αρχείου τη συνένωση του καταλόγου που δόθηκε σαν όρισμα και της τρέχουσας ημερομηνίας, θα αποθηκευτούν με αναδρομικό τρόπο όλα τα ονόματα αρχείων και υποκατάλογων του καταλόγου που δόθηκε σαν όρισμα.

Πρόγραμμα `cprecent`

```
#!/bin/bash
recent=`ls -t $1 | head -1`
cp $1/$recent $2
echo "Most recent file $recent copied"
echo "From: $1 To: $2 dir"
```

Το πρόγραμμα `cprecent` δέχεται δύο ορίσματα. Έναν κατάλογο αφετηρίας και έναν κατάλογο προορισμού και αντιγράφει από τον κατάλογο αφετηρίας στον κατάλογο προορισμού το πιο πρόσφατα ενημερωμένο αρχείο.

Ειδικές μεταβλητές.

Ο χαρακτήρας του δολαρίου (`$`) έχει μία ειδική σημασία όταν προηγείται άλλων χαρακτήρων, όπως όταν εμφανίζεται η παράσταση `$1` που σημαίνει το πρώτο όρισμα του προγράμματος. Υπάρχει διάφοροι συνδυασμοί του `$` με άλλους χαρακτήρες που έχουν μία ιδιαίτερη σημασία ως μεταβλητές

σε ένα πρόγραμμα φλοιού. Ο παρακάτω πίνακας δείχνει κάποιους συμβολισμούς τέτοιων ειδικών μεταβλητών μαζί με τη σημασία τους.

Μεταβλητή	Περιγραφή
`$1,$2,...`	Τα ορίσματα που δίνονται στο πρόγραμμα φλοιού
`$0`	Το όνομα του προγράμματος φλοιού
`$#`	Ο αριθμός των ορισμάτων που έχουν δοθεί.
`$*`	Όλα τα ορίσματα σε ένα string με διατήρηση όλων των κενών
`$@`	Όλα τα όρισμα σε ένα string με ένα μόνο κενό ανάμεσά τους
`$?`	Κωδικός εξόδου της τελευταίας εντολής που έτρεξε από το πρόγραμμα
`$$`	pid της τρέχουσας διεργασίας (δηλ. του ίδιου του προγράμματος φλοιού)
`$!`	pid της τελευταίας διεργασίας παρασκηνίου (background job)

Στο bash shell δεν υπάρχει περιορισμός στον αριθμό των ορισμάτων που μπορούν να αναγνωστούν μέσα σε ένα πρόγραμμα φλοιού. Σε άλλους φλοιούς όμως υπάρχει ο περιορισμός ότι μόνο μέχρι εννέα ορίσματα μπορούν να διαβαστούν (δηλαδή μπορεί να χρησιμοποιηθεί η μεταβλητή `$9` αλλά όχι η `$10`). Αυτός ο περιορισμός παρακάμπτεται με την οδηγία `shift` η οποία κάθε φορά που δίνεται μετακινεί τα ορίσματα αριστερά κατά μία θέση. Έτσι, η πρώτη κλήση της `shift` θα έχει ως αποτέλεσμα το πρώτο όρισμα να χαθεί, το δεύτερο να πάει στην μεταβλητή `$1`, το τρίτο στη μεταβλητή `$2` κ.ο.κ.. Η επόμενη κλήση της `shift` θα συνεχίσει τη διαδικασία με μία επί πλέον ολίσθηση. Αυτή η λειτουργία υποστηρίζεται και από τον φλοιό bash ανεξάρτητα από το αν χρειάζεται.

Ειδικοί χαρακτήρες.

Σε αυτήν την παράγραφο περιγράφεται η σημασία που έχουν σε ένα πρόγραμμα φλοιού ορισμένοι ειδικοί χαρακτήρες. Κάποιοι από αυτούς έχουν ήδη παρουσιαστεί σε ορισμένα παραδείγματα.

- Χαρακτήρας `#`

Ότι ακολουθεί αυτόν τον χαρακτήρα σε μία γραμμή θεωρείται σχόλιο και δεν λαμβάνεται υπ' όψη.

- Χαρακτήρας `;`

Είναι το διαχωριστικό των εντολών. Σε μία γραμμή προγράμματος (ακόμα και στο αλληλεπιδραστικό περιβάλλον τη γραμμής εντολών) μπορούν να μπουν περισσότερες από μία εντολές εφ' όσον διαχωρίζονται από αυτόν τον χαρακτήρα.

- Χαρακτήρας `\` (backslash)

Η σημασία του είναι ίδια με αυτήν που έχει στα ονόματα αρχείων, δηλαδή κάνει τον επόμενο χαρακτήρα να ερμηνεύεται κυριολεκτικά και όχι με την πιθανώς ειδική σημασία που έχει.

- Χαρακτήρες `' '` (απλά εισαγωγικά – strong quoting)

Όπως και πριν, οτιδήποτε υπάρχει ανάμεσά τους ερμηνεύεται κυριολεκτικά. Τυχόν ειδικοί χαρακτήρες χάνουν τη σημασία τους.

- Χαρακτήρες `" "` (διπλά εισαγωγικά – partial quoting)

Σχεδόν οτιδήποτε υπάρχει ανάμεσά τους ερμηνεύεται κυριολεκτικά. Τυχόν ειδικοί χαρακτήρες χάνουν τη σημασία τους με ορισμένες εξαιρέσεις. Εξαιρείται ο χαρακτήρας του δολαρίου, δηλαδή τυχόν μεταβλητές που υπάρχουν διαβάζονται. Επίσης ο χαρακτήρας διαφυγής και τα απλά εισαγωγικά διατηρούν τη σημασία τους.

- Χαρακτήρες `` ` ` `` (ανάποδα εισαγωγικά – command substitution)

Η τιμή της παράστασης που αποτελείται από ανάποδα εισαγωγικά είναι το αποτέλεσμα της εξόδου της εντολής που βρίσκεται μέσα.

- Χαρακτήρας `$` (ανάγνωση μεταβλητής)

Μπαίνοντας μπροστά από το όνομα μίας μεταβλητής υπολογίζει το όνομά της.

- Χαρακτήρες `${ }` (ανάγνωση μεταβλητής)

Υπολογίζει την τιμή της μεταβλητής που υπάρχει ανάμεσα στα άγκιστρα. Υπάρχει μία λεπτή διαφορά ανάμεσα σε αυτήν την σύνταξη και την προηγούμενη. Αν μία μεταβλητή με όνομα

variable έχει τιμή 123 και μία άλλη με όνομα var την ίδια τιμή τότε, η παράσταση $variable θα δώσει την τιμή της 123, ενώ η ${var}iable την τιμή 123iable.

- Χαρακτήρας :

Παριστάνει την κενή εντολή (no operation), δηλαδή μία εντολή που δεν κάνει τίποτα. Έχει και άλλες χρήσεις, όπως σαν διαχωριστικό μονοπατιών στη μεταβλητή PATH.

- Χαρακτήρες [ ]

Ισοδυναμεί με την εντολή **test** , αλλά και περιγράφει περιοχές τιμών, π.χ. [a-f].

- Χαρακτήρας *

Ο γνωστός μεταχαρακτήρας για ονόματα αρχείων

- Χαρακτήρας ?

Ο γνωστός μεταχαρακτήρας για ονόματα αρχείων, αλλά έχει και άλλες χρήσεις.

Το αποτέλεσμα της εκτέλεσης του πιο κάτω προγράμματος φλοιού με όνομα shvar διευκρινίζει το αποτέλεσμα της χρήσης μερικών από τους ειδικούς χαρακτήρες.

Πρόγραμα shvar

```
#!/bin/bash
# Displays some variables
echo "Search path is : $PATH"
echo 'Search path is : $PATH'
echo "Var \$HOSTNAME is :    $HOSTNAME"
echo Var \$HOSTNAME is :    $HOSTNAME
echo "My home dir is : $HOME"
echo `uname -a | cut -d" " -f3`
```

```
$ ./shvar
Search path is :
/home/adp/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
/sbin:/bin:/usr/bin/X11:/usr/games
Search path is : $PATH
Var $HOSTNAME is :    erasmus
Var $HOSTNAME is : erasmus
My home dir is : /home/adp
2.6.15-26-server
```

5. Υπολογισμοί και παραστάσεις.

Έχοντας στη διάθεσή μας μεταβλητές και σταθερές μπορούμε να κάνουμε διάφορες πράξεις μεταξύ τους όπως πράξεις μεταξύ συμβολοσειρών (strings) ή αριθμών. Έχει τονιστεί ότι οι μεταβλητές δεν έχουν τύπο, έτσι ανάλογα με τους τελεστές που χρησιμοποιούμε αποφασίζεται εκείνη την ώρα αν το τελούμενο είναι συμβολοσειρά ή αριθμός. Οι αριθμοί που υποστηρίζονται από το bash είναι μόνο ακέραιοι προσημασμένοι των 32 bits, δηλαδή οι αριθμοί μεταξύ -2^{31} και 2^{31} που αντιστοιχούν στους αριθμούς -2147483648 και +2147483647. Δεν υπάρχει δυνατότητα για πράξεις μεταξύ αριθμών κινητής υποδιαστολής παρά μόνο αν χρησιμοποιηθεί το πρόγραμμα **bc** που είναι ουσιαστικά μία αριθμομηχανή.

Παραστάσεις συμβολοσειρών.

Στον πίνακα που ακολουθεί δίνονται κάποιες παραστάσεις με πράξεις συμβολοσειρών μαζί με τη σημασία τους. Σαν string αναφέρεται μία μεταβλητή η οποία θεωρείται ότι είναι συμβολοσειρά. Το substring είναι μία σταθερή συμβολοσειρά ή μία μεταβλητή ή μία κανονική έκφραση (δεν έχει αναλυθεί ακόμα). Το position είναι μία αριθμητική σταθερά ή μία μεταβλητή που περιέχει έναν ακέραιο. Το ίδιο ισχύει και με το length.

Παράσταση	Σημασία
`${#string}`	Το μήκος της μεταβλητής
`${string:position}`	Εξαγωγή χαρακτήρων από τη θέση `position` και δεξιότερα.
`${string:position:length}`	Εξαγωγή χαρακτήρων από τη θέση position και για μήκος `length`
`${string#substring}`	Αποκοπή από την αρχή του `string` του μικρότερου δυνατού μέρους που ταιριάζει με το `substring`
`${string##substring}`	Αποκοπή από την αρχή του `string` του μεγαλύτερου δυνατού μέρους που ταιριάζει με το `substring`
`${string%substring}`	Αποκοπή από το τέλος του `string` του μικρότερου δυνατού μέρους που ταιριάζει με το `substring`
`${string%%substring}`	Αποκοπή από το τέλος του `string` του μεγαλύτερου δυνατού μέρους που ταιριάζει με το `substring`

Δίνονται μερικά παραδείγματα:

```
$ string=abcdefghijklmnopqrstuvwxyz
$ echo $string
abcdefghijklmnopqrstuvwxyz
$ substring=abc
$ echo ${#string}
26
$ echo ${string:5}
fghijklmnopqrstuvwxyz
$ echo ${string:$substring}
abcdefghijklmnopqrstuvwxyz
$ echo ${string#$substring}
defghijklmnopqrstuvwxyz
$ echo ${string%xyz}
abcdefghijklmnopqrstuvw
```

Υπάρχουν και αρκετές άλλες εκφράσεις για συμβολοσειρές που μπορούν να αναζητηθούν από το on-line εγχειρίδιο του bash shell (**`man bash`**).

Αριθμητικές παραστάσεις.

Με αυτές επιτυγχάνονται κοινές αριθμητικές πράξεις μεταξύ μεταβλητών και αριθμών που παριστάνουν ακεραίους. Επίσης υπάρχει η δυνατότητα για *δυαδικές πράξεις bit προς bit* μεταξύ αριθμών (**bitwise operations**).
Θεωρώντας πράξεις μεταξύ δύο τελούμενων (η επέκταση σε παραπάνω ορίσματα είναι προφανής) υπάρχουν διάφοροι τρόποι σύνταξης. Δίνεται ο πιο απλός τρόπος που υποστηρίζει το bash shell και θα αναφερθούν και άλλοι που υποστηρίζονται και από άλλους φλοιούς :
`result``=$((`*`var1 operator var2`*`))`
Στην σύνταξη αυτή *`result`* είναι μία μεταβλητή που θα πάρει το αποτέλεσμα της πράξης, *`var1`* και *`var2`* είναι μεταβλητές ή σταθερές και *`operator`* είναι ένας αριθμητικός ή λογικός τελεστής σύμφωνα με τους δύο πιο κάτω πίνακες.

Τελεστές αριθμητικών πράξεων

Τελεστής	Σημασία
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
**	Εκθετικό
%	Υπόλοιπο διαίρεσης

Τελεστές λογικών πράξεων

Τελεστής	Σημασία
&	Λογικό AND
|	Λογικό OR
^	Λογικό XOR
~	Λογικό NOT
>>	Δεξιά ολίσθηση
<<	Αριστερή ολίσθησ

Ακολουθούν μερικά παραδείγματα :

```
$ x=1
$ y=2
$ z=5
$ a=0xFFFFFFFF
$ echo $((y*z))
10
$ echo $((y*z+x+z/y))
13
$ echo $((a & x))
1
$ ((a >> 28))
15
```

Ο υπολογισμός παραστάσεων είναι γνωστός με τον όρο *αριθμητική επέκταση* (**arithmetic expansion**). Υπάρχουν και άλλες παραλλαγές σύνταξης (όπως και ορισμένοι ακόμα τελεστές που δεν αναφέρθηκαν) για την αριθμητική επέκταση οι οποίες έχουν παραμείνει για συμβατότητα με άλλος φλοιούς.
Η μία παραλλαγή γράφεται : *result*=`**expr** $*var1 operator* $*var2*` και η άλλη γράφεται : **let** *"result=var1 operator var2"*.

Υπολογισμός ορισμάτων-μεταβλητών.

Πολλές φορές χρειάζεται να υπολογιστεί η τιμή μίας μεταβλητής η οποία περιέχει άλλες μεταβλητές. Ένα παράδειγμα είναι όταν μία μεταβλητή αναφέρεται στη σειρά ενός ορίσματος του προγράμματος φλοιού. Ας υποθέσουμε ότι έχουμε ένα πρόγραμμα φλοιού `evalexample` το οποίο καλούμε με παραμέτρους ως εξής : **evalexample cherry grape peach**. Αν η μεταβλητή `i` έχει μία δεδομένη στιγμή την τιμή 2 , θέλουμε να τη χρησιμοποιήσουμε για να βρούμε τι τιμή έχει το δεύτερο όρισμα (δηλαδή `grape`) του προγράμματος φλοιού που τη χρησιμοποιεί. Η απλή αναφορά `$i` θα δώσει φυσικά 2. Το επιθυμητό είναι να δημιουργηθεί μία παράσταση που δίνει τιμή `$2`. Αν προσπαθήσουμε με `$$2` το απότέλεσμα θα είναι ο αριθμός διεργασίας που δίνει η παράσταση `$$` συνενομένος με τον χαρακτήρα `i`. Ακόμα και η `\$$i` θα δώσει αποτέλεσμα `$2` και όχι `grape`. Η λύση είναι η χρήση της εντολής `eval` ως εξής :
`eval arg=\$$i`.
Παρακάτω δίνεται το σχετικό παράδειγμα μαζί με το αποτέλεσμα εκτέλεσης του.

Πρόγραμμα `evalexample`

```
#!/bin/bash
i=1
var1=$i          ;       echo $var1
var2="$i"        ;       echo $var2
var3=$$i         ;       echo $var3
var4=\$$i        ;       echo $var4
eval var5=\$$i   ;       echo $var5
```

```
$ ./evalexample cherry grape peach
1
1
30649i
$1
cherry
```

Η **eval** μπορεί να εφαρμοστεί διαδοχικά για να υπολογίσει βαθύτερα επίπεδα μεταβλητών μέσα σε παραστάσεις όπως φαίνεται από το επόμενο παράδειγμα

```
$ a='$b'
$ b='$c'
$ c=d
$ echo $a
$b
$ eval echo $a
$c
$ eval eval echo $a
d
```

6. Έλεγχος ροής προγράμματος.

Χωρίς τη δυνατότητα για λήψη αποφάσεων λίγες δυνατότητες θα υπήρχαν. Το bash shell προσφέρει τις εξής δομές για λήψης αποφάσεων, γνωστές από άλλες γλώσσες προγραμματισμού, αλλά με διαφορετική σύνταξη, Αυτές είναι το απλό if – then – else, η πολλαπλή επιλογή case , και βρόχοι με χρήση for, while και until.

Δομή ελέγχου if-then-else.

Ελέγχει αν ισχύει μία συνθήκη και ανάλογα τρέχει μία ομάδα από εντολές ή δεν τις τρέχει. Μπορεί διαδοχικά να κάνει έλεγχο πολλών συνθηκών και να τρέχει τις αντίστοιχες ομάδες εντολών. Αν μία συνθήκη είναι αληθής τότε, δεν ελέγχονται οι υπόλοιπες. Η γενική μορφή σύνταξης μίας τέτοιας δομής φαίνεται παρακάτω :

```
if condition1 ; then
   commands
elif condition2 ; then
   commands
elif condition3 ; then
   commands
      .....
else
   commands
fi
```

Το σκιασμένο τμήμα είναι προαιρετικό. Αν ισχύει η πρώτη συνθήκη *condition1* τότε εκτελούνται όλες οι εντολές μέχρι το πρώτο **elif**, αν υπάρχει ή μέχρι το **else** αν υπάρχει ή μέχρι το **fi** αν δεν υπάρχει τίποτα από αυτά. Αντίστοιχα ισχύουν και για τους υπόλοιπους κλάδους, δηλαδή εκτελούνται οι αντίστοιχες εντολές.
Συνθήκη είναι μία παράσταση που μπορεί να είναι αληθής ή ψευδής και μπορεί να πάρει διάφορες μορφές, όπως να είναι ένας αριθμητικός έλεγχος (μία μεταβλητή να είναι μεγαλύτερη από έναν ακέραιο), έλεγχος σε συμβολοσειρά (π.χ. μία μεταβλητή να έχει μηδενικό μήκος ή να είναι ίση με μία άλλη σταθερά). Επίσης μπορεί να είναι έλεγχος πάνω σε κάποιο αρχείο, (υπάρχει το αρχείο ή

δεν υπάρχει, έχει την τάδε εξουσιοδότηση κ.τ.λ.). Τέλος, σαν συνθήκη μπορεί να δοθεί μία εντολή και το αποτέλεσμα της εντολής να σηματοδοτήσει την ισχύ ή όχι της συνθήκης.
Εδώ πρέπει να αναφερθεί ότι όλες οι εντολές του Linux έχουν έναν κωδικό εξόδου που είναι ο ακέραιος 0 αν η εντολή εκτελέστηκε με επιτυχία και διαφορετικός από μηδέν αν για κάποιο λόγο δεν τερμάτισε επιτυχώς. Πολλές φορές ο κωδικός αποτυχίας έχει τιμή που διαφέρει ανάλογα με το λόγο της αποτυχίας. Ένα παράδειγμα εντολής που μπορεί να αποτύχει είναι η **rm** όταν δεν υπάρχουν οι σχετικές εξουσιοδοτήσεις για διαγραφή αρχείου.
Έτσι, η εντολή-συνθήκη θεωρείται αληθής αν η εντολή έχει κωδικό 0, δηλαδή εκτελέστηκε με επιτυχία. Προσοχή γιατί η κωδικοποίηση της αληθούς συνθήκες σε 0 είναι αντίθετη από αυτήν που συνηθίζεται σε άλλες γλώσσες όπως η C.
Η σύνταξη για τη συνθήκη είναι **[** *expression1 operator expression2* **]** (προσοχή στα κενά) ή **test** *expression1 operator expression2* όπου *expression1* και *expression2* είναι μία μεταβλητή ή σταθερά και *operator* είναι ένας τελεστής σύγκρισης/ελέγχου που μπορεί να αφορά αριθμό, συμβολοσειρά ή αρχείο. Υπάρχει η περίπτωση κάποιος τελεστής να είναι ενός μόνο ορίσματος, οπότε οι παραπάνω παραστάσεις γίνονται **[** *operator expression* **]** ή **test** *operator expression*. Οι πίνακες που ακολουθούν περιγράφουν μερικούς αριθμητικούς τελεστές, τελεστές συμβολοσειρών και τελεστές αρχείων. Επίσης δίνονται και κάποια παραδείγματα.

Αριθμητικοί τελεστές σύγκρισης

Τελεστής	Σημασία
-eq	=
-ne	≠
-ge	≥
-gt	>
-le	≤
-lt	<

Παραδείγματα αριθμητικών συνθηκών:
```
test "$1" -eq "$counts"
[ $files -lt 4 ]
test $n -ne $procs
```

Τελεστές σύγκρισης συμβολοσειρών

Τελεστής	Σημασία
-z	Μηδενικό μήκος
-n	Μη μηδενικό μήκος
=	Ίδια συμβολοσειρά
!=	Διαφορετική συμβολοσειρά
	Δεν είναι κενή

Παραδείγματα συνθηκών συμβολοσειρών:
```
test -z $input1
[ $filename = "password" ]
test $searchString
```

Τελεστές αρχείων

Τελεστής	Σημασία
-e	Υπάρχει
-r	Εξουσιοδότηση για ανάγνωση

-w	Εξουσιοδότηση για εγγραφή
-x	Εξουσιοδότηση για εκτέλεση
-f	Είναι κανονικό αρχείο
-d	Είναι κατάλογος
-h	Είναι σύνδεσμος
-nt	Είναι πιο καινούργιο από το άλλο
-ot	Είναι πιο παλιό από το άλλο

Παραδείγματα συνθηκών για αρχεία:

```
test -x script1
test -a $file
[ -s $outFile ]
[ $file -nt "/etc/passwd" ]
```

Στο τμήμα V.4 δώθηκε το παράδειγμα του προγράμματος με όνομα `cprecent` . Το πρόγραμμα αναμένει ο κατάλογος που δίνεται σαν το δεύτερο όρισμα κατά την κλήση να υπάρχει. Αν δεν υπάρχει τότε η εντολή **cp** θα αποτύχει με σχετικό μήνυμα λάθους. Αυτό μπορεί να αποφευχθεί αν γίνει έλεγχος της ύπαρξης του καταλόγου και εφ΄όσον δεν υπάρχει τότε, να δημιουργείται. Αυτό ακριβώς κάνει το πρόγραμμα που ακολουθεί. Παρατηρήστε ότι η συνθήκη για τον κλάδο **elif** είναι η ίδια εντολή **mkdir**. Έτσι, αν αυτή αποτύχει (αν για παράδειγμα η διαδρομή που δώθηκε είναι λάθος ή δεν υπάρχουν τα σχετικά δικαιώματα) τότε, δεν θα εκτελεστεί η **cp**.

Πρόγραμμα `cprecent2`

```
#!/bin/bash
recent=`ls -t $1 | head -1`
if [ -d $2 ] ; then
   cp $1/$recent $2
elif mkdir $2 ; then
   cp $1/$recent $2
fi
```

Λογικός συνδυασμός συνθηκών.

Πολλές φορές χρειάζεται μία συνθήκη που θα μπει σε μία σχετική εντολή ελέγχου όπως η **if** να είναι λογικός συνδυασμός δύο ή περισσότερων συνθηκών (να ισχύουν όλες οι συνθήκες, να ισχύει έστω και μία από όλες ή η άρνηση κάποιας συνθήκης ή ένας πιο πολύπλοκος συνδυασμός). Αυτό μπορεί να επιτευχθεί με τους παρακάτω τρόπους (δίνεται η περίπτωση για δύο μόνο συνθήκες ή μίας για την περίπτωση της άρνησης, η επέκταση είναι προφανής).

```
[ condition1 ] && [ condition2 ]      Λογικό «ΚΑΙ»
[ condition1 -a condition2 ]          Λογικό «ΚΑΙ»
[ condition1 ] || [ condition2 ]      Λογικό «Η»
[ condition1 -o condition2 ]          Λογικό «Η»
[ ! condition ]                       Λογικό «ΟΧΙ»
```

Επίσης, με δεδομένο ότι μία συνθήκη μπορεί να είναι και μία απλή εντολή ισχύουν και οι παρακάτω συνδυασμοί που έχουν μία ιδιαιτερότητα ως προς το ποια εντολή θα εκτελεστεί :

command1 **&&** *command2*

Η συνθήκη είναι αληθής (0) αν **και** οι δύο εντολές τερματίστηκαν με επιτυχία. Η δεύτερη εντολή θα τρέξει μόνο αν τερματιστεί με επιτυχία η πρώτη.

command1 **||** *command2*

Η συνθήκη είναι αληθής (0) αν **κάποια** από τίς δύο εντολές τερματίστηκε με επιτυχία. Η δεύτερη εντολή θα τρέξει μόνο αν **δεν** τερματιστεί με επιτυχία η πρώτη.

Δομή πολλαπλής επιλογής case.

Η δομή πολλαπλής επιλογής εκτελεί μία από πολλές ομάδες εντολών, ανάλογα με την τιμή που παίρνει μία παράσταση. Η παράσταση αυτή μπορεί να είναι μία μεταβλητή, μία αριθμητική παράσταση, μία υποκατάσταση εξόδου εντολής ή ένα όνομα αρχείου το οποίο μπορεί να δίνεται με ένα γενικευμένο τρόπο με χρήση wildcards. Η τιμή συγκρίνεται με μία λίστα τιμών που υπάρχει σε κάθε κλάδο της δομής πολλαπλής επιλογής και αν συμπίπτει με κάποια από αυτές τότε, εκτελούνται οι σχετικές εντολές. Σε αυτήν την περίπτωση δεν ελέγχονται οι επόμενοι κλάδοι όπως στη γλώσσα C, αλλά ο έλεγχος της ροής μεταφέρεται εκτός της δομής ελέγχου. Η γενική σύνταξη της δομής πολλαπλής επιλογής φαίνεται παρακάτω, με το σκιασμένο τμήμα να είναι προαιρετικό. Δηλαδή, είναι δυνατόν να υπάρχει μόνο ένας κλάδος κάτι που κάνει τη δομή ισοδύναμη με ένα απλό if-fi.

```
case expression in
pattern1 )
      commands ;;
pattern2 )
      commands ;;
...
patternN )
      commands ;;
esac
```

Στους κλάδους της επιλογής τα *pattern1*,*pattern2*,… μπορεί να είναι το καθένα μία σταθερά (συμβολοσειρά ή αριθμός), μία λίστα από σταθερές (χωρισμένες με το σύμβολο `|`), περιοχές σταθερών χαρακτήρων (σύμβολισμός `[-]` ή `[]`), και επί πλέον μπορεί να είναι συνδυασμός σταθερών με wildcards όπως το `?` και το `*` ή ακόμα και αριθμητική παράσταση. Έτσι, ο αντίστοιχος κλάδος default που έχει η γλώσσα C στο bash shell είναι `*)` γιατί ο αστερίσκος ταιριάζει με οτιδήποτε. Αυτός ο κλάδος όμως θα πρέπει να βρίσκεται τελευταίος γιατί αν δεν είναι τελευταίος τότε, επειδή αληθεύει οπωσδήποτε δε θα ελεγχθούν οι κλάδοι που τον ακολουθούν. Αυτό ισχύει και γενικότερα, δηλαδή ένας γενικός κλάδος πρέπει να ακολουθεί τους πιο ειδικούς.

Στο πιο κάτω παράδειγμα δείχνονται κάποιες από τις δυνατότητες που έχει η δομή case. Το πρόγραμμα `numcheck` αναμένει έναν χαρακτήρα από το πληκτρολόγιο και ύστερα τον συγκρίνει με κάθε κλάδο στη σειρά. Ο τελευταίος κλάδος είναι αληθής αν δοθούν οι αριθμοί 0,1 ή οποιοσδήποτε άλλος μονός χαρακτήρας ή ακόμα και μία λέξη.

Πρόγραμμα `numcheck`

```
#!/bin/bash
echo –n ”Hit a key + Enter”
read key
case $key in
[2-5] )
      echo ”Small” ;;
[67] )
      echo ”Medium” ;;
8|9 )
      echo ”Big” ;;
* )
      echo ”Other” ;;
esac
```

Έτσι, αν το πρόγραμμα γραφόταν όπως στο παρακάτω παράδειγμα numcheck_wrong το αποτέλεσμα δε θα ήταν το επιθυμητό γιατί δεν θα εξέταζε καθόλου την περίπτωση να έχουν δοθεί οι αριθμοί 8 ή 9.

Πρόγραμμα numcheck_wrong

```
#!/bin/bash
echo –n ”Hit a key + Enter”
read key
case $key in
[2-5] )
      echo ”Small” ;;
[67] )
      echo ”Medium” ;;
* )
      echo ”Other” ;;
8|9)
      echo ”Big” ;;
esac
```

Βρόχος for.

Ο βρόχος for στο φλοιό bash διαφέρει λίγο σε σχέση με τους συνηθισμένους βρόχους for που υποστηρίζουν οι πιο πολλές γλώσσες προγραμματισμού στο ότι η μεταβλητή που σαρώνει τιμές δε δέχεται μία περιοχή τιμών ορισμένη με τα όριά της (άνω και κάτω) αλλά παίρνει τις τιμές της από μία λίστα τιμών. Η γενική σύνταξη δίνεται παρακάτω :

```
for var in list
do
      commands
done
```

Σε αυτήν τη σύνταξη οι εντολές που βρίσκονται ανάμεσα στα **do** και **done** εκτελούνται διαδοχικά για κάθε τιμή που παίρνει η μεταβλητή *var* από τη λίστα *list*. Η λίστα τιμών *list* μπορεί να δοθεί ρητά σαν μία παράθεση τιμών (λέξεις, αριθμοί, ονόματα αρχείων), σαν η έξοδος μίας εντολής (δηλαδή στη θέση του *list* θα μπει ο συμβολισμός υποκατάσταση εντολής `` `command` ``, η λίστα των ορισμάτων που δόθηκαν κατά την κλήση του προγράμματος ($@) ή ακόμα και μία λίστα από ονόματα αρχείων. Η τελευταία περίπτωση αποτελεί και την πιο κοινή χρήση για το βρόχο for. Σε αυτή την περίπτωση ένας ή περισσότεροι μεταχαρακτήρες στη θέση της *list* αναλαμβάνουν τη σάρωση των αρχείων. Ορισμένα παραδείγματα θα αποσαφηνίσουν αυτές τις δυνατότητες.

Πρόγραμμα capfname

```
#!/bin/bash
for f in *
do
      mv $f `ls $f | tr [a-z] [A-Z]`
done
```

Στο πρόγραμμα capfname η μεταβλητή f παίρνει διαδοχικά σαν τιμές όλα τα ονόματα αρχείων (*) που βρίσκονται στον κατάλογο εργασίας (δηλαδή τον κατάλογο από όπου έχει κληθεί το capfname).
Για κάθε αρχείο εκτελείται η εντολή **mv $f `ls $f | tr [a-z] [A-Z]`**, δηλαδή αλλάζει το όνομα του κάθε αρχείου σε ένα ίδιο όνομα με όλους τους αλφαβητικούς χαρακτήρες να είναι πλέον κεφαλαίοι.
Αυτό το παράδειγμα δεν είναι η μοναδική περίπτωση σάρωσης λίστας ονομάτων αρχείων. Μπορεί στη θέση της λίστας να μπει οποιοσδήποτε συνδυασμός διαδρομής που έχει μέσα μεταχαρακτήρες.

Π.χ. η λίστα **for f in chap?/*.txt** περιλαμβάνει όλα τα αρχεία που καταλήγουν σε .txt και βρίσκονται μέσα στους υπόκαταλογους που το όνομά τους ξεκινάει από chap και έχει έναν ακόμα χαρακτήρα..

Πρόγραμμα total

```
#!/bin/bash
total=0
for i in $@
do
      total=$((total+i))
done
echo Sum is : $total
```

Το πρόγραμμα total αθροίζει όλα τα ορίσματα που δίνονται κατά την κλήση του (υποθέτοντας ότι είναι ακέραιοι αριθμοί. Η σάρωση των ορισμάτων θα μπορούσε να γίνει και έχοντας σαν κεφαλή του βρόχου το **for i** χωρίς λίστα. Σε μία τέτοια περίπτωση υπονοείται ότι η λίστα είναι η $@, δηλαδή όλα τα ορίσματα που δόθηκαν στο πρόγραμμα.

Πρόγραμμα chext

```
#!/bin/bash
for f in *
do
      mv $f ${f}.bak
done
```

Τέλος,, το πρόγραμμα chext μετονομάζει όλα τα αρχεία του τρέχοντος καταλόγου, σε αρχεία με ίδιο όνομα αλλά με κατάληξη .bak. Τώρα σα λίστα χρησιμοποιείται η λίστα με ονόματα αρχείων που παράγει ο μεταχαρακτήρας *.

Βρόχος while.

Η γενική σύνταξη του βρόχου while στο bash shell είναι :

```
while condition
do
      commands
done
```

Ο βρόχος είναι ενεργός, δηλαδή οι εντολές ανάμεσα στα **do** και **done** επαναλαμβάνονται, όσο η συνθήκη *condition* είναι αληθής. Σα συνθήκη μπαίνει οποιαδήποτε συνθήκη μπορεί να μπει και στη δομή if.
Το πιο κάτω παράδειγμα υπολογίζει τον μέγιστο από όλους τους αριθμούς που δίνονται σαν όρισμα κατά την κλήση. Στην αρχή η μεταβλητή max που παριστάνει τον μέγιστο τίθεται στην ελάχιστη δυνατή τιμή (-2^{31}) και ο μετρητής count των ορισμάτων στην τιμή 0. Ο βρόχος τρέχει όσο ο μετρητής δεν έχει φτάσει τον αριθμό των ορισμάτων που δόθηκαν. Μέσα στο βρόχο ο μετρητής επαυξάνεται και ο προσωρινός μέγιστος συγκρίνεται με το καινούργιο όρισμα που διαβάζεται.

Πρόγραμμα maxarg1

```
#!/bin/bash
max=-2147483648
count=0
Nargs=$#
while [ $count -lt $Nargs ]
do
   count=$((count+1))
```

```
   if [ $1 -gt $max ] ; then
      max=$1
   fi
  shift
done
echo $max
```

Βρόχος until.

Η γενική σύνταξη του βρόχου until στο bash shell είναι :

```
until condition
do
      commands
done
```

Ο βρόχος είναι ενεργός, δηλαδή οι εντολές ανάμεσα στα **do** και **done** επαναλαμβάνονται, όσο η συνθήκη *condition* **δεν** είναι αληθής. Δηλαδή η συνθήκη έχει την αντίθετη λειτουργία σε σχέση με το βρόχο while.
Έτσι, το πιο κάτω παράδειγμα maxarg2 κάνει ακριβώς ότι και το maxarg1, μόνο που έχει μπει **until** αντί **while** και η συνθήκη έχει αλλάξει έτσι ώστε οι δύο βρόχοι να είναι ισοδύναμοι.

Πρόγραμμα maxarg2

```
#!/bin/bash
max=-2147483648; count=0 ; Nargs=$#
until [ $count -eq $Nargs ]
do
   count=$((count+1))
   if [ $1 -gt $max ] ; then
      max=$1
   fi
   shift
done
echo $max
```

Απότομη έξοδος από βρόχους – **break** και **continue**.

Υπάρχουν δύο εντολές που διαταράσσουν την κανονική ροή των βρόχων for, while και until όταν εκτελεστούν μέσα στο σώμα τους. Η πρώτη είναι η **break** και η εκτέλεσή της έχει ως αποτέλεσμα τον τερματισμό του βρόχου. Η δεύτερη είναι η **continue** και κάνει τον βρόχο να αγνοήσει τις επόμενες εντολές (μέχρι το **done**) και να επιστρέψει πάλι στην αρχή (στο **do**) για ένα νέο κύκλο.
Γενικά, δε θεωρείται καλή τακτική προγραμματισμού η χρήση των **break** και **continue**, όμως υπάρχουν περιπτώσεις που αυτές η δύο εντολές είναι χρήσιμες. Όπως στο παρακάτω πρόγραμμα maxarg3 που γίνεται η εύρεση του μέγιστου αριθμού μέχρι τον 1000. Αν δεν είχε μπει το **break** θα γίνονταν άσκοπες πράξεις.

Πρόγραμμα maxarg3

```
#!/bin/bash
max=-2147483648; count=0 ; Nargs=$#
until [ $count -eq $Nargs ]
do
   count=$((count+1))
   if [ $1 -gt $max ] ; then
      max=$1
   fi
```

```
    if [ $max -ge 1000 ] ; then
       max=1000
       break
    fi
    shift
done
echo $max
```

7. Συναρτήσεις και άλλες δυνατότητες.

Συναρτήσεις.

Η συνάρτηση στον φλοιό bash shell είναι ένα κομμάτι κώδικα φλοιού, που μπορεί να επαναχρησιμοποιηθεί (κληθεί) πολλές φορές μέσα στο πρόγραμμα φλοιού. Ο ορισμός της συνάρτησης πρέπει να γίνεται πριν από την πρώτη κλήση του μέσα στο πρόγραμμα. Επίσης, μπορούν μέσα στη συνάρτηση να περαστούν παράμετροι (όπως και στις συναρτήσεις των άλλων γλωσσών προγραμματισμού). Τέλος, προαιρετικά η συνάρτηση μπορεί να επιστρέφει και ένα αποτέλεσμα (ακέραιο αριθμό) στο κυρίως πρόγραμμα. Η γενική μορφή για το ορισμό της συνάρτησης και την κλήση της φαίνεται πιο κάτω :

commands …	Διάφορες εντολές πριν τον ορισμό της Συνάρτησης
function *fname* **{** **local** *Lvars* *commands* **return** *n* **}**	Ορισμός της συνάρτησης με όνομα *fname* Ορισμός τοπικών μεταβλητών συνάρτησης Εντολές συνάρτησης Επιστροφή αποτελέσματος (προαιρετικό)
… *commands* *fname arg1 arg2* *commands*	 Άλλες εντολές εκτός συνάρτησης Κλήση συνάρτησης με παραμέτρους Υπόλοιπες εντολές συνάρτησης

Ο τρόπος που μπορεί να χρησιμοποιηθούν μέσα στο σώμα της συνάρτησης οι παράμετροι που δίνονται κατά την κλήση της είναι όπως και με τις παραμέτρους που δίνονται σε ένα πρόγραμμα φλοιού, δηλαδή `$1`,`$2`, κ.τ.λ. Επίσης, το σώμα της συνάρτησης μπορεί να έχει τοπικές μεταβλητές που είναι ορατές μόνο μέσα στο σώμα. Αυτές ορίζονται με τη λέξη `local`.
Η προαιρετική επιστροφή του αποτελέσματος μίας συνάρτησης γίνεται με τη δεσμευμένη λέξη **`return`**. Συνήθως το αποτέλεσμα επιστροφής έχει σχέση με την επιτυχία ή αποτυχία της συνάρτησης, δίνοντας ανάλογο κωδικό, έτσι ώστε αυτός να γίνει εκμεταλεύσιμος από το κυρίως πρόγραμμα σε λήψη αποφάσεων.
Κατά την κλήση μίας συνάρτησης από το κυρίως πρόγραμμα, αυτή έχει προτεραιότητα έναντι τυχόν εξωτερικών ή εσωτερικών εντολών με το ίδιο όνομα. Γενικά, η προτεραιότητα είναι alias, function, εσωτερική εντολή, εξωτερική εντολή. (Για alias δείτε το εγχειρίδιο του bash).
Η μεγάλη ισχύς των συναρτήσεων βρίσκεται στο γεγονός ότι μία συνάρτηση μπορεί να καλέσει μία άλλη, ακόμα και τον εαυτό της, κάνοντας έτσι δυνατό τον αναδρομικό προγραμματισμό. Ένα παράδειγμα είναι το πρόγραμμα `treer`.

Πρόγραμμα `treer`

```
#!/bin/bash
function traverse {
for file in *
do
      if [ -d $file ];
            then
            cd $file
            echo "DIR : $file"
```

```
            traverse
            cd ..
      else
            ls -l $file
      fi
done
}
traverse
```

Σε αυτό το πρόγραμμα η συνάρτηση `traverse` σαρώνει όλα τα αρχεία του τρέχοντος καταλόγου και τα εμφανίζει, εκτός αν συναντήσει αρχείο που είναι κατάλογος. Σε αυτή την περίπτωση μπαίνει μέσα στον κατάλογο, καλεί τον εαυτό της (προκειμένου να κάνει το ίδιο με πριν) και επιστρέφει στο γονικό κατάλογο. Το αποτέλεσμα είναι παρόμοιο με αυτό της αναδρομικής κλήσης **`ls -lR`**.

Κωδικός εξόδου προγράμματος.

Όπως όλα τα προγράμματα του Linux (και οι συναρτήσεις όπως φάνηκε παραπάνω) επιστρέφουν έναν κωδικό που δηλώνει επιτυχία (0) ή αποτυχία και πιθανώς την αιτία (≠0), έτσι και ένα πρόγραμμα φλοιού μπορεί να επιστρέψει το δικό του κωδικό. Αυτό γίνεται χρησιμοποιώντας την εντολή **`exit`** *`n`*, όπου *n* είναι ο κωδικός εξόδου.

Χειρισμός διακοπών - σημάτων.

Στην επόμενη ενότητα θα αναφερθεί η έννοια των σημάτων (signals) προς διεργασίες. Ένα πρόγραμμα φλοιού μπορεί να χειρίζεται τέτοια σήματα που λαμβάνει με την εντολή
`trap` *`'commands' signal`*
Μέσα σε απλά εισαγωγικά μπαίνουν οι εντολές που θα εκτελεστούν όταν παραληφθεί ένα σήμα με κωδικό signal. Ο κωδικός μπορεί να είναι ο αριθμητικός ή ο συμβολικός.
Το πιο κάτω παράδειγμα απενεργοποιεί τη δυνατότητα τερματισμού που υπάρχει όταν πατηθεί ο συνδυασμός πλήκτρων **`Ctrl-C`** και εμφανίζει ένα μήνυμα. Το πρόγραμμα είναι ουσιαστικά ένας ατέρμονος βρόχος για να δοθεί η ευκαιρία στο χρήστη να πατήσει το **`Ctrl-C`**.

Πρόγραμμα `trapexample`

```
#~/bin/bash
#trap 'echo Ctrl-C' 2
while true
do
   :
done
```

Συμπερίληψη πολλών αρχείων.

Μέσα σε ένα αρχείο προγράμματος φλοιού μπορούν να συμπεριληφθούν σε οποιοδήποτε σημείο του και άλλα αρχεία τα οποία μπορεί να περιέχουν εντολές οι συναρτήσεις. Αυτό γίνεται βάζοντας την εντολή **`source`** *`file`* ή με **`.`** *`file`* στο σημείο που θέλουμε να συμπεριληφθεί το αρχείο με όνομα *`file`*. Το αποτέλεσμα θα είναι σαν να υπάρχει ένα ενιαίο αρχείο. Συνήθως στα αρχεία που εισάγουμε υπάρχουν κάποιες συναρτήσεις.

Δυνατότητες εκσφαλμάτωσης (debugging).

Τέλος, δίνεται μία χρήσιμη δυνατότητα για την εύρεση λογικών λαθών κατά τη διάρκεια ανάπτυξης ενός προγράμματος. Κάθε εντολή που εκτελείται εμφανίζεται στην οθόνη προκειμένου να ξέρει ο προγραμματιστής σε πιο σημείο βρίσκεται κάθε στιγμή το πρόγραμμα. Αυτό ενεργοποιείται

εισάγοντας μέσα στο πρόγραμμα (στην αρχή) την εντολή **`set -x`**, ή καλώντας το με **`bash -x`** *`script`* όπου *`script`* είναι το όνομά του.

Ασκήσεις.

1. Δημιουργήστε ένα script με όνομα `lsd` που θα καλείται με σύνταξη :
   `lsd [dir]`
   Το πρόγραμμα αυτό θα εμφανίζει μόνο τους καταλόγους που βρίσκονται κάτω από τον κατάλογο ***dir*** ή κάτω από τον τρέχοντα κατάλογο σε περίπτωση που η παράμετρος ***dir*** δε δίνεται (οι αγκύλες δηλώνουν ότι η παράμετρος ***dir*** είναι προαιρετική). Εάν δίνεται η παράμετρος, αλλά δεν υπάρχει ο κατάλογος τότε να εμφανίζεται σχετικό μήνυμα λάθους.

2. Δημιουργήστε ένα script με όνομα `luname` το οποίο θα καλείται με σύνταξη:
   `luname {-l|-u} [dir]`
   Το πρόγραμμα φλοιού, θα μετονομάζει όλα τα αρχεία που βρίσκονται στον κατάλογο ***dir*** στα αντίστοιχα ονόματα με πεζά γράμματα (επιλογή **`-l`**) ή κεφαλαία γράμματα (επιλογή **`-u`**). Αν δε δίνεται ***dir*** θα θεωρείται ο τρέχων κατάλογος. Αν δίνεται, αλλά δεν υπάρχει θα εμφανίζεται μήνυμα λάθους. Το πρόγραμμα να σχεδιαστεί έτσι ώστε η διαδοχική του εκτέλεση με ίδια επιλογή στον ίδιο κατάλογο να μην εμφανίζει μηνύματα λάθους της εντολής **`mv`** . Η σύνταξη **`{-l|-u}`** σημαίνει ότι υποχρεωτικά μία από τις δύο επιλογές θα πρέπει να δίνεται, διαφορετικά να εμφανίζεται σχετικό μήνυμα λάθους.

3. Δημιουργήστε ένα script με όνομα `cprecent` το οποίο θα καλείται με σύνταξη:
   `cprecent [-a] [-n] [source_dir] dest_dir`
   Το πρόγραμμα φλοιού πρέπει να αντιγράφει τα πιο πρόσφατα ***n*** αρχεία του καταλόγου ***source_dir*** που έχουν δημιουργηθεί ή έχουν προσπελαστεί (**`-a`**), στον κατάλογο ***dest_dir***
   Οι παράμετροι μέσα σε αγκύλες **`[]`** είναι προαιρετικές, δηλαδή το script θα πρέπει να χειριστεί όλες τις περιπτώσεις είτε δίνονται κάποιες παράμετροι είτε λείπουν.
   Παράμετροι :
   `-a` : Αν δίνεται, τότε να αντιγραφούν τα αρχεία που προσπελάστηκαν πιο πρόσφατα, ενω αν δε δίνεται τότε θα πρέπει να αντιγραφούν τα αρχεία που δημιουργήθηκαν ή άλλαξαν πιο πρόσφατα.
   `-n` : Πόσα από τα πιο πρόσφατα αρχεία να αντιγραφούν (>0). Αν δε δίνεται αυτή η παράμετρος, τότε θεωρείται ότι έχει τιμή 1.
   source_dir : Αν δίνεται και υπάρχει είναι ο κατάλογος όπου θα γίνει η αναζήτηση των αρχείων προς αντιγραφή. Αν δεν υπάρχει ο κατάλογος που δίνεται να εμφανίζεται σχετικό μήνυμα λάθους. Αν δε δίνεται ο κατάλογος να θεωρείται ότι είναι ο τρέχων κατάλογος
   dest_dir : Αν δίνεται και υπάρχει είναι ο κατάλογος όπου θα γίνει η αντιγραφή αρχείων, αν δεν υπάρχει να δημιουργείται, ενώ αν δε δίνεται η παράμετρος να εμφανίζεται μήνυμα λάθους.
   Βοήθεια : Χρησιμοποιήστε και το χειρισμό strings που προσφέρει το bash

4. Δημιουργήστε ένα script με όνομα `ftree` το οποίο θα καλείται με σύνταξη:
   `ftree dir file`
   Το πρόγραμμα πρέπει να ψάχνει στον κατάλογο ***dir*** και όλους τους υποκαταλόγους του για αρχεία με όνομα ***file*** και όπου βρίσκει τέτοιο αρχείο να εμφανίζει το μονοπάτι στο οποίο βρέθηκε. Αν δεν υπάρχει ο κατάλογος να εμφανίζει μήνυμα λάθους. Αν δεν βρέθηκε αρχείο να εμφανίζει σχετικό μήνυμα. Να αναζητούνται και καταλόγοι εκτός από κανονικά αρχεία.

Παρατηρήσεις για όλες τις ασκήσεις:

1. Οι αγκύλες και τα άγκιστρα είναι συμβολισμός που δηλώνει πραιρετική παράμετρο και υποχρεωτική επιλογή ανάμεσα σε δύο περιπτώσεις παραμέτρων, αντίστοιχα. Δε θα πρέπει να δίνονται κατά την κλήση των scripts.
2. Χρησιμοποιήστε την εντολή **`shopt -s nullglob`** (τίθεται η επιλογή nullglob σε κατάσταση on). Αν δε γίνει αυτό η χρήση π.χ. της **`for f on *`** θα δώσει την τιμή **`*`** στη μεταβλητή **`f`** σε περίπτωση που ένας κατάλογος είναι κενός και επί πλέον ο βρόγχος θα εκτελεστεί μία φορά, κάτι που δεν είναι επιθυμητό. Αντίθετα, με την παραπάνω επιλογή η **`f`** θα παραμείνει κενή και ο βρόγχος δε θα εκτελεστεί.

Λειτουργικό Σύστημα Linux
Ενότητα VI
ΔΙΕΡΓΑΣΙΕΣ και ΜΝΗΜΗ

1. Εισαγωγή.

Η διεργασία (process) είναι ένα πρόγραμμα που εκτελείται από το λειτουργικό σύστημα, δηλαδή βρίσκεται στη μνήμη και απασχολεί κατά διαστήματα τον επεξεργαστή (ή τους επεξεργαστές) του υπολογιστή μέχρι να ολοκληρωθεί και επίσης, κατέχει ή συναγωνίζεται για πόρους του υπολογιστή. Η έννοια της διεργασίας είναι αρκετά διαφορετική από αυτήν του προγράμματος γιατί σε αυτήν (εκτός του στατικού δυαδικού κώδικα του προγράμματος) περιλαμβάνονται και δυναμικές πληροφορίες, δηλαδή πληροφορίες που αλλάζουν με το χρόνο, όσο η διεργασία προχωράει προς την ολοκλήρωσή της. Τέτοιες πληροφορίες που αφορούν την τρέχουσα κατάσταση της διεργασίας είναι π.χ. η τιμή του Μετρητή Προγράμματος (ποια εντολή θα ακολουθήσει), του Μετρητή Στοίβας, οι τιμές όλων των υπόλοιπων καταχωρητών, τα περιεχόμενα της μνήμης που έχει διατεθεί για την αποθήκευση των διαφόρων μεταβλητών, ο κάτοχος (user) της διεργασίας, ποια είναι τα ανοικτά αρχεία που διαβάζει ή γράφει η διεργασία, ποια κομμάτια του κώδικα βρίσκονται στη φυσική μνήμη και ποια στο σκληρό δίσκο λόγω σελιδοποίησης, κ.ο.κ. Αντίθετα, ένα πρόγραμμα είναι απλώς ο εκτελέσιμος δυαδικός κώδικας που βρίσκεται αποθηκευμένος σε ένα αρχείο. Έτσι, και επειδή το Linux μπορεί να τρέχει πολλές διεργασίες ταυτόχρονα, είναι δυνατόν το ίδιο ακριβώς πρόγραμμα να υπάρχει σε εκτέλεση την ίδια χρονική στιγμή σαν πολλές διεργασίες.
Η διαχείριση των διεργασιών σε ένα σύστημα Linux γίνεται από τον πυρήνα (kernel) του λειτουργικού συστήματος, δηλαδή το υποσύνολο του λειτουργικού που βρίσκεται πιο κοντά και σε άμεση συνεργασία με το υλικό. Ο πυρήνας εκτός από τη διαχείριση των διεργασιών εκτελεί και άλλες λειτουργίες μία από τις οποίες είναι και η διαχείριση μνήμης, δηλαδή η προστασία περιοχών μνήμης από μη εξουσιοδοτημένες προσπελάσεις, η δημιουργία εικονικής μνήμης με το σύστημα της σελιδοποίησης κ.α. Ο πυρήνας που είναι εγκατεστημένος σε ένα λειτουργικό σύστημα Unix δημιουργεί τις ουσιαστικές διαφορές ανάμεσα στις διάφορες παραλλαγές των Unix συστημάτων, αλλά και διαφορές ανάμεσα στα Linux συστήματα. Έτσι, είναι δυνατόν να υπάρχει διαφορετική έκδοση πυρήνα ακόμα και σε Linux της ίδιας διανομής. Στο Linux ο πυρήνας που χρησιμοποιείται φαίνεται με την εντολή **`uname -a`**.

2. Χρονοδρομολόγηση ή χρονοπρογραμματισμός.

Ο τρόπος που επιτυγχάνονται τα χαρακτηριστικά της πολυχρησίας, πολυπρογραμματισμού και πολυεπεξεργασίας σε ένα λειτουργικό σύστημα Linux είναι μέσω χρονοδρομολόγησης (scheduling) που επιτελεί ο πυρήνας, δηλαδή της διαδοχικής εκχώρησης του επεξεργαστή κατά μικρά χρονικά διαστήματα στις διάφορες διεργασίες. Έτσι, η κάθε μία διεργασία λαμβάνει ένα μικρό χρόνο εκτέλεσης στον επεξεργαστή, διακόπτεται, ο επεξεργαστής δίνεται στην επόμενη διεργασία κ.ο.κ. Το συνολικό αποτέλεσμα είναι η ψευδαίσθηση των χρηστών ότι το σύστημα τρέχει ταυτόχρονα όλες τις διεργασίες, κάτι φυσικά που δεν ισχύει γιατί μόνο μία διεργασία εκτελείται κάθε χρονική στιγμή από τον επεξεργαστή (εκτός και αν διατίθενται περισσότεροι επεξεργαστές ή επεξεργαστές με πολυνημάτωση).
Εκτός από τη διακοπή μίας διεργασίας για λόγους χρονοδρομολόγησης, αυτή μπορεί να διακοπεί και για άλλους λόγους, όπως όταν περιμένει είσοδο/έξοδο (από ένα τερματικό, από το σκληρό δίσκο κ.α.) ή περιμένει αποτελέσματα από κάποια άλλη διεργασία ή περιμένει να μπει σε κρίσιμο τμήμα του κώδικά της για χρήση ενός κοινόχρηστου πόρου που όμως τη στιγμή αυτή έχει δεσμευθεί από κάποια άλλη διεργασία. Έτσι, το Linux διαχωρίζει τις διεργασίες ως προς το θέμα της εκτέλεσής τους σε κατηγορίες, όπου οι πιο συνηθισμένες φαίνονται παρακάτω:

- Εκτελέσιμη ή έτοιμη (Runnable,Ready) – Πρόκειται για διεργασίες που είτε εκτελούνται από τον επεξεργαστή, είτε είναι έτοιμες προς εκτέλεση όταν τους διατεθεί χρόνος. Δηλαδή η εκτελέσιμη διεργασία είναι δυνατόν να μην εξυπηρετείται μία χρονική στιγμή από τον επεξεργαστή, αλλά είναι «εν δυνάμει» σε εκτέλεση.
- Σε ύπνωση (Sleeping) – Είναι οι διεργασίες που έχει διακοπεί η εκτέλεσή τους ή η δυνατότητα που έχουν να εκτελεστούν από τον επεξεργαστή, γιατί περιμένουν κάποιο συμβάν. Τέτοιο συμβάν μπορεί να είναι η προσπέλαση μίας συσκευής εισόδου/εξόδου η οποία δεν είναι έτοιμη ακόμα (π.χ. κάρτα δικτύου, σκληρός δίσκος), η αναμονή κάποιου σήματος διακοπής (π.χ. πληκτρολόγιο), ή η αναμονή κάποιου σήματος από μία άλλη διεργασία. Η κατάσταση της ύπνωσης στην ορολογία του Linux χωρίζεται στις εξής δύο υποπεριπτώσεις: με δυνατότητα διακοπής (Interruptible) ή με αδυναμία διακοπής (Uninterruptible). Η πιο συνηθισμένη κατάσταση είναι η πρώτη.
- Ακινητοποιημένη (Stopped) : Η διεργασία έχει μπει σε μία κατάσταση παγώματος εξ' αιτίας ειδικών σημάτων από άλλες διεργασίες ή από παρέμβαση του χρήστη.
- Ζόμπι (Zombie) : Είναι διεργασία που ουσιαστικά έχει τελειώσει το έργο της, αλλά δεν έχει φύγει από τη λίστα των διεργασιών γιατί ο η *διεργασία – γεννήτορας* (**parent process)** δεν την έχει διαγράψει ακόμα. (Στο μοντέλο του Unix όλες οι διεργασίες γεννούνται από κάποιες άλλες που λέγονται γεννήτορες, με αποτέλεσμα να δημιουργείται μία δενδρική ιεραρχία διεργασιών με ρίζα την αρχική διεργασία του Unix που λέγεται `init`).

Η χρονοδρομολόγηση των διεργασιών επιτελείται από ένα ειδικό τμήμα του πυρήνα που λέγεται *χρονοδρομολογητής* (**scheduler**). Ο χρονοδρομολογητής είναι αυτός που αποφασίζει ποια είναι η επόμενη διεργασία που θα ανατεθεί στον επεξεργαστή. Ο αλγόριθμος της χρονοδρομολόγησης είναι διαφορετικός στις λεπτομέρειες σε κάθε λειτουργικό σύστημα Unix και επίσης είναι διαφορετικός και στο ίδιο το Linux ανάλογα με την έκδοση του πυρήνα. Πάντως όλοι η αλγόριθμοι ανήκουν στην κατηγορία της *χρονοδρομολόγησης με προεκχώρηση* (**preemptive scheduling**). Η χρονοδρομολόγηση που θα περιγραφεί αφορά τον πυρήνα του Linux με έκδοση 2.6.18.

Οι βασικός στόχος της χρονοδρομολόγησης είναι να γίνεται βέλτιστη εκμετάλλευση του επεξεργαστή. Έτσι βασικές παράμετροι που θα πρέπει να ρυθμίζει ο χρονοδρομολογητής είναι πόσος χρόνος (timeslice) θα διατίθεται σε κάθε διεργασία όταν της ανατίθεται ο επεξεργαστής πριν την αποσύρει εκ νέου από αυτόν και πόσο, και πόσο συχνά θα εκχωρείται ο επεξεργαστής σε κάθε διεργασία. Και οι δύο παράμετροι στον χρονοδρομολογητή του πυρήνα 2.6.18 είναι δυναμικές.

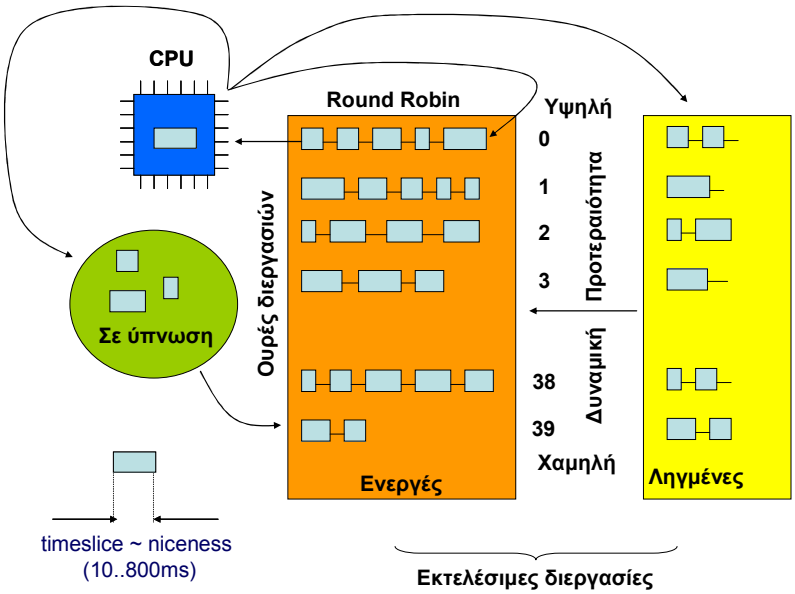
Οι διεργασίες μπορούν να κατηγοριοποιηθούν ως προς τον τύπο της εκτέλεσης σε διαβαθμίσεις ανάμεσα σε δύο άκρα :

- *Διεργασίες εξαρτημένες από τον επεξεργαστή* (**CPU-bound**) : Διεργασίες που ξοδεύουν το κύριο μέρος του χρόνου τους (πραγματικού χρόνου) εκτελώντας εντολές (π.χ. μία διεργασία που κάνει εντατικούς μαθηματικούς υπολογισμούς).
- *Διεργασίες εξαρτημένες από Είσοδο/Έξοδο* (**I/O-bound**) : Διεργασίες που ξοδεύουν το μεγαλύτερο ποσοστό του χρόνου τους περιμένοντας να παραλάβουν ή να στείλουν δεδομένα από/προς κάποια συσκευή εισόδου-εξόδου.

Η χρονοδρομολόγηση θα πρέπει να προσπαθήσει να συμβιβάσει δύο αντικρουόμενες απαιτήσεις : Μικρό χρόνο απόκρισης (**low latency**) σε όλες τις διεργασίες προκειμένου να δώσει στους χρήστες που κυρίως τρέχουν διαλογικές εργασίες την αίσθηση ότι το σύστημα είναι γρήγορο, και μεγάλη ικανότητα διεκπεραίωσης (**high throughput**), δηλαδή μία διεργασία που ξεκίνησε να τελειώσει σε εύλογο χρονικό διάστημα. Το Linux ευνοεί τις απαιτήσεις για μικρό χρόνο απόκρισης, δηλαδή ουσιαστικά ευνοεί τις διεργασίες που είναι διαλογικές (I/O bound) χωρίς όμως να αδικεί τις διεργασίες που είναι προσανατολισμένες σε υπολογισμούς (CPU-bound).

Ο τρόπος που λειτουργεί ο αλγόριθμος χρονοδρομολόγησης παρουσιάζεται παραστατικά στο επόμενο σχήμα όπου φαίνεται ότι χρησιμοποιεί πολλαπλές ουρές διαφορετικών προτεραιοτήτων με την κάθε μία να χρησιμοποιεί αλγόριθμο εξυπηρέτησης εκ περιτροπής (round robin). Οι ουρές έχουν όλες τις διεργασίες που βρίσκονται σε εκτελέσιμη κατάσταση (runnable). Η κάθε ουρά έχει προτεραιότητα από 0 (η πιο υψηλή) ως 39 (η πιο χαμηλή). Η προτεραιότητα αυτή είναι δυναμική και υπολογίζεται εκ νέου κάθε φορά που μία διεργασία βγαίνει από τις ουρές. Ο επεξεργαστής παραλαμβάνει διαδοχικά διεργασίες από την ουρά της πιο υψηλής προτεραιότητας (μπορεί να μην είναι απαραίτητα η 0 εάν αυτή έχει αδειάσει) και τις εκτελεί για ένα προκαθορισμένο κβάντο χρόνου (timeslice). Το κβάντο είναι διαφορετικό για κάθε διεργασία και εξαρτάται από την δυναμική

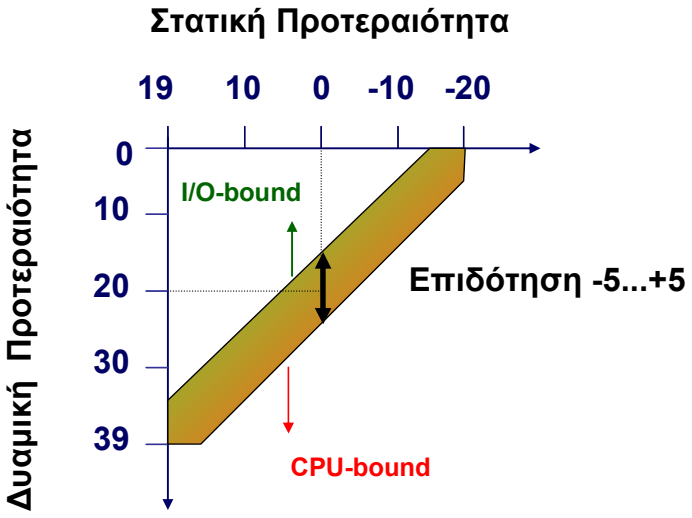
προτεραιότητα έχει η διεργασία πριν να μπει στις ουρές. Ο χρόνος που εκχωρείται παίρνει τιμές από 10ms ως 800ms. Μία διεργασία που βρίσκεται στον επεξεργαστή μπορεί να αποσυρθεί από αυτόν για τρεις λόγους :
1) Εξάντληση του κβάντου χρόνου. Τότε μεταφέρεται σε μία διαφορετική λίστα, εκτελέσιμων μεν διεργασιών, αλλά *ληγμένων* (**expired**).
2) Άφιξη νέας διεργασίας με μεγαλύτερη δυναμική προτεραιότητα.
3) Αλλαγή κατάστασης της διεργασίας από εκτελέσιμη σε αναστολή (π.χ. λόγω αναμονής για είσοδο ή έξοδο).



Ο υπολογισμός της δυναμικής προτεραιότητας γίνεται ως εξής : Αρχικά όλες οι διεργασίες όταν δημιουργούνται έχουν μία *στατική προτεραιότητα* («ευγένεια»-**niceness**) που παίρνει τιμές από -20 ως 19. Η στατική προτεραιότητα αν δεν καθοριστεί έχει αρχική τιμή 0. Μπορεί όμως να καθοριστεί ή να επανακαθοριστεί κατά τη διάρκεια της εκτέλεσης μίας διεργασίας με τις εντολές **`nice`** και **`renice`**. Η δυναμική προτεραιότητα που έχει κάθε διεργασία ανά πάσα στιγμή υπολογίζεται από τη σχέση :

Δ.Π. = Σ.Π. + Επιδότηση, με την Δ.Π. να είναι πάντα 0...39 (όπου Επιδότηση=-5...+5).

Τα κβάντα χρόνου υπολογίζονται εκ νέου για κάθε διεργασία όταν αυτή μεταπίπτει στη λίστα με τις ληγμένες διεργασίες. Ο τρόπος υπολογισμού είναι να δίνεται το μεγαλύτερο κβάντο (800ms) χρόνου στις διεργασίες με στατική προτεραιότητα -20 και το μικρότερο (10ms) στις διεργασίες με στατική προτεραιότητα 19. Οι ενδιάμεσες προτεραιότητες αποκτούν ενδιάμεσες τιμές κβάντων χρόνου με γραμμικό τρόπο όπως φαίνεται στην εικόνα που ακολουθεί.



Μία διεργασία που έχει μία δεδομένη στατική προτεραιότητα κινείται αναγκαστικά σε μία ζώνη δυναμικών προτεραιοτήτων πλάτους 11 όπως δείχνει και το παραπάνω σχήμα. Η επιδότηση ουσιαστικά είναι αυτή που μεταβάλει τη δυναμική προτεραιότητα, αφού η στατική γενικά μένει σταθερή. Η επιδότηση παίρνει τιμές από -5 ως +5 και έχει σκοπό να ευνοήσει τις διαλογικές διεργασίες, δηλαδή αυτές που έχουν σημαντική αλληλεπίδραση με είσοδο ή έξοδο. Η επιδότηση είναι -5 για τις διεργασίες που είναι καθαρά I/O-bound και +5 για τις CPU-bound. Η κατάταξη μίας διεργασίας για τον υπολογισμό της επιδότησης γίνεται μετρώντας το ποσοστό του χρόνου που βρίσκεται σε ύπνωση (δηλαδή περιμένει είσοδο-έξοδο) σε σχέση με το χρόνο που είναι εκτελέσιμη (δηλαδή βρίσκεται στις ουρές αναμονής ή στον επεξεργαστή).
Με αυτό τον τρόπο υπολογισμού επιτυγχάνονται τα εξής: Οι διαλογικές διεργασίες παίρνουν μεγαλύτερο κβάντο χρόνου και έτσι τους δίνεται μεγαλύτερο ποσοστό χρόνου για να ολοκληρωθούν βελτιώνοντας την αποκρισιμότητα του συστήματος. Από την άλλη, στις CPU-bound διεργασίες δίνεται δυναμικά λίγο πιο μικρή προτεραιότητα, μη επιτρέποντας έτσι την μονοπώληση του επεξεργαστή από αυτές, αφού σε περίπτωση που αυτός τους διατεθεί, μικρές πιθανότητες υπάρχουν να τον εγκαταλείψουν λόγω μετάπτωσης σε κατάσταση ύπνωσης.
Όταν όλες οι διεργασίες που είναι στη λίστα των *ενεργών* (**active**) διεργασιών έχουν εξαντλήσει το κβάντο χρόνου τους, και επομένως έχουν μεταφερθεί στη λίστα με τις ληγμένες (expired) τότε, γίνεται αμοιβαία εναλλαγή και οι ληγμένες διεργασίες γίνονται ενεργές. Τώρα οι προτεραιότητες είναι αλλαγμένες με βάση τον αλγόριθμο που περιγράφηκε πριν.

3. Εντολές ελέγχου διεργασιών.

Οι βασικές εντολές με τις οποίες μπορεί να γίνει παρατήρηση των διεργασιών και των χαρακτηριστικών τους που υπάρχουν σε ένα σύστημα (ανεξάρτητα από την κατάστασή τους) είναι η **ps** και η **top**. Η **ps** δίνει ένα στιγμιότυπο των διεργασιών που υπάρχουν στο σύστημα τη στιγμή που εκείνη εκτελείται, ενώ ή **top** μπορεί να δίνει περιοδικά επαναλαμβανόμενα στιγμιότυπα. Και οι δύο εντολές έχουν μία πληθώρα από παραμέτρους οι οποίες μπορεί να διαφοροποιούνται ανάλογα με την έκδοση του Unix. Διαφοροποίηση μπορεί να υπάρχει και στη μορφοποίηση των πληροφοριών που βγαίνουν στην έξοδο.
Ακολουθεί ένα απόσπασμα εξόδου της εντολής **ps -el** (με την επιλογή **-e** εμφανίζονται όλες οι διεργασίες και με την **-l** εμφανίζονται πολλές λεπτομέρειες για την κάθε μία – **l**ong format).

```
$ ps -el
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  0  75   0 -   487 -      ?        00:00:00 init
1 S     0     2     1  0 -40   - -     0 migrat ?        00:00:00 migration/0
1 S     0     3     1  0  94  19 -     0 ksofti ?        00:00:00 ksoftirqd/0
5 S     0     4     1  0  70  -5 -     0 worker ?        00:00:00 events/0
1 S     0     5     1  0  73  -5 -     0 worker ?        00:00:00 khelper
1 S     0     6     1  0  71  -5 -     0 worker ?        00:00:00 kthread
1 S     0     9     6  0  70  -5 -     0 worker ?        00:00:00 kblockd/0
...
5 S    33  7077  6408  0  85  10 -  5270 374284 ?        00:00:00 apache2
4 S     0  7600  2822  0  76   0 -  1928 -      ?        00:00:00 sshd
5 S  1000  7602  7600  0  75   0 -  1928 429496 ?        00:00:00 sshd
0 S  1000  7603  7602  0  75   0 -  1350 wait   pts/0    00:00:00 bash
0 R  1000  7737  7603  0  77   0 -   805 -      pts/0    00:00:00 ps
```

Το παραπάνω στιγμιότυπο αντικατοπτρίζει την κατάσταση των διεργασιών τη χρονική στιγμή που έτρεξε η εντολή **ps**. Κάθε γραμμή δίνει πληροφορίες για μία ξεχωριστή διεργασία. Οι πληροφορίες χωρίζονται σε στήλες και μπορούν να επιλέγονται μέσα από επιλογές της εντολής **ps** διαφορετικές πληροφορίες για επίδειξη κάθε φορά. Η **ps** έχει μία πληθώρα από επιλογές (options) που μεταξύ άλλων αφορούν ποιες από όλες τις διεργασίες θα εμφανιστούν, ποια χαρακτηριστικά θα εμφανίζονται για κάθε διεργασία κ.ά.
Ακολουθεί η περιγραφή μερικών από τις στήλες που εμφανίζονται παραπάνω:
Στήλη S: Η κατάσταση εκτέλεσης της διεργασίας. Οι δυνατές τιμές είναι R (**R**unnable) για εκτελέσιμη, S (**S**leeping interruptible) όταν βρίσκεται σε ύπνωση με δυνατότητα διακοπής, D (sleeping uniterruptible) όταν βρίκεται σε ύπνωση χωρίς δυνατότητα διακοπής, T (s**T**opped) για ακινητοποιημένη διεργασία ή διεργασία στην οποία γίνεται εκσφαλμάτωση, Z (**Z**ombie).

Στήλη UID: Ο αριθμός ταυτότητας του χρήστη (**U**ser **ID**entifier) στον οποίο ανήκει η διεργασία.
Στήλη PID: Ο αριθμός ταυτότητας της διεργασίας (**P**rocess **ID**entifier).
Στήλη PPID: Ο αριθμός ταυτότητας της διεργασίας που γέννησε την συγκεκριμένη διεργασία (**P**arent **P**rocess **ID**entifier)..
Στήλη PRI: Η δυναμική προτεραιότητα (**PRI**ority) που έχει η διεργασία με ένα offset 60. Οι δυναμικές προτεραιότητες που αναφέρει η **ps** παίρνουν τιμές από -40 εώς 99. Οι τιμές από -40 εώς και 59 αντιστοιχούν σε διεργασίες πραγματικού χρόνου (real time processes) που χρονοδρομολογούνται με διαφορετικό τρόπο από αυτόν που έχει περιγραφεί. Οι κανονικές διεργασίες έχουν δυναμική προτεραιότητα από 60 ως 99 με την διεργασία 60 να είναι υψηλότερης προτεραιότητας. Δηλαδή η προτεραιότητα PRI της **ps** με τιμή 60 αντιστοιχεί σε δυναμική προτεραιότητα 0 και η προτεραιότητα PRI με τιμή 99 αντιστοιχεί σε δυναμική προτεραιότητα 39.
Στήλη NI : Η «ευγένεια» (**NI**ceness) που έχει η διεργασία.
Στήλη TTY : Ο κωδικός τερματικού από το οποίο έχει ξεκινήσει η διεργασία ή από το οποίο μπορεί να γίνει ο έλεγχός της. Για παράδειγμα μπορεί να διακοπεί από το συγκεκριμένο τερματικό με το συνδυασμό Ctrl-C.
Στήλη TIME : Ο συνολικός χρόνος που έχει απασχολήσει η συγκεκριμένη διεργασία την μονάδα επεξεργασίας (CPU time). Ο χρόνος αυτός μπορεί να διαφέρει σημαντικά από τον πραγματικό χρόνο που είναι ενεργή αυτή η διεργασία. Π.χ. αν η διεργασία περνά το μεγαλύτερο μέρος του χρόνου της σε αναμονή εισόδου-εξόδου, ο χρόνος CPU θα είναι ελάχιστος.
Στήλη CMD : Το όνομα του εκτελέσιμου προγράμματος που αντιστοιχεί σε αυτή τη διεργασία.
Η εντολή **top** από την άλλη, εκτός από τις πληροφορίες που δίνει για κάθε μία διεργασία, δίνει και συνοπτικές πληροφορίες για το σύστημα όπως πόσες διεργασίες τρέχουν, πόση μνήμη υπάρχει, πόση από τη μνήμη αυτή χρησιμοποιείται κ.α. Εκτός από αυτό η top εμφανίζει περιοδικά (σε χρόνο που μπορεί να ρυθμιστεί με την επιλογή **-d** *secs*) τις διεργασιες που απασχολούν πιο πολύ τον επεξεργαστή σε φθίνουσα σειρά απασχόλησης. Η ταξινόμηση των διεργασιών είναι δυνατόν να γίνει και με άλλα κριτήρια.
Μία τυπική έξοδος της εντολής **top** φαίνεται παρακάτω :

```
$ top
top - 22:11:28 up 24 days,  4:21,  1 user,  load average: 1.83, 1.42, 1.31
Tasks:  96 total,   5 running,  91 sleeping,   0 stopped,   0 zombie
Cpu(s): 66.4%us,  0.3%sy, 33.2%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    516872k total,   508312k used,     8560k free,    99388k buffers
Swap:  1510068k total,        4k used,  1510064k free,   284204k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
10736 adp       25   0  1420  244  192 R 66.6  0.0   0:36.72 loop2
10563 adp       35  10  1424  248  192 R 33.3  0.0 101:04.58 loop1
    1 root      15   0  1948  648  552 S  0.0  0.1   0:00.93 init
    2 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    3 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    4 root      10  -5     0    0    0 S  0.0  0.0   0:00.01 events/0
    5 root      13  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
    6 root      11  -5     0    0    0 S  0.0  0.0   0:00.00 kthread
    9 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kblockd/0
...
```

Πολλές από τις στήλες των ιδιοτήτων είναι παρόμοιες με αυτές της εντολής **ps**. Π.χ οι στήλες PID,USER,PR,NI,TIME,COMMAND αντιστοιχούν στις στήλες PID,UID,PRI,NI,TIME,CMD της **ps**. Υπάρχουν όμως και κάποιες διαφοροποιήσεις : Π.χ. η στήλη που δείχνει τη δυναμική προτεραιότητα δείχνει την πραγματική δυναμική προτεραιότητα όπως την έχει ορισμένη ο χρονοδρομολογητής και οι τιμές βρίσκονται στην περιοχή από 0..39 σε αντίθεση με την **ps** που δίνει τιμές 60..99. Οι διεργασίες που χρονοδρομολογούνται σαν real time σημειώνονται με το σύβολο RT. Έτσι ισχύει για τις συνηθισμένες διεργασίες ότι $PR_{top}=PRI_{ps}-60$.
Η στήλη %CPU δείχνει το ποσοστό του χρόνου που αφιέρωσε ο επεξεργαστής για τη δεδομένη διεργασία. Επομένως, αν μία διεργασία ανάμεσα δύο διαδοχικές εμφανίσεις έχει χρησιμοποιήσει συνολικό χρόνο επεξεργασίας ptime secs (διαφορά δύο διαδοχικών τιμών TIME+) και η ανανέωση της top συμβαίνει κάθε rtime secs, το αντίστοιχο ποσοστό %CPU θα είναι ptime/rtime. Φυσικά το

άθροισμα των τιμών %CPU για όλες τις διεργασίες δε θα πρέπει να υπερβαίνει το 100%. Σε περίπτωση όμως που υπάρχουν περισσότεροι επεξεργαστές στο σύστημα το άθροισμα είναι δυνατόν να πλησιάζει το n·100%, όπου n ο αριθμός των επεξεργαστών. Π.χ. μπορεί να υπάρχουν δύο διεργασίες CPU-bound και η κάθε μία να έχει κατάληψη 100% για κάθε έναν επεξεργαστή, αφού ο χρονοδρομολογητής θα φροντίσει να αφιερώσει ξεχωριστό επεξεργαστή σε κάθε επεξεργασία.
Πριν την ανάλυση των χαρακτηριστικών των διεργασιών υπάρχει μία επικεφαλίδα με γραμμές που δείχνουν διάφορα χαρακτηριστικά του συστήματος. Ακολουθεί μία σύντομη περιγραφή μερικών από αυτά :
Στην πρώτη γραμμή φαίνονται ο πραγματικός χρόνος που έχει περάσει από την τελευταία επανεκκίνηση, ο αριθμός των χρηστών, ο μέσος φόρτος του συστήματος (load average) κατά σειρά το τελευταίο λεπτό, τα τελευταία 5 και τελευταία 15 λεπτά. Ο φόρτος του συστήματος είναι ο μέσος όρος των εκτελέσιμων διεργασιών (δηλαδή αυτών που είναι σε κατάσταση R) προς τον αριθμό των επεξεργαστών. Ουσιαστικά δείχνει πόσο επαρκεί η ισχύς του επεξεργαστή για τις συγκεκριμένες συνθήκες. Έτσι αν υπάρχει ένας μόνο επεξεργαστής και ο φόρτος είναι πάνω από 1 τότε, υπάρχει ανάγκη για περισσότερους επεξεργαστές.
Έχει σημασία ότι ο φόρτος εξαρτάται και από τον τύπο των δειργασιών. Έτσι ενώ μία CPU-bound διεργασία θα έφτανε εύκολα το φόρτο σε τιμή 1, μία I/O bound διεργασία έχει πολύ μικρή επίπτωση στο φόρτο, γιατί το μεγαλύτερο ποσοστό του χρόνου βρίσκεται σε κατάσταση S, επομένως δεν υπολογίζεται.
Στη δεύτερη γραμμή αναλύεται η κατανομή όλων των διεργασιών στις καταστάσεις εκτέλεσης που βρίσκονται (running,sleeping,stopped,zombie). Τονίζεται και πάλι σε αυτό το σημείο ότι μία διεργασία σε κατάσταση running δε σημαίνει απαραίτητα ότι βρίσκεται στον επεξεργαστή.
Η τρίτη γραμμή φανερώνει την ποσοστιαία κατανομή του χρόνου του επεξεργαστή σε διάφορες κατηγορίες εργασιών όπως : διεργασίες χρηστών στις οποίες δεν έχει αυξηθεί η εξ΄ορισμού τιμή niceness (%us), διεργασίες πυρήνα και συστήματος (%sy), διεργασίες χρηστών στις οποίες έχει αυξηθεί η τιμή niceness, ποσοστό χρόνου που δεν χρησιμοποείται για καμμία διεργασία (%id), ποσοστό χρόνου που καταναλώνεται σε αναμονή εισόδου – εξόδου (%wa), εξυπηρέτηση διακοπών υλικού (%hi), εξυπηρέτηση διακοπών λογισμικού (%si).
Οι επόμενες δύο γραμμές αφορούν χαρακτηριστικά της μνήμης του συστήματος και θα αναλυθούν στη σχετική ενότητα.
Η εντολή **top** εκτός από την πληθώρα επιλογών έχει και τη δυνατότητα για τη διαλογική (interactive) αλλαγή του τρόπου λειτουργίας της ενόσω τρέχει με τη χρήση συγκεκριμένων πλήκτρων. Π.χ. το πλήκτρο 1 αλλάζει την εμφάνιση της γραμμής CPU στην επικεφαλίδα από σωρρευτική απεικόνιση σε αναλυτική για κάθε επεξεργαστή ξεχωριστά (εφ'όσον υπάρχουν πάνω από ένας). Επίσης, μπορούν να επιλέγονται επί πλέον πεδία προς εμφάνιση ή αλλαγή της σειράς τους. Το πλήκτρο ? δίνει πληροφορίες για όλες αυτές τις επιλογές.
Πολλές φορές είναι επιθυμητή η προώθηση ορισμένων διεργασιών πιο ψηλά ή πιο χαμηλά στις προτεραιότητες, ιδιαίτερα όταν ένα σύστημα είναι φορτωμένο, προκειμένου να τερματιστούν πιο γρήγορα κάποιες από αυτές. Οι εντολές που χρησιμοποιούνται για μία τέτοια αλλαγή είναι οι **nice** και **renice**. Αυτές αλλάζουν ουσιαστικά την τιμή της στατικής προτεραιότητας niceness η οποία επηρρεάζει και το κβάντο χρόνου της διεργασίας.
Η εντολή nice χρησιμοποιείται όταν ξεκινάει μία διεργασία και έχει τη σύνταξη:
nice -n *value cmd*
όπου *value* είναι η τιμή της στατικής προτεραιότητας που θα πάρει το εκτελέσιμο πρόγραμμα *cmd*. Ένας απλός χρήστης μπορεί να δώσει τιμές προτεραιότητας από 0 ως 19, δηλαδή στην πραγματικότητα μπορεί μόνο να «χειροτερεύσει» τη θέση του έναντι άλλων χρηστών στο σύστημα αφού η εξ' ορισμού τιμή αν δε χρησιμοποιηθεί η **nice** είναι το 0. Από εκεί προέρχεται και το όνομα της εντολής nice, δηλαδή ευγενικός. Αντίθετα ο διαχειριστής (root) μπορεί να δώσει όλες τις τιμές από -20 ως 19 και έτσι να δώσει αυξημένη προτεραιότητα σε μία διεργασία.
Η εντολή renice έχει σκοπό την αλλαγή προτεραιότητας μίας διεργασίας ενώ αυτή τρέχει ήδη στο σύστημα. Η σύνταξή της είναι :
renice *value* **-p** *pid*
Όπως και πριν *value* είναι η τιμή της νέας στατικής προτεραιότητας και οι τιμές που μπορεί να πάρει είναι 0..19 για απλό χρήστη και -20..19 για τον διαχειριστή. Η διαφορά με την **nice** είναι ότι

η διεργασία πρέπει να αναφερθεί με τον κωδικό της (pid) ο οποίος μπορεί να βρεθεί με τις εντολές **ps** και **top**.

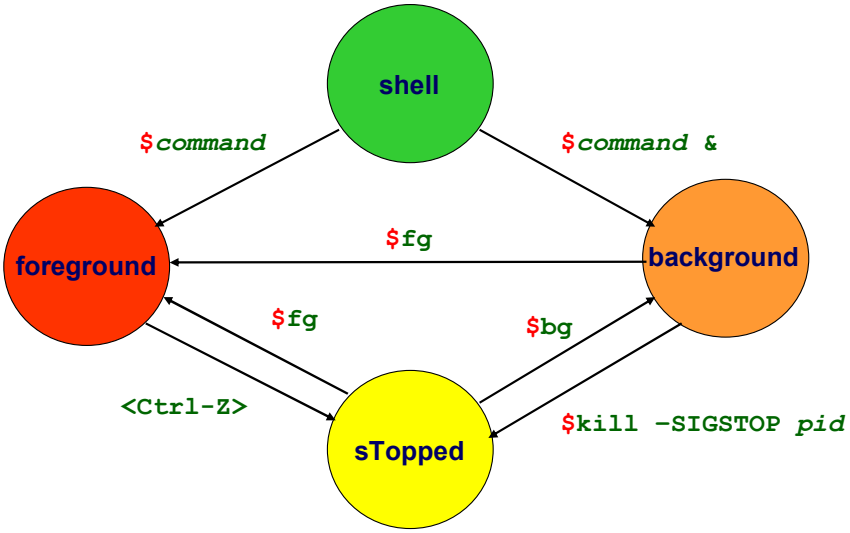
4. Προγράμματα στο προσκήνιο και παρασκήνιο.

Όταν ξεκινάει ένα πρόγραμμα από το φλοιό, ο φλοιός παραμένει δεσμευμένος μέχρι την ολοκλήρωσή του. Δηλαδή δεν αποδίδει στο χρήστη το σύμβολο προτροπής για να εισαχθεί νέα εντολή. Κατά τη διάρκεια της δέσμευσης αυτής ο φλοιός χρησιμοποιείται για είσοδο και έξοδο αποτελεσμάτων. Εάν διακοπεί ο φλοιός (π.χ. διακοπεί η απομακρυσμένη σύνδεση με την οποία έχει ανοίξει ο φλοιός) τότε, διακόπτεται και το πρόγραμμα. Όταν ένα πρόγραμμα εκτελείται με αυτό τον τρόπο λέγεται ότι τρέχει στο *προσκήνιο* (**foreground**).
Υπάρχουν περιπτώσεις όμως που είναι επιθυμητό το πρόγραμμα που ενεργοποιείται από το φλοιό να είναι ανεξάρτητο από αυτόν, δηλαδή να απελευθερώνει τον φλοιό για νέες εντολές ή και να συνεχίσει να τρέχει ακόμα και αν φλοιός τερματιστεί. Μία περίπτωση είναι όταν υπάρχει απομακρυσμένη σύνδεση σε έναν υπολογιστή και χρειάζεται να ξεκινήσει ένα πρόγραμμα που είναι χρονοβόρο και η σύνδεση πρέπει να διακοπεί σύντομα. Μεγαλύτερη ανάγκη υπήρχε στο παρελθόν όπου η πρόσβαση σε υπολογιστές γινόταν με μη γραφικά τερματικά και ο χρήστης είχε στη διάθεσή του μόνο έναν φλοιό. Όταν ένα πρόγραμμα εκτελείται με αυτό τον τρόπο λέγεται ότι τρέχει στο *παρασκήνιο* (**background**).
Η εκτέλεση ενός προγράμματος στο παρασκήνιο επιτυγχάνεται με την προσθήκη του συμβόλου & (ampersand) μετά το τέλος της εντολής, όπως παρακάτω:

```
$ ./loop1&
[1] 15137
$
```

Ο φλοιός στέλνει το πρόγραμμα στο παρασκήνιο, αναφέρει τον αριθμό της *εργασίας* (**task**) του τρέχοντος φλοιού (`[1]`), τον αριθμό της διεργασίας (`15137`) και ύστερα δίνει το σύμβολο της προτροπής έτοιμος να εκτελέσει την επόμενη εντολή. Ο αριθμός εργασίας αφορά τον τρέχοντα φλοιό μόνο και χρησιμοποιείται για το χειρισμό της συγκεκριμένης εργασίας όπως θα φανεί παρακάτω. Αν ένας άλλος φλοιός είναι ήδη ανοικτός θα κάνει ανεξάρτητη αρίθμηση των εργασιών ξεκινώντας και αυτός από τον αριθμό 1. Αντίθετα, ο αριθμός της διεργασίας είναι μοναδικός μέσα στο σύστημα.
Το πρόγραμμα τώρα θα τρέχει μέχρι να ολοκληρωθεί. Για το προηγούμενο παράδειγμα η ειδοποίηση τερματισμού θα έρθει από τον φλοιό με το μήνυμα `[1]+ Done loop1` όπου στις αγκύλες φαίνεται ο αριθμός της εργασίας παρασκηνίου που τερματίστηκε.
Αν κατά τη διάρκεια της εκτέλεσης παρουσιαστεί η ανάγκη για έξοδο στο τερματικό, αυτή θα γίνει κανονικά χωρίς να επηρρεαστούν τυχόν άλλες λειτουργίες που θα εκτελεί εκείνη τη στιγμή ο φλοιός. Προϋπόθεση για να συμβεί αυτό είναι να έχει δοθεί η ρύθμιση τερματικού **stty -tostop** αν δεν έχει ήδη δοθεί κατά την αρχικοποίηση του φλοιού. Ο έλεγχος της ρύθμισης μπορεί να γίνει με το συνδυασμό **sty -a | grep tostop**. Αντίθετα, αν είναι επιθυμητή η διακοπή του προγράμματος παρασκηνίου σε περίπτωση εξόδου στο τερματικό, η ρύθμιση θα πρέπει να είναι **stty tostop**. Φυσικά, ο χρήστης μπορεί με τους γνωστούς τρόπους ανακατεύθυνσης να φροντίσει ώστε η καθιερωμένη έξοδος να περνάει σε ένα αρχείο ή σε μία άλλη εντολή.
Σε περίπτωση που παρουσιαστεί η ανάγκη για είσοδο δεδομένων σε μία εργασία παρασκηνίου από το τερματικό τότε, αυτή θα τερματιστεί οπωσδήποτε εμφανίζοντας μήνυμα. Στο συγκεκριμένο παράδειγμα το μήνυμα θα ήταν `[1]+ Stopped loop1`. Η εργασία σε αυτήν την περίπτωση δεν αποσύρεται από τη μνήμη αλλά μένει σε κατάσταση ακινητοποίησης (T - sTopped). Μία ακινητοποιημένη εργασία μπορεί να επανενεργοποιηθεί, είτε στο προσκήνιο, είτε στο παρασκήνιο. Αυτό μπορεί να γίνει με τις εντολές ελέγχου εργασιών **fg** και **bg** αντίστοιχα, δηλαδή foreground και background. Επίσης η εντολή **fg** μπορεί να χρησιμοποιηθεί για να επαναφέρει μία εργασία από το παρασκήνιο στο προσκήνιο.
Το ακόλουθο σχήμα δείχνει όλες τις πιθανές μεταβάσεις μίας εργασίας, στις διάφορες καταστάσεις που μπορεί να έχει σε σχέση με το φλοιό. Στο σχήμα υπάρχουν και δύο ακόμα μεταβάσεις που θα περιγραφούν. Η πρώτη είναι η επαναφορά μίας εργασίας από το παρασκήνιο στο προσκήνιο με την

εντολή **fg**. Οι επόμενες μεταβάσεις είναι αυτές από το προσκήνιο ή το παρασκήνιο σε κατάσταση ακινητοποίησης. Αν μία εργασία τρέχει στο προσκήνιο μπορεί να ακινητοποιηθεί εφαρμόζοντας στο φλοιό το συνδυασμό πλήκτρων `Ctrl-Z`. Ο πλήρης τερματισμός εργασίας προσκηνίου γίνεται με το συνδυασμό `Ctrl-C`. Μία εργασία παρασκηνίου θα έχει οπωσδήποτε έναν αριθμό διεργασίας (pid). Η μετάπτωσή μίας τέτοιας διεργασίας σε κατάσταση ακινητοποίησης επιτυγχάνεται με την εντολή **kill -SIGSTOP *pid*** ή **kill -19 pid** που είναι ακριβώς το ίδιο. Η εντολή **kill** στέλνει διακοπές λογισμικού σε διεργασίες με διάφορους κωδικούς που αντιστοιχούνται σε συμβολικές λέξεις, όπως ο κωδικός 19 αντιστοιχεί για το Linux στη λέξη `SIGSTOP`. Μία άλλη διακοπή είναι αυτή με τον κωδικό 9 (`KILL`) όπου ακυρώνει εντελώς τη διεργασία-στόχο, δηλαδή είναι το αντίστοιχο σήμα που παράγει ο συνδυασμός πλήκτρων `Ctrl-C`.



Υπάρχουν διάφορες περιπτώσεις όπου θα μπορούσαν να χρησιμοποιηθούν αυτές οι μεταβάσεις. Μία υποθετική περίπτωση θα ήταν να έτρεχαν σε ένα σύστημα μία ή παραπάνω διεργασίες προσομοίωσης όπου θα ήταν γνωστό ότι θα διαρκούσαν πολύ ώρα. Κάπου στη μέση της προσομοίωσης μπορεί ο χρήστης να τρέξει μία επείγουσα εργασία σε σύντομο χρονικό διάστημα, αλλά αυτό να μην είναι δυνατό εξ΄αιτίας του μεγάλου φόρτου. Αυτό μπορεί να γίνει θέτοντας τις βαριές διεργασίες της προσομοίωσης σε κατάσταση ακινητοποίησης προκειμένου να απελευθερωθεί ο επεξεργαστής, κατόπιν να τρέξει τις σύντομες διεργασίες και ύστερα να επαναφέρει τις προσομοιώσεις (είτε στο προσκήνιο, είτε στο παρασκήνιο) και αυτές να συνεχίσουν από το σημείο όπου είχαν διακοπεί.

Είναι δυνατόν από ένα κέλυφος να έχουν ξεκινήσει πάνω από μία εργασίες που πιθανόν τρέχουν ή βρίσκονται ακινητοποιημένες στο παρασκήνιο. Για την καλύτερη εποπτεία τους υπάρχει η εντολή **jobs** που απαριθμεί τις εργασίες, με τη σειρά που ενεργοποιήθηκαν δείχνοντας το όνομά τους και την κατάστασή τους. Μία πιθανή έξοδος φαίνεται στο παράδειγμα που ακολουθεί.

```
$ jobs
[1]   Running                 ./loop1 &
[2]+  Stopped                 ./loop1
[3]-  Running                 ./loop1 &
```

Οι αριθμοί μέσα στις αγκύλες είναι ο αύξων αριθμός της εργασίας. Σε κάθε γραμμή επί πλέον φαίνεται η κατάσταση και το όνομα της εργασίας. Η δεύτερη εργασία φαίνεται ότι έχει ακινητοποιηθεί (ίσως με kill –SIGSTOP, ίσως με Ctrl-Z, ίσως γιατί έτρεχε στο παρασκήνιο και ζητήθηκε είσοδος από το τερματικό). Το σύμβολο + δείχνει την εργασία που εξ΄ορισμού θα επανέλθει στο προσκήνιο σε περίπτωση εντολής fg χωρίς όρισμα.

Ο αύξοντας αριθμός μίας εργασίας μπορεί να χρησιμοποιηθεί σαν όρισμα σε μία εντολή **fg** προκειμένου να επαναφερθεί στο προσκήνιο. Ο πίνακας δείχνει αυτή τη μορφή σύνταξης καθώς και άλλες παρεμφερείς περιπτώσεις με άλλα ορίσματα.

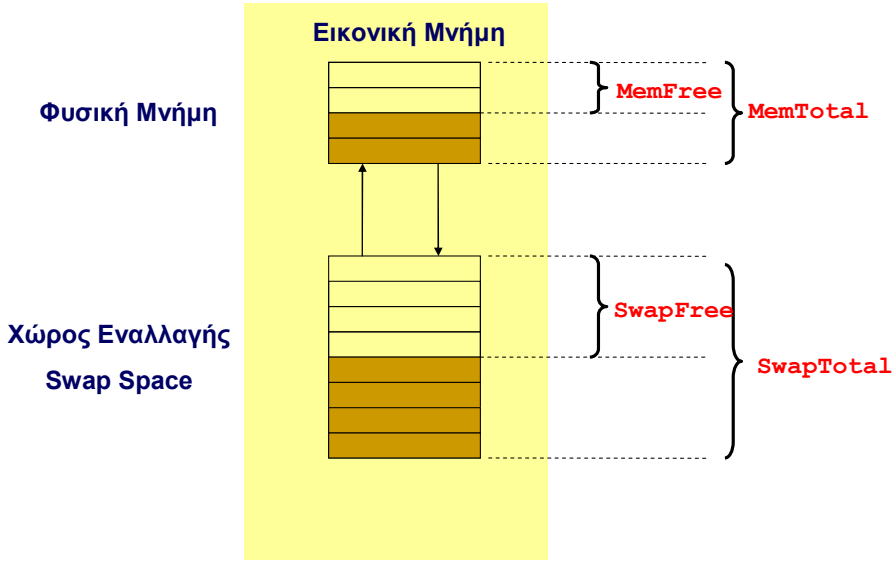
Σύνταξη	Σημασία
fg *%n*	Εργασία με αριθμό *n*
fg %+	Εργασία που στη λίστα της **jobs** έχει το σύμβολο + (Ίδιο με απλό fg)

fg %-	Εργασία που στη λίστα της **jobs** έχει το σύμβολο **-**
fg &?*name*	Εργασία με όνομα *name*

Τα ίδια ορίσματα μπορούν να χρησιμοποιηθούν και στην εντολή **bg**, αλλά εδώ έχουν νόημα μόνο για μία ακινητοποιημένη εργασία, αφού η jobs εμφανίζει εργασίες ή ακινητοποιημένες ή παρασκηνίου.

5. Φυσική και εικονική μνήμη.

Η διαχείριση μνήμης στο Linux επιτρέπει τη χρήση εικονικής μνήμης με τη χρήση σελιδοποίησης. Σε αυτό το μοντέλο υπάρχει ένας ενιαίος χώρος διεθυνσιοδότησης εικονικής μνήμης που αποτελείται από τη φυσική μνήμη (RAM) και από ένα τμήμα στο σκληρό δίσκο που είναι αφιερωμένο για αυτό το σκοπό και λέγεται χώρος *εναλλαγής σελίδων* (**swap space**). Ο σκοπός είναι η αύξηση της μνήμης που βλέπουν οι διεργασίες με μικρό κόστος. Ο χώρος της φυσικής μνήμης χωρίζεται σε ισομεγέθη τμήματα που λέγονται πλαίσια σελίδας (frame pages). Όταν η φυσική μνήμη γεμίσει, τότε το σύστημα διαχείρισης μνήμης που είναι συνδυασμός του πυρήνα του λειτουργικού συστήματος και του επεξεργαστή αποφασίζει ποια σελίδα από τη φυσική μνήμη θα μεταφερθεί στο σκληρό δίσκο προκειμένου να καταληφθεί από μία σελίδα που κρατιόταν στο σκληρό δίσκο και πρέπει να προσπελαστεί στη φυσική μνήμη.

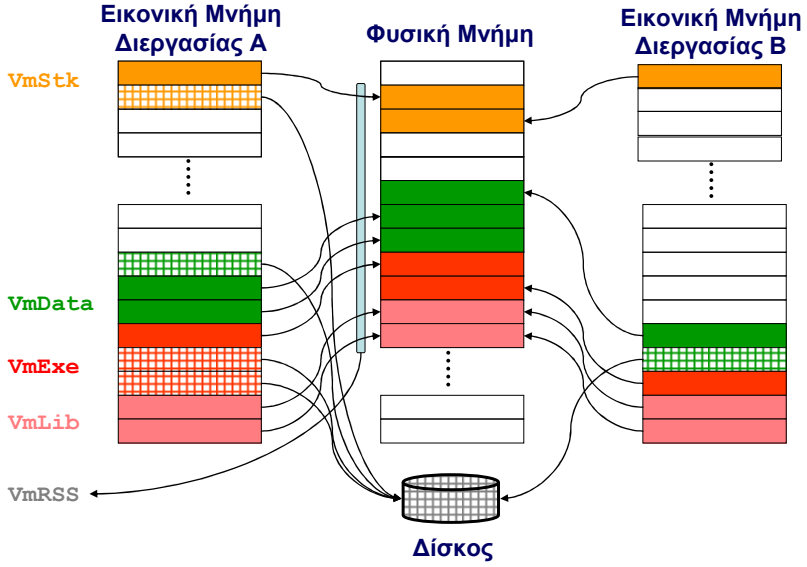


Το μέγεθος της σελίδας μπορεί να βρεθεί με την εντολή **getconf PAGESIZE** και συνήθως είναι 4096bytes. Ανά πάσα στιγμή μπορεί να ελεγχθεί η συνολική και διαθέσιμη ποσότητα φυσικής μνήμης και χώρου εναλλαγής που υπάρχει σε ένα σύστημα Linux. Η σχετική εντολή είναι :

```
$ cat /proc/meminfo
MemTotal:       516872 kB
MemFree:         10568 kB
...
SwapTotal:     1510068 kB
SwapFree:      1510064 kB
...
```

Τα περιεχόμενα του αρχείου έχουν πολλές γραμμές με πληροφορίες για τη μνήμη (όλα τα αρχεία στον κατάλογο `/proc` δεν είναι αρχεία του σκληρού δίσκου, αλλά ειδικά αρχεία του συστήματος που περιέχουν διάφορες πληροφορίες διαρρύθμισης). Οι γραμμές `MemTotal` και `MemFree` δείχνουν τη συνολική και την ελεύθερη φυσική μνήμη, αντίστοιχα. Οι γραμμές `SwapTotal` και `SwapFree` δείχνουν τη συνολική και την ελεύθερη μνήμη στο σκληρό δίσκο που διατίθεται για

εναλλαγή σελίδων, αντίστοιχα. Στο συγκεκριμένο παράδειγμα όλη η διατιθέμενη μνήμη σκληρού δίσκου είναι ελεύθερη, γιατί δεν έχει χρησιμοποιηθεί όλη η φυσική.
Εκτός από τη συνολική συμπεριφορά της μνήμης μπορεί να ελεγχθεί και η χρήση της μνήμης κατά διεργασία με διάφορους τρόπους. Κάθε διεργασία έχει πρόσβαση σε μία εικονική μνήμη που ένα μέρος της (ή και ολόκληρο) αντιστοιχεί στη φυσική μνήμη και ένα άλλο (όταν η φυσική μνήμη δεν έχει άλλο ελεύθερο χώρο) στο σκληρό δίσκο. Η ιδεατή μνήμη μίας διεργασίας χωρίζεται ανάλογα με τη χρήση στα εξής τέσσερα τμήματα : Μνήμη κώδικα (δηλαδή του προγράμματος σε μορφή δυαδικού κώδικα μηχανής), μνήμη δεδομένων (μεταβλητές κ.τ.λ.), στοίβα (κλήση υπορουτίνας κ.τ.λ.) και κοινή μνήμη βιβλιοθηκών (όταν κάποια μέρη του κώδικα χρησιμοποιούνται από έτοιμες βιβλιοθήκες). Ένα συνηθισμένο πρόγραμμα αποτελείται από το τμήμα του κώδικα το οποίο χρησιμοποιείται μόνο για ανάγνωση (αφού ο κώδικας δεν αλλάζει), αλλά και τμήμα των βιβλιοθηκών τις οποίες πιθανόν χρειάζεται ο κώδικας (π.χ. έτοιμες συναρτήσεις της C κ.α.). Η μνήμη που χρησιμοποιείται από το πρόγραμμα για ανάγνωση και εγγραφή είναι η μνήμη δεδομένων (δηλαδή οι μεταβλητές του προγράμματος) και ο χώρος για τη στοίβα.
Στο σχήμα που ακολουθεί απεικονίζονται δύο διεργασίες με διαφορετικό μέγεθος ιδεατής μνήμης και διαφορετικά τμήματα κώδικα, δεδομένων και στοίβας. Στο συγκεκριμένο παράδειγμα οι δύο διεργασίες τυχαίνει να έχουν κοινό τμήμα βιβλιοθήκης (shared).



VmSize = VmExe + VmLib + VmStk + VmData

Κάθε παραλληλόγραμμο αντιστοιχεί σε μία σελίδα. Σε κάθε τμήμα κάθε διεργασίας είναι δυνατόν κάποιες σελίδες του να βρίσκονται στη φυσική μνήμη και κάποιες άλλες σελίδες να βρίσκονται στο χώρο εναλλαγής σελίδων, δηλαδή το σκληρό δίσκο (αυτές που φαίνονται γραμμοσκιασμένες). Η αντιστοίχιση σελίδων στη φυσική μνήμη και το σκληρό δίσκο γίνεται από τον πυρήνα και αλλάζει συνεχώς. Έτσι, μία σελίδα ιδεατής μνήμης είναι δυνατόν μία χρονική στιγμή να βρίσκεται στη φυσική μνήμη, μία άλλη στιγμή στο σκληρό δίσκο και μία επόμενη πάλι στη φυσική μνήμη αλλά σε διαφορετικό πλαίσιο σελίδας.
Για πληροφορίες που αφορούν μία συγκεκριμένη διεργασία με αριθμό `pid` μπορεί κανείς να εμφανίσει το ψευδοαρχείο `/proc/pid/status`, όπου μεταξύ άλλων θα εμφανιστούν και πληροφορίες για την κατανομή της μνήμης, όπως παρακάτω :

```
$ cat /proc/5985/status
...
VmPeak:     1460 kB
VmSize:     1420 kB
VmLck:         0 kB
VmHWM:       244 kB
VmRSS:       244 kB
VmData:       28 kB
VmStk:        84 kB
VmExe:         4 kB
```

```
VmLib:      1284 kB
VmPTE:        12 kB
...
```

Στην πιο πάνω αναφορά τα μεγέθη των τμημάτων κώδικα, βιβλιοθηκών κώδικα, δεδομένων και στοίβας εμφανίζονται σαν `VmEXE`, `VmLib`, `VmData` και `VmStk`, αντίστοιχα. Το συνολικό μέγεθος ιδεατής μνήμης που είναι αντιστοιχισμένο σε φυσική μνήμη αναφέρεται σαν `VmRSS` (resident set size). Σε αυτό ανήκουν υποσύνολα των τεσσάρων προηγούμενων τμημάτων. Ο συνολικός εικονικός χώρος που έχει δεσμευτεί για την διεργασία έχει μέγεθος `VmSize` και μπορεί να είναι μεγαλύτερος από το άθροισμα των τεσσάρων τμημάτων `VmExe`, `VmLib`, `VmData` και `VmStk` γιατί μπορεί να έχει γίνει η δέσμευση (π.χ. με μία συνάρτηση `memalloc()` της C), αλλά να μην έχει γίνει ακόμα χρήση του από τη διεργασία.

Ανάλογες πληροφορίες μπορεί να δώσει και η εντολή **`top`**. Μία συνοπτική αναφορά για τη μνήμη δίνεται στην 4η και 5η γραμμή της εξόδου της εντολής και αφορά τη συνολική μνήμη του συστήματος. Η εντολή είναι διαλογική και μπορεί να αλλάξει διάφορες λειτουργίες της όπως τη μορφή της εξόδου της με το πάτημα συγκεκριμένων πλήκτρων. Για την προσθήκη στηλών όπως οι `SWAP`, `CODE` και `DATA` που κανονικά δεν εμφανίζονται έχει χρησιμοποιηθεί το πλήκτρο `f`. Ενώ για την αλλαγή της σχετικής θέσης των στηλών υπάρχει το πλήκτρο `o`.

```
top - 04:46:01 up  8:29,  1 user,  load average: 1.00, 0.98, 0.70
Tasks:  91 total,   3 running,  88 sleeping,   0 stopped,   0 zombie
Cpu(s):100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    516872k total,   308024k used,   208848k free,    22620k buffers
Swap:  1510068k total,        4k used,  1510064k free,   195060k cached

  PID USER      PR  NI  VIRT SWAP  RES CODE DATA  SHR S %CPU %MEM    TIME+  COMMAND
 5985 adp       25   0  1420 1176  244    4  112  192 R 99.9  0.0  19:26.68 loop1
 6001 adp       15   0  2228 1088 1140   48  352  860 R  0.3  0.2   0:00.11 top
 5938 adp       16   0  7884 6268 1616  312  712 1152 R  0.0  0.3   0:00.04 sshd
 5939 adp       15   0  5396 2480 2916  644 1748 1400 S  0.0  0.6   0:00.13 bash
```

Ακολουθεί η περιγραφή των στηλών που σχετίζονται με τη μνήμη. Η στήλη `VIRT` δίνει τη συνολική εικονική μνήμη, η `RES` τη φυσική μνήμη που χρησιμοποιεί η διεργασία, και η `SWAP` το μέρος της εικονικής μνήμης που βρίσκεται στο σκληρό δίσκο (έτσι ισχύει `VIRT=RES+SWAP`). Η `CODE` αφορά το μέγεθος του τμήματος κώδικα, η `DATA` αφορά τα τμήματα δεδομένων και στοίβας από κοινού και η `SHR` τη χρήση διαμοιραζόμενων βιβλιοθηκών. Τέλος, η στήλη `%MEM` δείχνει τι ποσοστό από τη φυσική μνήμη χρησιμοποιεί η κάθε διεργασία (έτσι θα ισχύει `%MEM`≈ `RES` /`MemTotal`).

Μία εξαιρετικά λεπτομερής ανάλυση της χρήσης σε επίπεδο διεργασίας μπορεί να παρουσιαστεί με την εντολή **`pmap -x`** *`pid`*. Δίνεται ένα παράδειγμα που αφορά το γνωστό πρόγραμμα `loop1`:

```
$ pmap -x 5985
5985:   ./loop1
Address   Kbytes     RSS    Anon  Locked Mode   Mapping
08048000       4       -       -       - r-x--  loop1
08049000       4       -       -       - rwx--  loop1
b7e5c000       4       -       -       - rwx--    [ anon ]
b7e5d000    1180       -       -       - r-x--  libc-2.3.6.so
b7f84000      20       -       -       - r-x--  libc-2.3.6.so
b7f89000       8       -       -       - rwx--  libc-2.3.6.so
b7f8b000      12       -       -       - rwx--    [ anon ]
b7f98000       8       -       -       - rwx--    [ anon ]
b7f9a000       4       -       -       - r-x--    [ anon ]
b7f9b000      84       -       -       - r-x--  ld-2.3.6.so
b7fb0000       8       -       -       - rwx--  ld-2.3.6.so
bfb11000      84       -       -       - rw---    [ stack ]
-------- ------- ------- ------- -------
total kB    1420       -       -       -
```

Κάθε γραμμή δείχνει και μία περιοχή εικονικής μνήμης με λεπτομέριες όπως τη διεύθυνση έναρξης και το μέγεθος, το είδος προσπέλασης (`rwx`) και τη χρήση. Η χρήση `[ anon ]` σημαίνει ότι η

περιοχή ανήκει στο τμήμα δεδομένων και η χρήση `[ stack ]` ότι ανήκει στο τμήμα της στοίβας. Όπου υπάρχει το όνομα του προγράμματος πρόκειται για το τμήμα του κώδικα, ενώ οι χρησιμοποιούμενες βιβλιοθήκες παρουσιάζονται με το όνομά τους και αυτές. Οι διευθύνσεις μνήμης αποτελούνται από 8 δεκαεξαδικά ψηφία γιατί η ιδεατή μνήμη έχει μέγεθος 4Gbytes (2^{32} bytes) σε επεξεργαστή Intel 32bits. Στο παράδειγμα με το πρόγραμμα `loop1` το άθροισμα δεδομένων και στοίβας που δίνει η **`pmap`** είναι 4+12+8+4+84kbytes=112kbytes που συμπίπτει με την αντίστοιχη έξοδο της **`top`** και της **`cat /proc/5985/status`** για το πεδίο `DATA` και `VmData+VmStk`, αντίστοιχα. Επίσης οι περιοχές r-x των βιβλιοθηκών δίνουν άθροισμα 1284kbytes, όσο δηλαδή και το πεδίο `VmLib`. Και τέλος το συνολικό μέγεθος χρήσης ιδεατής μνήμης που δίνει η **`pmap`** (1420kbytes) είναι ίδιο με τα πεδία `VIRT` και `VmSize` των δύο εντολών που προαναφέρθηκαν.

6. Η εντολή `ulimit`.

Τα αρχεία, οι διεργασία και η μνήμη θεωρούνται πόροι (resources) του συστήματος. Σε πολλούς από τους πόρους είναι δυνατόν να τεθούν όρια στη χρήση τους από (είτε απο απλό χρήστη, είτε από το διαχειριστή, ανάλογα με τον πόρο). Π.χ. είναι δυνατόν να τεθεί όριο στον αριθμό των διεργασιών που μπορεί να γεννήσει ένας χρήστη, είτε να τεθεί όριο στον αριθμό των ανοικτών αρχείων ή στο μέγεθος της μνήμης στοίβας. Οι τρέχουσες ρυθμίσεις εμφανίζονται με την εντολή `ulimit` η οποία πρέπει να δωθεί με μία επιλογή που θα δηλώνει το είδος του πόρου. Π.χ. ο έλεγχος για το μέγεθος της στοίβας μπορεί να ελεγχθεί με `ulimit -s`. Για επίδειξη όλων των ορίων υπάρχει η επιλογή `-a`.

```
$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
max nice                        (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) unlimited
max locked memory       (kbytes, -l) unlimited
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) unlimited
max rt priority                 (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) unlimited
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

Επίσης, είναι δυνατόν να γίνει αλλαγή της τρέχουσας ρύθμισης για κάποιον πόρο χρησιμοποιώντας τη σύνταξη **`ulimit -`***`resource_type newsize`*. Αν ήθελε κανείς να βάλει όρια στο μέγεθος της μνήμης που χρησιμοποιείται για τη στοίβα σε 16Mbytes από μία διεργασία θα πρέπει να δωθεί η εντολή **`ulimit -s 16384`**.

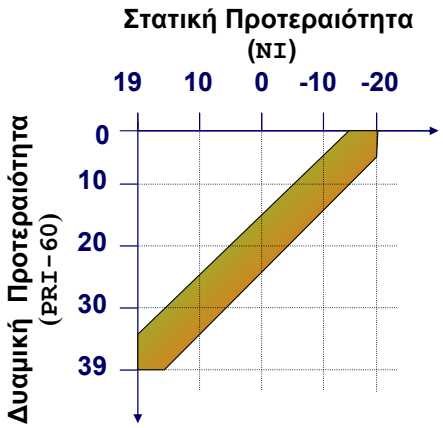
Άσκηση 1.

Δίνεται το παρακάτω πρόγραμμα `loop1.c` σε γλώσσα C.

```
main()
{
int i;
double a,b,c,d;
b = 1.32456;
c = 78910.123;
d = 34567.789;
i=1;
while (i>0)
        a = b * c + d;
}
```

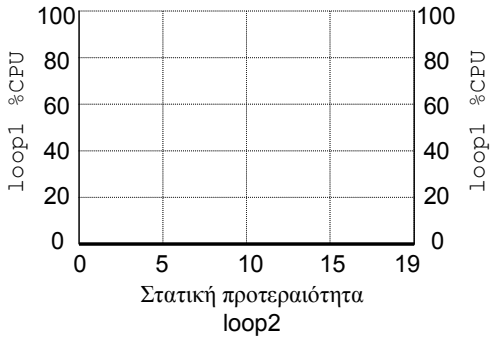
Το πρόγραμμα είναι ένας ατέρμονος βρόχος πολλαπλασιασμών και διαιρέσεων, δηλαδή πρόκειται για ένα CPU-bound πρόγραμμα. Η μεταγλώττισή του μπορεί να γίνει με έναν μεταγλωττιστή όπως ο gcc (Gnu C Compiler) που υπάρχει σχεδόν σε κάθε εγκατάσταση Linux. Η σχετική εντολή είναι **`gcc loop1.c -o loop1`** και το αποτέλεσμα θα είναι να παραχθεί ένα εκτελέσιμο αρχείο δυαδικού κώδικα μηχανής με όνομα `loop1` το οποίο θα μπορεί να εκτελεστεί εφ' όσον δοθούν οι σχετικές εξουσιοδοτήσεις, δηλαδή **`chmod u+x loop1`**.

1. Εκτελέστε το πρόγραμμα `loop1` σε έναν υπολογιστή που δεν είναι φορτωμένος (πεδίο `load average` της εντολής **`top`** περίπου 0).
2. Σε διαφορετικό παράθυρο εκτελέστε την εντολή **`top -d 5`** (με ανανέωση κάθε 5 δευτερόλεπτα) παρατηρώντας τη δυναμική προτεραιότητα της διεργασίας που αντιστοιχεί στο `loop1` καθώς και τα πεδία `%CPU`, `TIME+`, `load average` και `task running`. Συγκρίνετε το χρόνο που μένει η διεργασία στον επεξεργαστή σε κάθε ανανέωση της top (διαφορές δύο διαδοχικών TIME+) με τον πραγματικό που περνάει κάθε φορά (χρόνος ανανέωσης της top).
3. Αλλάξτε διαδοχικά χρησιμοποιώντας την εντολή **`renice`** την στατική προτεραιότητα της διεργασίας loop1 σε τιμές 5, 10, 15 και 19 και βρείτε τη δυναμική προτεραιότητα που αποκτά η διεργασία καθώς και την επιδότησή της. Συμπληρώστε στην παρακάτω γραφική παράσταση με σημάδια τα ζευγάρια στατικών και δυναμικών προτεραιοτήτων που προκύπτουν κάθε φορά (μαζί με την περίπτωση στατικής προτεραιότητας 0 από το προηγούμενο βήμα).



4. Διακόψτε τη διεργασία των προηγούμενων βημάτων. Αντιγράψτε το εκτελέσιμο αρχείο `loop1` σε άλλα δύο αρχεία με ονόματα `loop2` και `loop3` διατηρώντας την εξουσιοδότηση για εκτέλεση.
5. Εκτελέστε το πρόγραμμα `loop1` με την εξ' ορισμού στατική προτεραιότητα 0 και εκτελέστε το πρόγραμμα loop2 με διαδοχικές στατικές προτεραιότητες 0,5,10,15,19. Κάθε φορά παρατηρείτε τα πεδία `TIME+` και `%CPU` της εντολής **`top`**. Συμπληρώστε τον παρακάτω πίνακα και την γραφική παράσταση.

loop2 Σ.Π.	0	5	10	15	19
loop1 %CPU					
loop2 %CPU					



Ταιριάζουν τα αποτελέσματα με τη θεώρηση ότι το κβάντο χρόνου που έχει κάθε διεργασία είναι ανάλογο με τη στατική προτεραιότητα; Στον πυρήνα 2.6.x του Linux συνήθως το ελάχιστο κβάντο χρόνου είναι 10ms και αντιστοιχεί σε Σ.Π. 19 (niceness), ενώ το μέγιστο είναι 800ms και αντιστοιχεί σε Σ.Π. -20.
6. Στην περίπτωση που ο υπολογιστής έχει πάνω από έναν επεξεργαστή ή ο επεξεργαστής είναι πολυπύρηνος (multicore) επαναλάβετε το προηγούμενο βήμα τρέχοντας όμως αυτή τη φορά τρεις διεργασίες, δηλαδή τρέχοντας τα τρία πανομοιότυπα προγράμματα `loop1`, `loop2`, και `loop3`. Σε αυτήν την περίπτωση καταγράψτε τα αποτελέσματα για τις δύο διεργασίες που τρέχουν στον ίδιο πυρήνα ή στον ίδιο επεξεργαστή.

Άσκηση 2.

Δίνεται το παρακάτω πρόγραμμα `keyloop.c` σε γλώσσα C.

```
#include <stdio.h>
#include <stdlib.h>
main()
{
int ch;
int i;
double a,b,c,d;
b = 1.32456;
c = 78910.123;
d = 34567.789;
while (1)
          {
          printf(" hit any key - g to enter loop \n");
          ch=getchar();
          if (ch=='g')
               {
               printf(" g key pressed - running loop \n");
               for (i=0;i<2000000000;i++)
                              a = b * c + d;
               }
          }
}
```

Το πρόγραμμα είναι ένας ατέρμονος βρόχος που περιμένει συνεχώς είσοδο από το πληκτρολόγιο (I/O bound). Αν πατηθεί το πλήκτρο g τότε, εκτελεί για κάποια δευτερόλεπτα υπολογισμούς (CPU bound) και ξαναπεριμένει είσοδο. Αλλιώς, αν πατηθεί άλλο πλήκτρο συνεχίζει να περιμένει νέα είσοδο. Ουσιαστικά με το πλήκτρο g το πρόγραμμα μεταπίπτει στην κατηγορία CPU bound για κάποια δευτερόλεπτα.

1. Τρέξτε το πρόγραμμα και καταγράψτε με τη χρήση της εντολής top την δυναμική προτεραιότητα (α) για την περίπτωση αναμονής εισόδου και (β) για την περίπτωση που το πρόγραμμα κάνει υπολογισμούς.
2. Κάντε το ίδιο για την περίπτωση που το πρόγραμμα έχει στατική προτεραιότητα 10.

Άσκηση 3.

Χρησιμοποιήστε το πρόγραμμα `loop1` και εκτελέστε το. Περάστε το διαδοχικά από τις εξής καταστάσεις :

α.Προσκήνιο (foreground).
β.Ακινητοποίηση (stopped).
γ.Παρασκήνιο (background).
δ.Προσκήνιο.
ε.Τερματισμός.

Σε κάθε βήμα παρατηρείστε την κατάσταση της αντίστοιχης διεργασίας όπως φαίνεται από την εντολή **`top`**, καθώς και το πεδίο `%CPU`.

Άσκηση 4.

Χρησιμοποιήστε το πρόγραμμα `loop1` και εκτελέστε το στο παρασκήνιο. Περάστε το διαδοχικά από τις εξής καταστάσεις :

α. Παρασκήνιο.
β.Ακινητοποίηση.
γ.Προσκήνιο.
δ.Τερματισμός.

Σε κάθε βήμα παρατηρείστε την κατάσταση της αντίστοιχης διεργασίας όπως φαίνεται από την εντολή **`top`**, καθώς και το πεδίο `%CPU`.

Επί πλέον ασκήσεις.

Άσκηση 5.

Δίνεται το παρακάτω πρόγραμμα `vmfill.c` σε γλώσσα C.

```
#include <stdio.h>
#include <stdlib.h>
#define KILOBYTE 1024
int main(int argc, char *argv[])
{
void *myblock = NULL;
int count = 0;          // total blocks allocated
long totalSize = 0;     // total size allocated in kBytes
int blockSize = 10;     // block size in kBytes
while (1)
      {
      myblock = (void *) malloc(blockSize*KILOBYTE);
            if (!myblock) break;
```

```
            count++;
            totalSize = count*blockSize;
            printf("Currently allocating %u KB\n", totalSize);
            }
exit(0);
}
```

Το πρόγραμμα δεσμεύει διαδοχικά με τη χρήση ενός ατέρμονου βρόχου κομμάτια μνήμης μεγέθους `blocksize`. Τα κομμάτια μνήμης δε χρησιμοποιούνται, αλλά απλώς δεσμεύονται. Σε κάθε επανάληψη του βρόχου το πρόγραμμα εμφανίζει το μέγεθος της μ΄νημης που έχει δεσμευθεί. Το πρόγραμμα τερματίζεται όταν υπάρχει αδυναμία δέσμευσης επί πλέον χώρου.

1. Εκτελέστε το παραπάνω πρόγραμμα και παρατηρήστε με την βοήθεια την εντολής top την εξέλιξη του μεγέθους της εικονικής μνήμης (`VIRT`), φυσικής μνήμης (`RES`), μνήμης σκληρού δίσκου (`SWAP`) και μνήμης δεδομένων και στοίβας (`DATA`) που δεσμεύεται κατά τη διάρκεια της εκτέλεσης μέχρι και την διακοπή του προγράμματος.
2. Σημειώστε το μέγεθος της μνήμης που έχει δεσμευθεί ακριβώς πριν από τον τερματισμό του προγράμματος.

Άσκηση 6.

Δίνεται το παρακάτω πρόγραμμα `vmfillset.c` σε γλώσσα C.

```
#include <stdio.h>
#include <stdlib.h>
#define KILOBYTE 1024
int main(int argc, char *argv[])
{
void *myblock = NULL;
int count = 0;          // total blocks allocated
long totalSize = 0;     // total size allocated in kBytes
int blockSize = 10;     // block size in kBytes
while (1)
        {
        myblock = (void *) malloc(blockSize*KILOBYTE);
        if (!myblock) break;
        memset(myblock,1, blockSize*KILOBYTE);
        count++;
        totalSize = count*blockSize;
        printf("Currently allocating %u KB\n", totalSize);
        }
exit(0);
}
```

Το πρόγραμμα `vmfillset.c` κάνει ότι ακριβώς και το πρόγραμμα `vmfill.c` της προηγούμενης άσκησης, με τη διαφορά ότι χρησιμοποιεί πραγματικά την μνήμη που δεσμεύεται (γραμμή στον κώδικα C : `memset(myblock,1, blockSize*KILOBYTE);`) , δηλαδή γεμίζει με δυαδικά «1» όλες τις θέσεις μνήμης που δεσμεύονται.

1. Σημειώστε το μέγεθος της μνήμης που έχει δεσμευθεί ακριβώς πριν από τον τερματισμό του προγράμματος. Ποια είναι η σχέση της τιμής που βρήκατε με το μέγεθος της φυσικής μνήμης, της ελεύθερης φυσικής μνήμης και της μνήμης εναλλαγής (swap space);
2. Γιατί η τιμή αυτή διαφέρει σε σχέση με την τιμή της μνήμης που δεσμεύθηκε κατά την προηγούμενη άσκηση;

Άσκηση 7.

Δίνεται το παρακάτω πρόγραμμα `stack.c` σε γλώσσα C.

```
#include <stdio.h>
void f(int i);

int main(void)
{
f( 0 );
return 0;
}

void f(int i)
{
printf("%d \n", i);
f(i+1);
}
```

Το πρόγραμμα `stack.c` έχει ορισμένη τη συνάρτηση `f` η οποία δέχεται σαν όρισμα έναν ακέραιο, τον εμφανίζει στην οθόνη, τον αυξάνει κατά 1 και καλεί τον εαυτό της. Επομένως οι διαδοχικές κλήσεις του εαυτού της έχουν σαν αποτέλεσμα τη συνεχή αύξηση του μεγέθους της στοίβας.

1. Εκτελέστε το πρόγραμμα και σημειώστε στις πόσες κλήσεις γίνεται διακοπή του προγράμματος (segmentation fault) εξ΄ αιτίας υπέρβασης ορίου μνήμης. Παρακολουθώντας με την εντολή **`top`** το μέγεθος της μνήμης στοίβας διαπιστώστε το μέγεθος μνήμης στοίβας που χρησιμοποιήθηκε είναι συμβατό με το όριο που δίνει η εντολή **`ulimit -s`**. (Η **`top`** δίνει από κοινού τη χρήση μνήμης και δεδομένων στο πεδίο `DATA`. Όμως η χρήση δεδομένων σε αυτό το πρόγραμμα είναι ελάχιστη και επομένως η χρήση της στοίβας δίνεται με μεγάλη ακρίβεια από το πεδίο `DATA`).
2. Πόσα bytes χρησιμοποιούνται στη στοίβα για κάθε κλήση και που χρησιμοποιούνται ;
3. Διπλασιάστε το όριο χρήση στοίβας με την εντολή **`ulimit -s`** *`size`* (όπου *`size`* είναι το νέο όριο σε kbytes) και επαναλάβατε το βήμα 1.

Βιβλιογραφία

1. Το Περιβάλλον Προγραμματισμού UNIX –B.W.Kernighan,R.Pike, 2001 Κλειδάριθμος
2. Teach Yourself UNIX in 24 Hours, 3rd Ed. – D.Taylor, 2005 SAMS
3. Advanced Bash-shell Scripting Guide – R.Cooper, (http://www.tldp.org/LDP/abs/html/)
4. Learning the bash Shell, 2nd Ed. – C.Newham, B.Rosenblatt, 1998 O' Reilly
5. UNIX Shells by Example, 4th Ed. – E.Quigley, 2004 Prentice Hall
6. Unix Shell Programming, 3rd Ed. – S.G.Kochan, P.Wood, 2003 SAMS
7. Mastering Regular Expressions, 3rd Ed.- J. E.F.Friedl, 2006 O' Reilly,
8. System Performance Tuning, 2nd Ed. - GP.D.Musumeci,M.Loukides, 2002 O' Reilly
9. Understanding the Linux 2.6.8.1 CPU Scheduler – J.Aas, 2005 Silicon Graphics, (http://josh.trancesoftware.com/linux/linux_cpu_scheduler.pdf)

10. http://www.tldp.org/ (Διάφορα βιβλία για Linux)