Λειτουργικά Συστήματα

Σημειώσεις εργαστηρίου

Θεόδωρος Θεμελής

Περιεχόμενα

Εισαγωγή

Σύντομο ιστορικό

Το UNIX δημιουργήθηκε το 1969 από μια ομάδα μηχανικών της AT&T, μερικοί από τους οποίους ήταν οι Ken Thompon, Dennis Ritchie, Brian Kernighan και Douglas McIlroy. Από τότε, το UNIX έχει χρησιμοποιηθεί σαν βάση για τη δημιουργία πολλών εμπορικών και ανοικτού κώδικα λειτουργικών συστημάτων. Γενικά, μπορούμε να διακρίνουμε τις παρακάτω κατηγορίες:

Unix Εμπορικές υλοποιήσεις οι οποίες προέρχονται από τον αρχικό κώδικα της AT&T. Μερικά γνωστά λειτουργικά συστήματα που στηρίζονται στο UNIX είναι: AIX (Το UNIX της IBM), HP-UX (Το UNIX της Hewlett Packard), Solaris (Το UNIX της Sun Microsystems)

BSD Υλοποιήσεις ανοικτού κώδικα, οι οποίες προέρχονται από τον αρχικό κώδικα. Η ανάπτυξη του BSD (Berkeley Software Distribution) ξεκίνησε στην αρχή της δεκαετίας του 1970, αρχικά ως κλώνος του κώδικα της AT&T, αλλά σταδιακά ξαναγράφτηκε με σκοπό την αφαίρεση όλου του κώδικα που προστατευόταν με πνευματική ιδιοκτησία. Μερικά γνωστά λειτουργικά συστήματα που στηρίζονται στο BSD είναι: Darwin (η βάση των λειτουργικών συστημάτων της Apple), FreeBSD, OpenBSD, NetBSD

Linux Υλοποιήσεις ανοικτού κώδικα, οι οποίες προέρχονται από τον κώδικα που δημιούργησε ο Linus Torvalds όντας φοιτητής του πανεπιστημίου του Ελσίνκι το 1991. Το λειτουργικό σύστημα που δημιούργησε ο Torvalds δεν στηριζόταν στον κώδικα της αρχικής υλοποίησης του UNIX, παρά μόνο στη φιλοσοφία του. Μερικά γνωστά λειτουργικά συστήματα που στηρίζονται στο Linux είναι: RedHat, Fedora, Debian, Ubuntu, Suse, CentOS.

Ο πυρήνας (kernel)

Η καρδιά ενός λειτουργικού συστήματος τύπου UNIX είναι ο πυρήνας. Είναι το μοναδικό κομμάτι του λειτουργικού συστήματος το οποίο πρέπει να παραμένει καθ' όλη τη διάρκεια λειτουργίας του υπολογιστή, στη μνήμη. Ουσιαστικά, ο πυρήνας παίζει το ρόλο του μαέστρου ο οποίος διαχειρίζεται το υπολογιστικό σύστημα. Κάθε φορά που πληκτρολογούμε μια εντολή στο πληκτρολόγιο το υλικό του υπολογιστή μεταφέρει τα σήματα στον πυρήνα του ΛΣ για την περαιτέρω επεξεργασία τους.

Διανομές Linux

Όταν αναφερόμαστε στο Linux, ουσιαστικά μιλάμε για τον πυρήνα του λειτουργικού συστήματος, τον οποίο μπορούμε να τον προμηθευτούμε από τη διεύθυνση http://www.kernel.org. Αν μαζί με τον πυρήνα πακετάρουμε ένα σύνολο από προγράμματα συστήματος και εφαρμογών, τότε δημιουργείται μια διανομή (Linux distribution). Οι πιο γνωστές διανομές Linux είναι οι ακόλουθες:

Debian (http://www.debian.org) Μια δωρεάν, αλλά λίγο «δύσκολη» διανομή για αρχάριους, με μεγάλη ιστορία

RedHat Enterprise Linux (http://www.redhat.com) Μια διανομή με πληρωμή η οποία προορίζεται κυρίως για εταιρικά περιβάλλοντα

Fedora (http://fedoraproject.org)	Δωρεάν διανομή, η οποία χρησιμοποιείται από τη RedHat σαν δοκιμαστική για την έκδοση με πληρωμή
Ubuntu (http://www.ubuntu.com)	Δωρεάν διανομή η οποία στηρίζεται στην Debian και είναι πολύ δημοφιλής στους αρχάριους χρήστες

Για να δείτε και να κατεβάσετε τις περισσότερες από τις διανομές που υπάρχουν αυτήν την στιγμή, μπορείτε να μπείτε στη διεύθυνση http://distrowatch.com.

Φλοιός (Shell)

Ο φλοιός (Command line interpreter), είναι ένα πρόγραμμα του οποίου η εργασία είναι να περιμένει μια είσοδο από τον χρήστη (τυπικά μια εντολή οποιουδήποτε είδους) και να προσπαθεί να εκτελέσει την επιθυμητή λειτουργία. Αρχικά, ο φλοιός ελέγχει αν η εντολή που έχουμε εισάγει είναι συντακτικά σωστή. Αν είναι, θα καλέσει το συγκεκριμένο πρόγραμμα, ενώ αν δεν είναι θα εμφανίσει κάποιο μήνυμα λάθους. Από την στιγμή που θα ολοκληρωθεί η εκτέλεση ενός προγράμματος ο χρήστης επιστρέφει στον φλοιό.

Σήμα προτροπής (Command prompt)

Αριστερά από τον δρομέα (cursor) βρίσκεται το σήμα προτροπής του φλοιού (command prompt), το οποίο μας παρέχει κάποιες πληροφορίες. Η μορφή του σήματος προτροπής μπορεί να αλλάξει, αλλά γενικά χρησιμοποιείται η παρακάτω μορφή:

user	**@**	**localhost**	**:**	**~**	**$**
Το όνομα του χρήστη με το οποίο συνδέθηκε στο σύστημα	Διαχωριστικό	Το όνομα του συστήματος	Διαχωριστικό	Τρέχοντας φάκελο	Τύπος χρήστη

Το τελευταίο τμήμα του σήματος προτροπής μπορεί να είναι είτε το $ είτε η #. Το μεν $ υποδηλώνει ότι ο συγκεκριμένος χρήστης είναι ένας απλός χρήστης, ο οποίος δεν έχει πολλά δικαιώματα χρήσης στον υπολογιστή. Τυπικά, είναι ιδιοκτήτης και περιορίζεται μέσα σε ένα φάκελο, μέσα στον οποίο μπορεί να κάνει σχεδόν ότι θέλει. Η # υποδηλώνει ότι ο χρήστης είναι ο διαχειριστής (root) του συστήματος ο οποίος έχει όλα τα δικαιώματα χρήσης και μπορεί να κάνει οτιδήποτε θέλει στο σύστημα.

Τύποι φλοιών

Υπάρχουν αρκετοί τύποι φλοιών στον κόσμο του Linux. Κάθε ένας φλοιός έχει τα δικά του πλεονεκτήματα και μειονεκτήματα. Μερικοί από τους πιο διαδεδομένους φλοιούς είναι οι sh, bash, tcsh και ksh.

Στις παρούσες σημειώσεις θα ασχοληθούμε με τον bash, ο οποίος είναι η εξέλιξη του sh.

Σύνταξη εντολών

Υπάρχουν δύο ειδών εντολές, οι εσωτερικές οι οποίες είναι ενσωματωμένες στον φλοιό και οι εξωτερικές οι οποίες είναι προγράμματα τα οποία είναι αποθηκευμένα στο σύστημα αρχείων και ο φλοιός αναλαμβάνει να τα βρει και να τα εκτελέσει. Ο φλοιός δεν ψάχνει οπουδήποτε για τις εξωτερικές εντολές, αλλά μόνο στους φακέλους που ορίζονται στην μεταβλητή PATH.

Μία εντολή που γράφουμε στο τερματικό, ουσιαστικά είναι σαν μία πρόταση σε φυσική γλώσσα. Κάθε λέξη ή χαρακτήρας έχει ξεχωριστό νόημα και θα πρέπει να ακολουθούμε ένα συγκεκριμένο τρόπο σύνταξης, ο οποίος στη γενική του μορφή είναι:

```
εντολή [επιλογές] [ορίσματα]
```

- Όλες οι εντολές γράφονται με πεζούς χαρακτήρες
- [] ότι βρίσκεται μέσα σε αγκύλες είναι προαιρετικό (υπάρχουν εντολές που δεν δέχονται ούτε επιλογές ούτε ορίσματα)
- επιλογές (options), τροποποίηση της συμπεριφοράς της εντολής. Πριν από κάθε επιλογή εισάγουμε μια παύλα (-). Οι επιλογές θα πρέπει να χωρίζονται με κενό από τα ορίσματα (αν υπάρχουν).
- ορίσματα (arguments), είσοδος στοιχείων προς την εντολή. Το κάθε ένα στοιχείο πρέπει να χωρίζεται από το άλλο με κενό

Για παράδειγμα, με την εντολή ls, χωρίς ορίσματα και χωρίς επιλογές, μπορούμε να δούμε τα περιεχόμενα του τρέχοντα φακέλου

```
user@localhost:~$ ls
Desktop Music Videos Documents Downloads exercise1 exercise2
```

Αν στην παραπάνω εντολή προσθέσουμε την επιλογή -l τότε θα δούμε τα περιεχόμενα του τρέχοντα φακέλου με λεπτομέρειες

```
user@localhost:~$ ls -l
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Desktop
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Documents
drwxr-xr-x 4 user user 4096 2009-01-01 00:00 Downloads
drwxr-xr-x 5 user user 4096 2009-01-01 00:00 Music
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Videos
-rwxr-xr-x 1 user user 196 2009-01-01 00:00 exercise1
-rwxr-xr-x 1 user user 8996 2009-01-01 00:00 exercise2
```

Αν στην εντολή ls προσθέσουμε το όρισμα Desktop τότε θα δούμε τα περιεχόμενα του φακέλου Desktop

```
user@localhost:~$ ls Desktop
Document1 report Analysis
```

Φυσικά, μπορούμε να εισάγουμε και επιλογές και ορίσματα. Για παράδειγμα αν θέλουμε να δούμε τα περιεχόμενα του φακέλου Desktop με λεπτομέρειες, θα γράψουμε

```
user@localhost:~$ ls -l Desktop
```

Σε περίπτωση που έχουμε πληκτρολογήσει μία εντολή που δεν υπάρχει ή έχουμε εισάγει λάθος ορίσματα ή επιλογές τότε ο φλοιός θα εμφανίσει κάποιο μήνυμα λάθους. Για παράδειγμα αν εισάγουμε την εντολή

```
user@localhost:~$ Ls
Ls: command not found
```

θα πάρουμε το μήνυμα λάθους που μας πληροφορεί ότι δεν υπάρχει εντολή με όνομα Ls. Αν εισάγουμε την εντολή

```
user@localhost:~$ ls -w
ls: option requires an argument -- 'w'
Try `ls --help' for more information.
```

Μας πληροφορεί ότι δεν έχουμε εισάγει κάποιο όρισμα στην επιλογή w.

Εγχειρίδιο χρήσης εντολών

Επειδή είναι πολύ δύσκολο να θυμόμαστε όλες τις επιλογές και τα ορίσματα που μπορεί να πάρει κάθε μία εντολή, υπάρχει ένα ενσωματωμένο σύστημα μέσω του οποίου μπορούμε να δούμε το εγχειρίδιο χρήσης κάθε εντολής. Έτσι, αν θέλουμε να δούμε το εγχειρίδιο χρήσης της εντολής ls θα πρέπει να πληκτρολογήσουμε

```
user@localhost:~$ man ls
```

Στο εγχειρίδιο χρήσης μπορούμε να βρούμε όλες τις πληροφορίες που αφορούν την εντολή που έχουμε εισάγει σαν όρισμα στην man.

Η μετακίνηση μέσα στο εγχειρίδιο χρήσης γίνεται με την χρήση των πλήκτρων PgUp, PgDn και με τα βελάκια. Για έξοδο από το εγχειρίδιο χρήσης πρέπει να πατήσουμε το πλήκτρο q.

Χρήσιμες εντολές

Παρακάτω θα κάνουμε αναφορά σε μερικές χρήσιμες εντολές:

clear Με αυτήν την εντολή καθαρίζουμε την οθόνη από όλα τα περιεχόμενα που έχει.

date Με αυτήν την εντολή προβάλλουμε την ημερομηνία και την ώρα του συστήματος.

```
user@localhost:~$ date
Mon Dec 01 00:00:00 EET 2019
user@localhost:~$ date +%H:%M
23:06
```

Στη δεύτερη περίπτωση, χρησιμοποιώντας ορίσματα, εμφανίζεται μόνο η ώρα του συστήματος.

cal Για την προβολή του ημερολογίου ενός μήνα χρησιμοποιούμε την εντολή cal. Αν δεν εισάγουμε καμία επιλογή, θα προβληθεί το ημερολόγιο του τρέχοντα μήνα.

```
user@localhost:~$ cal
```

```
   December 2019
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

Αν θέλουμε να προβάλλουμε το ημερολόγιο του Σεπτεμβρίου του 2004, θα πρέπει να εισάγουμε δύο ορίσματα.

```
user@localhost:~$ cal 9 2004
```

```
   September 2004
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
```

who Με αυτήν την εντολή μπορούμε να δούμε όλους τους συνδεδεμένους χρήστες του συστήματος.

```
user@localhost:~$ who
user        tty1  2001-12-01 00:00
user1       tty2  2001-12-01 00:01
user2       tty3  2001-12-01 00:01
```

Στο παραπάνω παράδειγμα, το σύστημα υπάρχουν 3 χρήστες συνδεδεμένοι. Το όνομα χρήστη του καθένα εμφανίζεται στην πρώτη στήλη, το τερματικό σύνδεσης είναι στη δεύτερη στήλη, ενώ στην τρίτη και στην τέταρτη στήλη εμφανίζεται η ημερομηνία και η ώρα σύνδεσης.

Ειδικοί χαρακτήρες

Στον φλοιό, μπορούμε να χρησιμοποιήσουμε αρκετούς χαρακτήρες οι οποίοι έχουν ειδικό νόημα και είναι οι παρακάτω:

[κενό διάστημα] Τα κενά είναι πολύ σημαντικά και θα πρέπει να προσέχουμε που τα εισάγουμε, γιατί ο bash μέσω αυτών ξεχωρίζει τις εντολές από τα ορίσματα και τις επιλογές.

***** Αντικαθιστά οσουσδήποτε και οποιουσδήποτε χαρακτήρες

? Αντικαθιστά έναν, οποιονδήποτε χαρακτήρα

" " Ότι πληκτρολογήσουμε μέσα σε διπλά εισαγωγικά θα παραμείνει όπως το γράψαμε. Δηλαδή, μία πρόταση δεν θα χωριστεί στις λέξεις από τις οποίες αποτελείται. Επίσης, μέσα στα διπλά εισαγωγικά ακυρώνεται το ειδικό νόημα των μονών εισαγωγικών κάτι που δεν ισχύει για τους άλλους ειδικούς χαρακτήρες.

```
user@localhost:~$ echo "I am $USER"
I am user
```

Στο παραπάνω παράδειγμα το $USER είναι μία μεταβλητή.

' ' Βασικά κάνουν την ίδια δουλειά με τα διπλά εισαγωγικά, μόνο που εδώ ακυρώνεται το νόημα όλων των ειδικών χαρακτήρων.

```
user@localhost:~$ echo "I am $USER"
I am $USER
```

Ότι πληκτρολογήσουμε μετά από την δίεση ο φλοιός θα το αγνοήσει, γιατί θεωρεί ότι είναι ένα σχόλιο. Τα σχόλια εκτείνονται μέχρι το τέλος της γραμμής.

; Το ελληνικό ερωτηματικό χρησιμοποιείται για τον διαχωρισμό συνεχόμενων εντολών τις οποίες θέλουμε να γράψουμε σε μία γραμμή.

**** Αν εισαχθεί το backslash πριν από κάποιον χαρακτήρα, τότε αυτός ο χαρακτήρας θα χρησιμοποιηθεί χωρίς να πάρει κάποιο ειδικό νόημα. Ονομάζεται χαρακτήρας διαφυγής (escape character).

> Το αποτέλεσμα μιας εντολής ανακατευθύνεται σε κάποια άλλη

	έξοδο και όχι στην οθόνη.
|	Εισάγοντας αυτόν τον χαρακτήρα ανάμεσα από δύο εντολές, στέλνουμε τα αποτελέσματα της πρώτης εντολής σαν δεδομένα στην δεύτερη εντολή. Ονομάζεται χαρακτήρας σωλήνωσης (pipeline).
.	Συμβολίζει τον τρέχοντα φάκελο.
..	Συμβολίζει το φάκελο ο οποίος βρίσκεται ένα επίπεδο πιο πάνω από το φάκελο στον οποίο βρισκόμαστε (τρέχοντα). Με άλλα λόγια είναι ο γονικός φάκελος του τρέχοντα φακέλου.
~	Συμβολίζει τον προσωπικό φάκελο του χρήστη.
[[έκφραση]]	Γίνεται έλεγχος της έκφρασης αν είναι αληθής ή ψευδής.
$(εντολή)	Γίνεται εκτέλεση της εντολής που βρίσκεται μέσα στην παρένθεση και επιστρέφεται το αποτέλεσμά της στην συγκεκριμένη θέση.
(εντολή)	Δημιουργείται ένας υποφλοιός (subshell) στον οποίο γίνεται η εκτέλεση της εντολής.
((έκφραση))	Χρησιμοποιείται για την εκτέλεση μαθηματικών πράξεων.

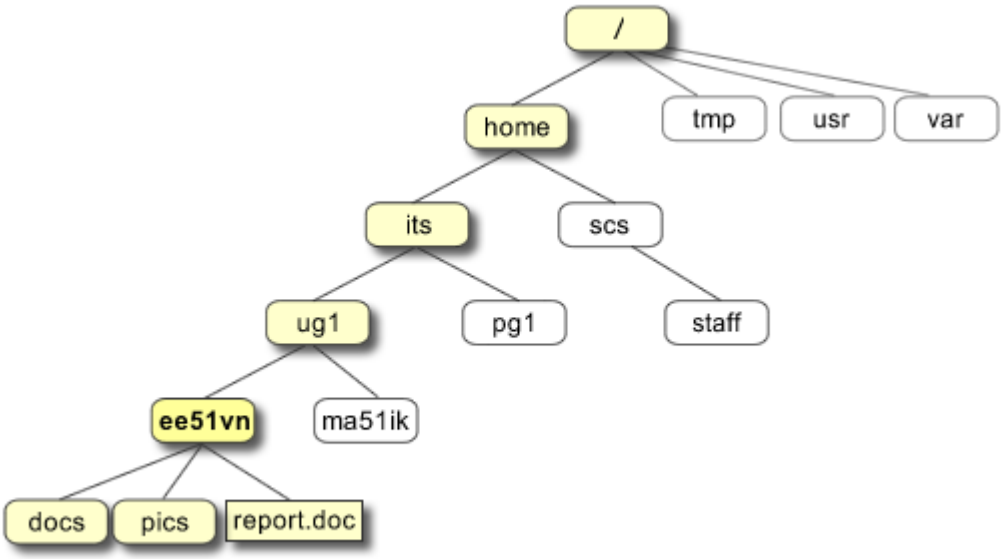
Σύστημα αρχείων

Γενικά, στον κόσμο του Linux ισχύει:

«Σ' ένα σύστημα UNIX όλα είναι αρχεία. Αν κάτι δεν είναι αρχείο, τότε είναι διεργασία».

Η παραπάνω πρόταση δεν είναι υπερβολική, γιατί σε ένα σύστημα Linux υπάρχουν ειδικά αρχεία που επιτελούν κάτι πολύ περισσότερο από ένα απλό αρχείο. Επίσης, δεν υπάρχει διαφορά σε ένα αρχείο και ένα φάκελο εκτός του ότι ο φάκελος είναι ένα αρχείο το οποίο περιέχει ονόματα άλλων αρχείων. Ακόμα και οι συσκευές εισόδου και εξόδου θεωρούνται αρχεία, μέσω των οποίων αποκτούμε πρόσβαση στις συγκεκριμένες συσκευές.

Για την ευκολότερη διαχείριση όλων αυτών των αρχείων ακολουθείται μια ιεραρχική δομή (σαν ένα ανάποδο δέντρο), όπου υπάρχουν φάκελοι μέσα σε άλλους φακέλους οι οποίοι στο τέλος μπορεί να περιέχουν αρχεία.



Εικόνα 1 Δενδρική δομή

Φάκελοι συστήματος

Σχεδόν σε όλες τις διανομές Linux υπάρχουν κάποιοι φάκελοι στους οποίους βρίσκονται αποθηκευμένα κάποια αρχεία τα οποία είναι χρήσιμα για το σύστημα. Μερικοί από αυτούς τους φακέλους είναι οι παρακάτω:

/bin	Φάκελος στον οποίο βρίσκονται τα περισσότερα προγράμματα στα οποία έχουν πρόσβαση όλοι οι χρήστες του συστήματος.
/sbin	Φάκελος στον οποίο υπάρχουν προγράμματα στα οποία έχει πρόσβαση, συνήθως μόνο, ο διαχειριστής.
/boot	Σ' αυτόν τον φάκελο υπάρχουν όλα τα αρχεία που είναι απαραίτητα για την εκκίνηση του συστήματος.
/dev	Εδώ βρίσκονται όλα τα αρχεία συσκευών. Δηλαδή τα αρχεία μέσω των οποίων μπορούμε να έχουμε πρόσβαση στις συσκευές οι οποίες είναι συνδεδεμένες στο σύστημα.
/etc	Το αρχείο ρυθμίσεων που μπορεί να έχει ένα πρόγραμμα βρίσκεται αποθηκευμένο σ' αυτό το φάκελο.

/home	Μέσα στο φάκελο βρίσκονται οι προσωπικοί φάκελοι των χρηστών του συστήματος.
/lib	Οι βιβλιοθήκες του συστήματος είναι αποθηκευμένες σ' αυτό το φάκελο.
/root	Είναι ο προσωπικός φάκελος του διαχειριστή του συστήματος (root).
/tmp	Σ' αυτό το φάκελο αποθηκεύονται προσωρινά αρχεία. Όλοι οι χρήστες του συστήματος έχουν δικαίωμα εγγραφής.

Απόλυτη διαδρομή (absolute path)

Η θέση κάθε ενός αρχείου στην ιεραρχική δομή αρχείων του Linux καθορίζεται από την απόλυτη διαδρομή του, η οποία είναι η διαδρομή η οποία ξεκινάει από τον ριζικό φάκελο και καταλήγει στο αρχείο. Ανάμεσα στους φακέλους εισάγουμε, σαν διαχωριστικό, το / (slash). Για παράδειγμα, η απόλυτη διαδρομή του αρχείου report.doc (στην παραπάνω εικόνα) είναι /home/its/ug1/ee51vn/report.doc.

Υπάρχει μία και μόνο μία απόλυτη διαδρομή για κάθε ένα αρχείο στο σύστημα.

Σχετική διαδρομή

Εκτός από την απόλυτη διαδρομή ενός αρχείου υπάρχει και η σχετική, η οποία είναι η διαδρομή η οποία σαν αρχή έχει κάποιον άλλον φάκελο εκτός του αρχικού. Αν ο τρέχοντας φάκελος είναι ο /ug1 τότε η σχετική διαδρομή του αρχείου report.doc είναι ee51vn/report.doc. **Παρατηρήστε ότι η διαδρομή δεν αρχίζει με /**. Ακόμα ένα παράδειγμα, αν ο /usr είναι ο τρέχοντας φάκελος η σχετική διαδρομή του αρχείου report.doc θα είναι .. /home/its/ug1/ee51vn/report.doc. **Στην αρχή εισάγουμε δύο τελείες για να «δούμε» τα περιεχόμενα του γονικού φακέλου στον οποίο βρίσκεται ο φάκελος stand**.

Υπάρχουν πολλές σχετικές διαδρομές για κάθε ένα αρχείο μέσα στο σύστημα, από τη στιγμή που η εκκίνηση μιας σχετικής διαδρομής είναι δική μας απόφαση.

Τύποι αρχείων

Τα περισσότερα αρχεία, είναι αρχεία τα οποία ονομάζουμε κανονικά αρχεία (regular files) και περιέχουν δεδομένα, όπως κείμενο, δυαδικά δεδομένα κτλ. Οι άλλοι τύπου αρχείων είναι οι παρακάτω:

Φάκελοι (Directories)	Αρχεία τα οποία περιέχουν άλλα αρχεία
Αρχεία συσκευών (device files)	Ειδικά αρχεία μέσω των οποίων έχουμε επικοινωνία με τις διάφορες συσκευές που είναι συνδεδεμένες στο σύστημα
Σύνδεσμοι (Links)	Αρχεία τα οποία δρουν σαν μια συντόμευση ενός άλλου αρχείου. Συνήθως δημιουργούμε ένα σύνδεσμο όταν προσπελαύνουμε πολύ συχνά ένα φάκελο ο οποίος έχει μεγάλη διαδρομή(path).
Sockets	Αρχεία τα οποία επιτρέπουν σε δύο διεργασίες να επικοινωνούν μέσω δικτύου μεταξύ τους.
Named pipes	Αρχεία τα οποία επιτρέπουν την επικοινωνία δύο διεργασιών.

Μπορούμε να διαπιστώσουμε τον τύπο κάθε ενός αρχείο με την εντολή:

```
user@localhost:~$ ls -l
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Desktop
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Documents
drwxr-xr-x 4 user user 4096 2009-01-01 00:00 Downloads
```

```
drwxr-xr-x 5 user user 4096 2009-01-01 00:00 Music
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Videos
-rwxr-xr-x 1 user user 196 2009-01-01 00:00 exercise1
-rwxr-xr-x 1 user user 8996 2009-01-01 00:00 exercise2
```

Ο πρώτος χαρακτήρας κάθε γραμμής, προσδιορίζει τον τύπο του αρχείου. Στον παρακάτω πίνακα μπορούμε να δούμε τις πιθανές τιμές που μπορεί να πάρει:

-	Κανονικό αρχείο
d	Φάκελος
l	Σύνδεσμος
c	Ειδικό αρχείο
s	Socket
p	Named pipe
b	Block device

Συμβάσεις στα ονόματα αρχείων

Στα ονόματα των αρχείων μπορούμε να χρησιμοποιήσουμε σχεδόν όλους τους χαρακτήρες με όριο τους 255. Γενικά όμως, δεν είναι καλή πρακτική να δημιουργούμε αρχεία με ονόματα που περιέχουν μεταχαρακτήρες (σχολιάζονται πιο κάτω), γιατί μπορεί να δημιουργηθούν προβλήματα.

Διαχείριση αρχείων

Γενικά, όλες οι εντολές που αφορούν αρχεία έχουν παρόμοιο τρόπο σύνταξης. Το βασικό είναι να γνωρίζουμε καλά να χειριζόμαστε τις διαδρομές (απόλυτες και σχετικές). Παρακάτω θα αναφερθούν οι βασικότερες εντολές διαχείρισης αρχείων και φακέλων του Linux.

Τρέχοντας φάκελος – pwd

Χρησιμοποιείται για την προβολή της απόλυτης διαδρομής του τρέχοντα φακέλου. Όπου τρέχοντας φάκελος, είναι ο φάκελος στον οποίο βρισκόμαστε την συγκεκριμένη χρονική στιγμή.

```
user@localhost:~$ pwd
/home/user
```

Περιεχόμενα φακέλου – ls

Χρησιμοποιείται για την προβολή των περιεχομένων ενός φακέλου. Προαιρετικά δέχεται μια παράμετρο η οποία είναι η διαδρομή του φακέλου τα περιεχόμενα του οποίου θέλουμε να δούμε. Αν δεν εισάγουμε καμία παράμετρο, τότε προβάλλονται τα περιεχόμενα του τρέχοντα φακέλου.

Η γενική μορφή της σύνταξης της εντολής είναι:

```
ls [επιλογές] [φάκελος]
```

Μερικές χρήσιμες επιλογές της εντολής ls είναι οι παρακάτω:

-l	Προβάλλονται όλες οι λεπτομέρειες (ιδιοκτήτης, δικαιώματα χρήσης, ημερομηνία τροποποίησης, μέγεθος κλπ) των αρχείων και των φακέλων
-a	Προβάλλονται και τα κρυφά αρχεία. Στο Linux όλα τα αρχεία των οποίων το όνομα αρχίζει από . είναι κρυφά
-t	Προβάλλονται τα αρχεία και οι φάκελοι ταξινομημένα κατά χρονο-

λογική σειρά

-S Προβάλλονται τα αρχεία και οι φάκελοι ταξινομημένα κατά μέγεθος

```
user@localost:~$ ls
lrwxrwxrwx 1 user user 10   2009-01-01   00:00   computers   ->
/etc/hosts
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Desktop
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Documents
drwxr-xr-x 4 user user 4096 2009-01-01 00:00 Downloads
drwxr-xr-x 5 user user 4096 2009-01-01 00:00 Music
-rwxr-xr-x 3 user user 0    2009-01-01 00:00 Videos
```

Πιο αναλυτικά το περιεχόμενο κάθε μιας στήλης είναι:

Δικαιώματα	**Αρ. Συνδέσμων**	**Ιδιοκτήτης**	**Ομάδα**	**Μέγεθος**	**Ημ. Τροποποίησης**	**Όνομα**
Τα δικαιώματα που έχουν οι χρήστες του συστήματος	Αριθμός συνδέσμων προς το αρχείο	Ο χρήστης ο οποίος είναι ιδιοκτήτης του αρχείου	Η ομάδα στην οποία ανήκει ο χρήστης	Το μέγεθος (σε bytes) του αρχείου	Η ημερομηνία και η ώρα της τελευταίας τροποποίησης του αρχείου	Το όνομα του αρχείου

Αλλαγή φακέλου – cd

Χρησιμοποιείται για την αλλαγή του τρέχοντα φακέλου. Αν δεν εισάγουμε καμία παράμετρο, το αποτέλεσμα θα είναι να μεταφερθούμε (δηλαδή, ο τρέχοντας φάκελος να είναι) στον προσωπικό μας φάκελο. Αν εισάγουμε σαν παράμετρο μια διαδρομή και αυτήν η διαδρομή υπάρχει, τότε θα μεταφερθούμε στο νέο φάκελο.

```
user@localhost:~$ cd /
user@localhost:/$
user@localhost:/$ cd /home/user
user@localhost:~$
user@localhost:/home/bob$ cd ../../usr/bin
user@localhost:/usr/bin$
```

Στο τελευταίο παράδειγμα, ενώ βρισκόμασταν στο φάκελο /home/bob εισάγοντας μια σχετική διαδρομή μεταφερθήκαμε στο φάκελο /usr/bin. Να σημειωθεί ότι οι δύο τελείες υποδηλώνουν το γονικό φάκελο.

Δημιουργία φακέλου – mkdir

Για τη δημιουργία ενός φακέλου χρησιμοποιούμε την εντολή mkdir, εισάγοντας σαν ορίσματα τα ονόματα (διαδρομές) των φακέλων που θέλουμε να δημιουργήσουμε. Η γενική της μορφή είναι:

`mkdir [επιλογές] φάκελος`

```
user@localost:~$ mkdir test1 test2 test3
user@localost:~$ ls
test1 test2 test3
```

Στο παραπάνω παράδειγμα δημιουργήθηκαν τρείς φάκελοι μέσα στον προσωπικό μας φάκελο.

```
user@localost:~/test1$ mkdir test3 ../test4
user@localost:~/test1$ ls
test3
user@localost:~/test1$ ls ..
test1 test2 test3 test4
```

Στο παραπάνω παράδειγμα δημιουργήθηκε ένας φάκελος μέσα στο φάκελο test1 με όνομα test3 και ένας φάκελος με όνομα test4 στον προσωπικό μας φάκελο.

Σημείωση: Ο δεύτερος φάκελος, στο παραπάνω παράδειγμα, δημιουργείται στον γονικό φάκελο του τρέχοντα φακέλου, αφού πριν από το όνομά του εισάγουμε δύο τελείες.

Χρησιμοποιώντας την επιλογή –p έχουμε τη δυνατότητα να δημιουργήσουμε φακέλους μέσα σε φακέλους οι οποίοι δεν υπάρχουν. Για παράδειγμα:

```
user@localost:~$ mkdir –p test5/test6/test7
user@localost:~$ ls
test1 test2 test3 test5
```

Θα δημιουργηθεί ο φάκελος test5, μέσα σ' αυτόν ο φάκελος test6 και μέσα σ' αυτόν ο test7. Αν δεν βάζαμε την επιλογή –p θα υπήρχε συντακτικό λάθος.

Διαγραφή φακέλου – rmdir

Η εντολή rmdir χρησιμοποιείται για τη διαγραφή μόνο κενών (χωρίς κανένα περιεχόμενο) φακέλων. Δέχεται ένα ή περισσότερα ορίσματα τα οποία αντιπροσωπεύουν φακέλους και η σύνταξή της είναι:

rmdir [επιλογές] φάκελος

```
user@localost:~$ rmdir test3
user@localost:~$ ls
test1 test2
```

Δημιουργία αρχείου – touch

Χρησιμοποιείται για τη δημιουργία κενών κανονικών αρχείων, ενώ η γενική μορφή της είναι:

touch [επιλογές] αρχείο

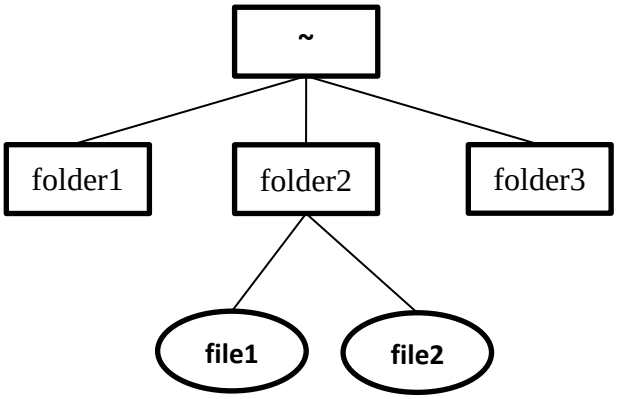
```
user@localost:~$ touch file1 file2 file3
user@localost:~$ ls
file1 file2 file3 test1 test2 test3 test5
```

Αντιγραφή αρχείου – cp

Όταν θέλουμε να αντιγράψουμε ένα αρχείο σε μια άλλη θέση του συστήματος αρχείων χρησιμοποιούμε την εντολή cp. Γενικά, η σύνταξη της είναι της μορφής:

cp [επιλογές] αρχείο_πηγή αρχείο_προορισμού

Αν βρισκόμαστε στον προσωπικό μας φάκελο, ο οποίος έχει την παρακάτω μορφή:



Αν θέλαμε να αντιγράψουμε το αρχείο file1 στο φάκελο folder3, θα γράφαμε:

```
user@localost:~$ cp folder2/file1 folder3
```

Αν ο τρέχοντας φάκελος είναι ο folder1 και θέλουμε να αντιγράψουμε μέσα του το αρχείο file2, θα γράφαμε:

```
user@localost:~$ cp ../folder2/file2 .
```

Σημείωση: Στο παραπάνω παράδειγμα ο προορισμός είναι ο τρέχοντας φάκελος, ο οποίος συμβολίζεται με την τελεία.

Για να αντιγράψουμε ολόκληρο φάκελο θα πρέπει να εισάγουμε την επιλογή –r. Για παράδειγμα, αν θέλουμε να αντιγράψουμε το φάκελο folder2 μέσα στο φάκελο folder3 με την προϋπόθεση ότι βρισκόμαστε στον προσωπικό μας φάκελο, θα γράφαμε:

```
user@localost:~$ cp –r folder2 folder3
```

Μετακίνηση - μετονομασία αρχείου – mv

Η εντολή για την μετακίνηση και την μετονομασία ενός αρχείου είναι η ίδια. Είναι η mv και η γενική της σύνταξη είναι της μορφής:

```
mv [επιλογές] αρχείο_πηγή αρχείο_προορισμού
```

Αν το αρχείο πηγής βρίσκεται στον ίδιο φάκελο με το αρχείο προορισμού, τότε γίνεται μετονομασία. Σε διαφορετική περίπτωση, γίνεται μετακίνηση. Χρησιμοποιώντας την πιο πάνω ιεραρχική δομή με τον προσωπικό φάκελο να είναι ο τρέχοντας:

```
user@localost:~$ mv folder1 f1
```

Η παραπάνω εντολή θα έχει σαν αποτέλεσμα την μετονομασία του φακέλου folder1 σε f1. Αν γράφαμε:

```
user@localost:~$ mv folder2 folder3
```

Το αποτέλεσμα θα ήταν να μετακινηθεί ολόκληρος ο φάκελος folder2 στο φάκελο folder3.

Διαγραφή αρχείου - rm

Τέλος, για τη διαγραφή αρχείων χρησιμοποιούμε την εντολή rm, η οποία συντάσσεται όπως:

```
rm [επιλογές] αρχείο
```

Αν θέλουμε να διαγράψουμε το αρχείο file1 που βρίσκεται στο φάκελο folder2 θα γράφαμε:

```
user@localost:~$ rm folder2/file1
```

Με αυτόν τον τρόπο, το αρχείο file1 διαγράφεται χωρίς τη δυνατότητα επαναφοράς του και χωρίς κάποιο προειδοποιητικό μήνυμα. Αν θέλουμε, μπορούμε να εισάγουμε την επιλογή –i πριν από διαγραφή αρχείου για την προβολή ενός μηνύματος προειδοποίησης.

```
user@localost:~$ rm –i folder2/file1
```

Σημείωση: Η διαγραφή αρχείων και φακέλων θα πρέπει να γίνεται με ιδιαίτερη προσοχή, γιατί δεν υπάρχει κάδος ανακύκλωσης (όπως στα Windows).

Για τη διαγραφή ολόκληρου φακέλου θα πρέπει να εισάγουμε την επιλογή –r.

```
user@localost:~$ rm –r folder2
```

Το αποτέλεσμα της παραπάνω εντολής θα είναι η διαγραφή του φακέλου folder2 (μαζί με τα περιεχόμενά του).

Περιεχόμενο αρχείων

Υπάρχουν αρκετές εντολές στο Linux οι οποίες μας επιτρέπουν να δούμε με διάφορες μορφές το περιεχόμενο των αρχείων κειμένου. Μερικές απ' αυτές είναι οι παρακάτω:

cat όνομα_αρχείου	Εμφανίζει όλα τα περιεχόμενα του αρχείου.
less όνομα_αρχείου	Εμφανίζει όλα τα περιεχόμενα του αρχείου, αλλά χωρισμένα σε σελίδες. Για να μεταβούμε στην επόμενη σελίδα πρέπει να πατήσουμε το πλήκτρο του κενού (spacebar), ενώ για να βγούμε από το περιβάλλον πρέπει να πατήσουμε το πλήκτρο q.
head [-n] όνομα_αρχείου	Προβάλλει τις δέκα πρώτες γραμμές του αρχείου. Αν εισάγουμε έναν αριθμό, σαν επιλογή, τότε θα εμφανιστεί ο συγκεκριμένος αριθμός γραμμών από την αρχή του αρχείου.
tail [-n] όνομα_αρχείου	Προβάλλει τις δέκα τελευταίες γραμμές του αρχείου. Αν εισάγουμε έναν αριθμό, σαν επιλογή, τότε θα εμφανιστεί ο συγκεκριμένος αριθμός γραμμών από το τέλος του αρχείου.

wc [επιλογές] όνομα_αρχείου	Κάνει καταμέτρηση των γραμμών, των λέξεων και των χαρακτήρων από το αρχείο εισόδου.
	`user@localost:~$ wc /etc/hosts`
	`23 77 683 /etc/hosts`

Αναζήτηση αρχείων

Αν θέλουμε να αναζητήσουμε ένα αρχείο χρησιμοποιούμε την εντολή find, η οποία συντάσσεται όπως παρακάτω:

`find επιλογές όνομα_αρχείου`

```
user@localost:~$ find . –name test*
```

Στο παραπάνω παράδειγμα, κάνουμε αναζήτηση στον τρέχοντα φάκελο και σε όλους τους υποφακέλους του, για τα αρχεία που το όνομά τους αρχίζει από test και τελειώνει σε οτιδήποτε. Γενικά, η αναζήτηση ξεκινάει από το φάκελο που θα δώσουμε σαν επιλογή στη find και εκτείνεται σε όλους τους υποφακέλους που πιθανόν έχει ο αρχικός φάκελος.

> Σημείωση: Αν θέλουμε να αναζητήσουμε ένα αρχείο σε όλο το σύστημα θα πρέπει να εισάγουμε σαν πρώτο όρισμα το /. Δηλαδή, find / -name test*

Η παραπάνω αναζήτηση γίνεται με βάση το όνομα ενός αρχείου. Έχουμε τη δυνατότητα να κάνουμε αναζήτηση με βάση τον τύπο του αρχείου. Για παράδειγμα, μπορούμε να αναζητήσουμε μόνο αρχεία ή μόνο φακέλους.

```
user@localost:~$ find . –type f
```

Στο παραπάνω παράδειγμα, κάνουμε αναζήτηση μέσα στον τρέχοντα φάκελο για όλα τα κανονικά αρχεία. Αν θέλαμε η αναζήτηση να περιοριστεί μόνο στους φακέλους, αντί για f στο τέλος θα εισάγαμε το γράμμα d.

Συμβολικοί σύνδεσμοι

Ένα πολύ χρήσιμο χαρακτηριστικό είναι οι συμβολικοί σύνδεσμοι, των οποίων η λειτουργία είναι παρόμοια με των συντομεύσεων από τον κόσμο των Windows. Υπάρχουν δύο είδη συνδέσμων, οι συμβολικοί και σκληροί (hard). Εμείς θα ασχοληθούμε μόνο με τους συμβολικούς.

Για τη δημιουργία ενός συμβολικού συνδέσμου χρησιμοποιούμε την εντολή ln με την επιλογή –s. Για παράδειγμα, η παρακάτω εντολή θα δημιουργήσει ένα συμβολικό σύνδεσμο, στον τρέχοντα φάκελο, του αρχείου /etc/hosts με όνομα computers.

```
user@localost:~$ ln -s /etc/hosts computers
user@localost:~$ ls
lrwxrwxrwx 1 user user 10   2009-01-01   00:00   computers   ->
/etc/hosts
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Desktop
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Documents
drwxr-xr-x 4 user user 4096 2009-01-01 00:00 Downloads
drwxr-xr-x 5 user user 4096 2009-01-01 00:00 Music
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Videos
-rwxr-xr-x 1 user user 196 2009-01-01 00:00 exercise1
-rwxr-xr-x 1 user user 8996 2009-01-01 00:00 exercise2
```

Μεταχαρακτήρες

Μερικές φορές υπάρχει η ανάγκη να κάνουμε με εργασία με ένα σύνολο αρχείων τα οποία έχουν κάποια ομοιότητα στο όνομά τους. Για παράδειγμα, η διαγραφή όλων των αρχείων που το όνομά τους αρχίζει από f. Για αυτές τις περιπτώσεις είναι απαραίτητοι οι μεταχαρακτήρες.

Τους μεταχαρακτήρες, μπορούμε να τους χρησιμοποιήσουμε με όλες τις εντολές που αφορούν στη διαχείριση αρχείων.

Μεταχαρακτήρας ?

Αντικαθιστά έναν οποιονδήποτε χαρακτήρα, οπουδήποτε εμφανίζεται σε ένα όνομα αρχείου. Για παράδειγμα, η παρακάτω εντολή θα διαγράψει όλα τα αρχεία που το όνομά τους αρχίζει από f και αποτελείται από δύο χαρακτήρες.

```
user@localost:~$ rm f?
```

Στο παρακάτω παράδειγμα, θα διαγραφούν όλα τα αρχεία που το όνομά τους αρχίζει από file ακολουθούμενο από δύο οποιουσδήποτε χαρακτήρες και τελειώνει σε .txt.

```
user@localost:~$ rm file??.txt
```

Μεταχαρακτήρας *

Αντικαθιστά οσουσδήποτε και οποιουσδήποτε χαρακτήρες στο σημείο της διαδρομής του αρχείου που εμφανίζεται. Για παράδειγμα, αν θέλουμε να αντιγράψουμε όλα τα αρχεία που αρχίζουν από file και τελειώνουν σε οτιδήποτε στο φάκελο folder2 θα γράφαμε:

```
user@localost:~$ cp file* folder2
```

Επίσης, αν θέλουμε να αντιγράψουμε όλα τα αρχεία που τελειώνουν σε cat στο φάκελο folder3 θα γράφαμε:

```
user@localost:~$ cp *cat folder3
```

Μεταχαρακτήρες [,],-,!

Τις αγκύλες τις χρησιμοποιήσουμε για να ταιριάξουμε έναν συγκεκριμένο χαρακτήρα από αυτούς που βρίσκονται μέσα σε αυτές. Δηλαδή:

```
user@localost:~$ cp cat[1234] folder3
```

Η παραπάνω εντολή θα έχει σαν αποτέλεσμα να αντιγράψει όλα τα αρχεία που ξεκινούν από cat και καταλήγουν σε 1 ή 2 ή 3 ή 4 στο φάκελο folder3.

Υπάρχει η δυνατότητα να ορίσουμε περιοχές χαρακτήρων (σύμφωνα με τη διάταξη ASCII) μέσα στις αγκύλες. Για παράδειγμα:

```
user@localost:~$ cp cat[A-K] folder3
```

Όλα τα αρχεία που αρχίζουν από cat και καταλήγουν σε κάποιο από τα γράμματα από το A μέχρι το K θα ταιριάξουν με την παραπάνω έκφραση και θα γίνει η αντιγραφή του στο φάκελο folder3.

Τέλος, με τον μεταχαρακτήρα ! εισάγουμε άρνηση, με αποτέλεσμα ότι βάζουμε μέσα στην αγκύλη να μην περιλαμβάνεται. Έτσι:

```
user@localost:~$ ls cat[!1234]
```

Η παραπάνω εντολή θα εμφανίσει όλα τα αρχεία των οποίων το όνομα αρχίζει από cat και δεν τελειώνει ούτε σε 1, ούτε σε 2, ούτε σε 3 και ούτε σε 4.

Μεταχαρακτήρας \

Ονομάζεται χαρακτήρας διαφυγής (escape character) και ο σκοπός του είναι να ερμηνεύει κυριολεκτικά τον χαρακτήρα που ακολουθεί, αφαιρώντας οποιαδήποτε ειδική σημασία. Χαρακτήρες με ειδική σημασία είναι το ?, το * και άλλοι που είδαμε πιο πάνω. Για παράδειγμα:

```
user@localost:~$ ls
file? file1 file2 test
user@localost:~$ rm file?
user@localost:~$ ls
test
```

Μετά την εντολή rm file? διαγράφηκαν όλα τα αρχεία που το όνομά τους άρχιζε από file και ακολουθούσε ένας μόνο χαρακτήρας. Αν ο στόχος ήταν να διαγράψουμε μόνο το αρχείο file?, τότε θα έπρεπε να γράψουμε:

```
user@localost:~$ rm file\?
user@localost:~$ ls
file1 file2 test
```

Δικαιώματα χρήσης

Στο Linux, υπάρχουν διάφοροι τύποι λογαριασμών χρηστών, ανάλογα με τα δικαιώματα που έχουν στο σύστημα. Ο χρήστης που έχει όλα τα δικαιώματα να είναι ο διαχειριστής του συστήματος και συνήθως έχει όνομα root. Υπάρχουν αρκετοί χρήστες οι οποίοι χρησιμοποιούνται για την εκτέλεση διεργασιών του συστήματος και ονομάζονται χρήστες συστήματος. Τέλος, υπάρχουν οι απλοί χρήστες, των οποίων τα δικαιώματα χρήσης περιορίζονται στον προσωπικό τους φάκελο.

Επίσης, ο διαχειριστής του συστήματος μπορεί να κατανείμει τους χρήστες σε ομάδες για την καλύτερη διαχείριση τους.

Τα δικαιώματα χρήσης που έχει κάθε ένας χρήστης ή μέλος μιας ομάδας σε κάθε ένα αρχείο μπορούμε να τα δούμε στην πρώτη στήλη του αποτελέσματος της εντολής ls –l.

```
user@localost:~$ ls
lrwxrwxrwx 1 user user 10   2009-01-01   00:00   computers   ->
/etc/hosts
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Desktop
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Documents
drwxr-xr-x 4 user user 4096 2009-01-01 00:00 Downloads
drwxr-xr-x 5 user user 4096 2009-01-01 00:00 Music
-rwxr-xr-x 3 user user 0    2009-01-01 00:00 Videos
```

Άδειες πρόσβασης

Σε κάθε ένα αρχείο υπάρχει ένα σύνολο από άδειες πρόσβασης οι οποίες είναι οι:

- Ανάγνωση (read) – συμβολίζεται με το γράμμα r
- Εγγραφή (write) – συμβολίζεται με το γράμμα w
- Εκτέλεση (execution) – συμβολίζεται με το γράμμα x

Όποτε υπάρχει το κάποιος από τους συγκεκριμένους χαρακτήρες, σημαίνει ότι η άδεια είναι ενεργή, ενώ όταν υπάρχει παύλα σημαίνει ότι η αδεία δεν είναι ενεργή.

Όπως παρατηρούμε από το παραπάνω αποτέλεσμα της εντολής ls, υπάρχουν τρεις τριάδες αδειών πρόσβασης οι οποίες αναφέρονται με τη σειρά στον ιδιοκτήτη του αρχείου, στα μέλη της ομάδας στην οποία ανήκει ο ιδιοκτήτης και τέλος σε όλους τους υπόλοιπους. Να υπενθυμίσουμε ότι ο ιδιοκτήτης εμφανίζεται στην τρίτη στήλη, ενώ η ομάδα του ιδιοκτήτη στην τέταρτη.

Συνοπτικά, η ερμηνεία που έχουν οι άδειες χρήσης ανάλογα με τον τύπου του αρχείου είναι οι παρακάτω:

	Ανάγνωση (read) - r	**Εγγραφή (write) - w**	**Εκτέλεση (execution) - x**
Κανονικό αρχείο	Ανάγνωση	Αλλαγή περιεχομένων, διαγραφή	Εκτέλεση
Φάκελο	Εμφάνιση περιεχομένων	Προσθήκη, διαγραφή, μετονομασία αρχείων	Είσοδος στο φάκελο, πρόσβαση στα αρχεία

Τροποποίηση των δικαιωμάτων χρήσης

Μόνο ο ιδιοκτήτης του αρχείου έχει τη δυνατότητα τροποποίησης των δικαιωμάτων χρήσης ενός αρχείου. Η τροποποίηση γίνεται με την εντολή chmod της οποίας η γενική μορφή σύνταξης είναι:

```
chmod συμβολικά_δικαιώματα αρχείο
```

Στη θέση συμβολικά_δικαιώματα εισάγουμε:

- Κατηγορία ιδιοκτήτη, με πιθανές τιμές

- u, για τον ιδιοκτήτη του αρχείου
- g, για τα μέλη της ομάδας του ιδιοκτήτη
- o, για όλους τους άλλους χρήστες
- a, για όλους τους χρήστες

- Τελεστή, με πιθανές τιμές
  - =, για ακριβή προσδιορισμό άδειας πρόσβασης
  - -, για αφαίρεση άδειας πρόσβασης
  - +, για προσθήκη άδειας πρόσβασης
- Άδεια πρόσβασης, με πιθανές τιμές
  - r, για ανάγνωση
  - w, για εγγραφή
  - x, για εκτέλεση

```
user@localost:~$ ls
lrwxrwxrwx 1 user user 10   2009-01-01   00:00   computers   ->
/etc/hosts
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Desktop
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Documents
drwxr-xr-x 4 user user 4096 2009-01-01 00:00 Downloads
drwxr-xr-x 5 user user 4096 2009-01-01 00:00 Music
-rwxr-xr-x 3 user user 0    2009-01-01 00:00 Videos
user@localost:~$ chmod u-wx Videos
user@localost:~$ ls
lrwxrwxrwx 1 user user 10   2009-01-01   00:00   computers   ->
/etc/hosts
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Desktop
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Documents
drwxr-xr-x 4 user user 4096 2009-01-01 00:00 Downloads
drwxr-xr-x 5 user user 4096 2009-01-01 00:00 Music
-r--r-xr-x 3 user user 0    2009-01-01 00:00 Videos
```

Με την παραπάνω εντολή αφαιρέσαμε τις άδειες για εγγραφή και εκτέλεση από το αρχείο Videos για τον ιδιοκτήτη του αρχείου.

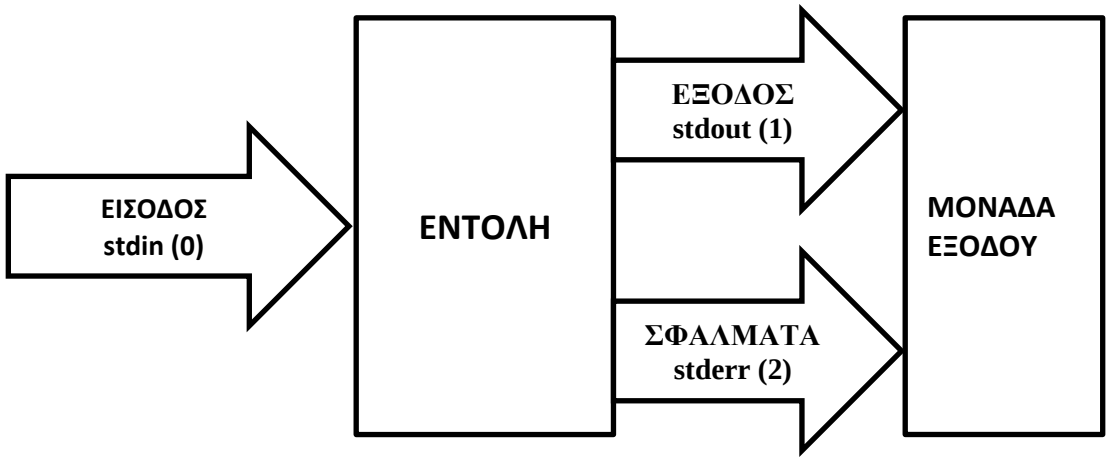
```
user@localost:~$ chmod go=r Videos
user@localost:~$ ls
lrwxrwxrwx 1 user user 10   2009-01-01   00:00   computers   ->
/etc/hosts
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Desktop
drwxr-xr-x 3 user user 4096 2009-01-01 00:00 Documents
drwxr-xr-x 4 user user 4096 2009-01-01 00:00 Downloads
drwxr-xr-x 5 user user 4096 2009-01-01 00:00 Music
-r--r--r-- 3 user user 0    2009-01-01 00:00 Videos
```

Η παραπάνω εντολή προσδιορίζει με ακρίβεια τα δικαιώματα (μόνο ανάγνωση) που επιθυμούμε να έχουν τα μέλη της ομάδας του ιδιοκτήτη και οι υπόλοιποι.

Ανακατεύθυνση – Σωλήνωση

Σε ένα σύστημα Linux, η πληκτρολόγηση μιας εντολής συνοδεύεται συνήθως από μια είσοδο (καθιερωμένη είσοδος – stdin) και αφού πατήσουμε το πλήκτρο Enter περιμένουμε κάποιο αποτέλεσμα. Η προκαθορισμένη μονάδα εξόδου για τα αποτελέσματα των εντολών είναι η οθόνη και ονομάζε-

ται καθιερωμένη έξοδος (standard output – stdout). Δηλαδή, το αποτέλεσμα μιας εντολής, η οποία έχει εκτελεστεί με επιτυχία, κατευθύνεται μέσω μιας ροής δεδομένων (ο κωδικός της είναι το 1) στην οθόνη. Η εκτέλεση μιας εντολής όμως, μπορεί να έχει σαν αποτέλεσμα κάποιο σφάλμα. Τα σφάλματα προβάλλονται και αυτά στην οθόνη, αλλά μέσω μιας άλλης ροής δεδομένων η οποία ονομάζεται καθιερωμένη έξοδος σφαλμάτων (standard error – stderr), με κωδικό 2.



Ανακατεύθυνση

Με την ανακατεύθυνση (redirection) μπορούμε να στείλουμε τα αποτελέσματα της καθιερωμένης εξόδου ή της εξόδου σφαλμάτων ή και των δύο, σε ένα αρχείο.

Για να ανακατευθύνουμε την καθιερωμένη έξοδο σε ένα αρχείο, χρησιμοποιούμε τους χαρακτήρες > και >>. Δηλαδή, αν θέλουμε να γράψουμε το αποτέλεσμα μιας εντολής σε ένα αρχείο θα γράφαμε:

```
$ who > connected_users
```

Με την παραπάνω εντολή στέλνουμε το αποτέλεσμα της εντολής who στο αρχείο connected_users. Αν το αρχείο δεν υπάρχει, θα δημιουργηθεί αυτόματα. Αν το αρχείο υπάρχει, τότε θα διαγραφεί το παλιό περιεχόμενο του αρχείου και θα εγγραφεί το αποτέλεσμα της εντολής.

```
$ cat connected_users
user pts/0 2010-01-01 00:00 (192.168.1.1)
```

Αν θέλουμε να προσθέσουμε το αποτέλεσμα μιας εντολής στο τέλος ενός αρχείου, θα πρέπει να χρησιμοποιήσουμε το σύμβολο >>.

```
$ date > connected_users
$ cat connected_users
user pts/0 2010-01-01 00:00 (192.168.1.1)
Mon Jan 01 00:00:00 EET 2010
```

Για ανακατεύθυνση της εξόδου σφαλμάτων χρησιμοποιούμε τον συμβολισμό 2>. Δηλαδή, αν θέλουμε να αποθηκεύσουμε σε ένα αρχείου το σφάλμα που μπορεί να προκύψει από μια εντολή θα γράφαμε:

```
$ ls
folder1 folder2 folder4
$ ls folder1 folder2 folder3
ls: folder3: No such file or directory
folder1 folder2
$ ls folder1 folder2 folder3 2> errors
folder1 folder2
$ cat errors
ls: folder3: No such file or directory
```

Έχουμε τη δυνατότητα να ανακατευθύνουμε ταυτόχρονα την καθιερωμένη έξοδο και την έξοδο σφαλμάτων σε δύο αρχεία. Για την ανακατεύθυνση της καθιερωμένης εξόδου χρησιμοποιούμε το 1> και την ανακατεύθυνση της εξόδου σφαλμάτων το 2>. Για παράδειγμα:

```
$ ls folder1 folder2 folder3 1> folders 2> errors
$ cat folders
folder1 folder2
$ cat errors
ls: folder3: No such file or directory
```

Με την πρώτη εντολή του παραδείγματος, στέλνουμε το αποτέλεσμα της εντολής στο αρχείο folders και τα πιθανά σφάλματα στο αρχείο errors.

Όπως και πριν, μπορούμε να χρησιμοποιήσουμε τους συμβολισμούς 1>> και 2>> για προσθήκη στα αρχεία.

Σωλήνωση

Η αποστολή του αποτελέσματος μιας εντολής όχι στην καθιερωμένη έξοδο, αλλά σαν είσοδο σε μια άλλη εντολή, ονομάζεται σωλήνωση (pipeline). Ο συμβολισμός που χρησιμοποιούμε είναι το |.

```
$ ls
folder1 folder2 folder4
$ ls -l | wc -l
4
```

Στη δεύτερη εντολή του παραδείγματος, στέλνουμε το αποτέλεσμα της εντολής ls -l σαν είσοδο στην εντολή wc -l. Με αυτόν τον τρόπο μετράμε το πλήθος των γραμμών, άρα και των αρχείων και φακέλων, του τρέχοντα φακέλου.

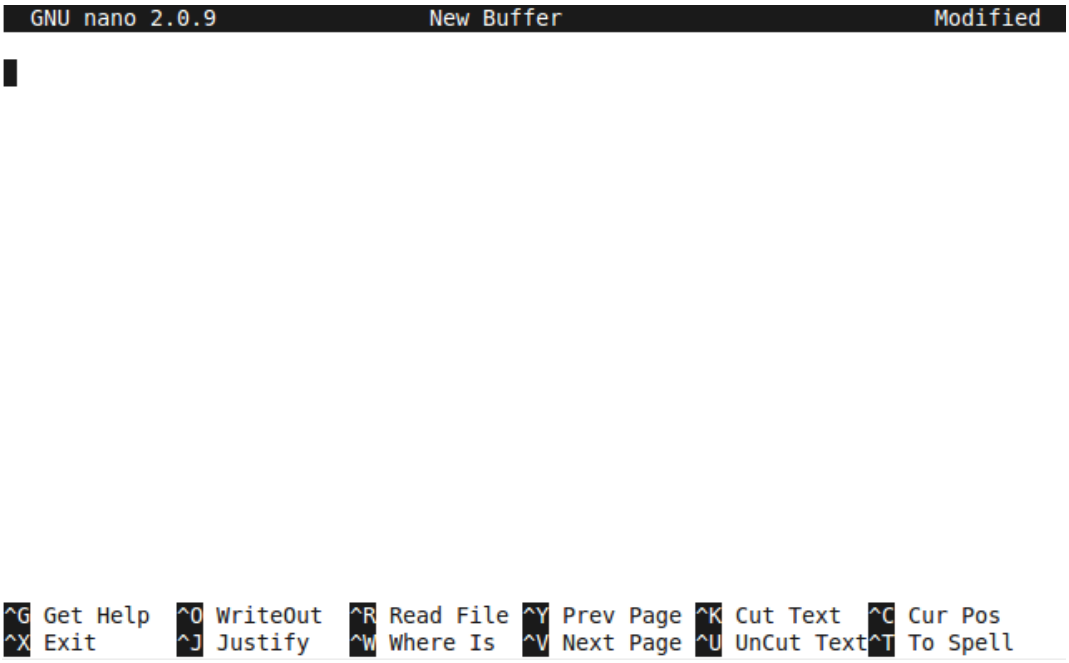
Εργασία με αρχεία κειμένου

Η εργασία με αρχεία κειμένου είναι πολύ σημαντική, γιατί η πλειοψηφία των αρχείων ρυθμίσεων και άλλες πολλές εργασίες πραγματοποιούνται με τέτοιου τύπου αρχεία. Θα δούμε λοιπόν τον τρόπο με τον οποίο χρησιμοποιούμε:

- Έναν επεξεργαστή κειμένου σε περιβάλλον κειμένου
- Εντολές που φιλτράρουν τα περιεχόμενα των αρχείων κειμένου
- Εντολές ταξινόμησης των περιεχομένων των αρχείων κειμένου

Ο επεξεργαστής κειμένου nano

Ο επεξεργαστής κειμένου nano παρότι εκτελείται σε περιβάλλον κειμένου έχει αρκετές δυνατότητες. Επίσης, βρίσκεται προεγκατεστημένος σχεδόν σε όλες τις διανομές.



Οι περισσότερες εντολές του nano εκτελούνται έχοντας πατημένο το πλήκτρο Ctrl και πατώντας μία φορά κάποιο άλλο πλήκτρο. Για παράδειγμα, ^X σημαίνει ότι έχοντας πατημένο το πλήκτρο Ctrl πατάμε μία φορά το πλήκτρο X.

Μερικές από τις πιο χρήσιμες εντολές του nano είναι οι παρακάτω:

Ctrl+G Προβολή της βοήθειας του nano.

Ctrl+R Εισαγωγή ενός αρχείου κειμένου στη θέση του δρομέα. Αφού έχουμε πατήσει τον συνδυασμό των πλήκτρων Ctrl+R, για την επιλογή του αρχείου που θέλουμε να εισάγουμε πατάμε Ctrl+T.

Alt+A Έναρξη επιλογής κειμένου και στη συνέχεια τα βελάκια (δεξιά ή αριστερά).

Alt+6 Αντιγραφή κειμένου στη μνήμη.

Ctrl+K Αποκοπή της γραμμής που βρίσκεται ο δρομέας.

Ctrl+U Επικόλληση του περιεχομένου της μνήμης στη θέση που βρίσκεται ο δρομέας.

Ctrl+O Αποθήκευση ή/και αποθήκευση ως. Μετά το πάτημα του συνδυασμού εισάγουμε το όνομα του αρχείου.

Ctrl+X Αποθήκευση των αλλαγών και έξοδος από τον nano.

Ταξινόμηση με την sort

Όταν θέλουμε να ταξινομήσουμε το περιεχόμενο ενός αρχείου κειμένου ή την έξοδο μιας εντολής χρησιμοποιούμε την εντολή sort. Η γενική της σύνταξη είναι:

```
sort [επιλογές] αρχείο
```

Μερικές χρήσιμες επιλογές της εντολής είναι οι παρακάτω:

- -n: Ταξινόμηση αριθμών.
- -kn: Όπου n είναι ένας αριθμός που προσδιορίζει τη στήλη με βάση την οποία θέλουμε να γίνει η ταξινόμηση.

Για παράδειγμα αν έχουμε το αρχείο towns με περιεχόμενα τα παρακάτω:
Kastoria
Thessaloniki
Athens
Patra
Larisa
Γράφοντας την παρακάτω εντολή θα γίνει ταξινόμηση των περιεχομένων του αρχείου towns

```
user@localhost:~ $ sort towns
Athens
Kastoria
Larisa
Patra
Thessaloniki
```

Αν θέλουμε να ταξινομήσουμε τα αρχεία ενός φακέλου με βάση το μέγεθός του θα γράφαμε:

```
user@localhost:~ $ -ls -l | sort -n -k5
```

Την παράμετρο n την εισάγουμε, γιατί θέλουμε να γίνει ταξινόμηση αριθμών, ενώ με την παράμετρο k ορίζουμε την στήλη με βάση την οποία θα γίνει η ταξινόμηση.

Αναζήτηση σε αρχείο με την grep

Η εντολή grep μας επιτρέπει να κάνουμε αναζητήσεις μέσα σε ένα αρχείο ή να τη χρησιμοποιούμε στο δεύτερο μέρος μιας σωλήνωσης για να φιλτράρουμε την έξοδο του πρώτου μέρους. Για παράδειγμα, η παρακάτω εντολή αναζητά για την συμβολοσειρά sa αρχείο towns.

```
user@localhost:~ $ grep sa towns
Larisa
Thessaloniki
```

Η αναζήτηση πραγματοποιείται σε όλο το μήκος και πλάτος του αρχείου και επιστρέφεται ολόκληρη η γραμμή/ες στην οποία βρέθηκε η συμβολοσειρά αναζήτησης. Μπορούμε όμως να αναγκάσουμε την grep να ψάχνει στην αρχή ή στο τέλος κάθε γραμμής. Αυτό γίνεται με την εισαγωγή δύο ειδικών χαρακτήρων, του ^ όταν θέλουμε η αναζήτηση να γίνει στην αρχή της γραμμής και του $ όταν θέλουμε η αναζήτηση να γίνει στο τέλος της γραμμής. Για παράδειγμα, αν θέλουμε να δούμε όλους τους χρήστες του συστήματος που το όνομα τους αρχίζει από s θα πρέπει να γράψουμε:

```
user@localhost:~ $ grep ^s /etc/passwd
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
syslog:x:101:102::/home/syslog:/bin/false
```

```
saned:x:110:116::/home/saned:/bin/false
```

Για να δούμε τους χρήστες που έχουν σαν προκαθορισμένο φλοιό τον bash θα πρέπει να γράψουμε:

```
user@localhost:~ $ grep bash$ /etc/passwd
root:x:0:0:root:/root:/bin/bash
theo:x:1000:1000:Theo Kordos,,,:/home/theo:/bin/bash
```

Χρησιμοποιώντας την επιλογή –v έχουμε τη δυνατότητα να αντιστρέψουμε το αποτέλεσμα. Δηλαδή, να εμφανίσουμε όλες τις γραμμές οι οποίες δεν περιέχουν το αλφαριθμητικό αναζήτησης.

Η γλώσσα awk

Η γλώσσα προγραμματισμού awk είναι σχεδιασμένη για να βρίσκει κάποιο αλφαριθμητικό σε ένα αρχείο και να κάνει κάποιες ενέργειες σ' αυτό. Το μεγάλο όπλο της awk είναι ότι επεξεργάζεται το αρχείο σαν να είναι χωρισμένο σε πεδία και εμείς θα χρησιμοποιήσουμε μόνο αυτό το χαρακτηριστικό της.

Η γενική σύνταξη της εντολή είναι η παρακάτω:

awk [επιλογές] αρχείο

Θεωρούμε ότι υπάρχει ένα αρχείο με όνομα cities με το παρακάτω περιεχόμενο:

Athens 5000000 Attikis
Kastoria 15000 Kastorias
Larisa 300000 Larisas
Patra 300000 Achaias
Thessaloniki 900000 Thessalonikis

Για να κάνουμε μία απλή αναζήτηση της λέξης Larisa στο αρχείο των προηγούμενων παραδειγμάτων, θα πρέπει να γράψουμε:

```
user@localhost:~ $ awk '/Larisa/' cities
Larisa 300000    Larisas
```

Η λέξη αναζήτησης τοποθετείται ανάμεσα σε χαρακτήρες / και όλη η παράσταση μέσα σε μονά εισαγωγικά. Οι χαρακτήρες για αναζήτηση στο τέλος ή στην αρχή της κάθε γραμμής ισχύουν όπως ακριβώς και στην περίπτωση της grep.

Αν θέλουμε να αναζητήσουμε όλες τις γραμμές οι οποίες τελειώνουν σε as θα γράφαμε:

```
user@localhost:~ $ awk '/as$/' cities
Kastoria 15000   Kastorias
Larisa 300000    Larisas
Patra 300000     Achaias
```

Η awk χωρίζει το περιεχόμενο ενός αρχείου σε πεδία χρησιμοποιώντας έναν χαρακτήρα διαχωρισμού. Ο προκαθορισμένος χαρακτήρας διαχωρισμού των πεδίων είναι το κενό διάστημα ή το TAB.

Στο αρχείο του παραδείγματός μας, παρατηρούμε ότι τα περιεχόμενα κάθε μίας γραμμής είναι χωρισμένα με κενό. Έτσι, μέσω της awk μπορούμε να τα χειριστούμε σαν πεδία. Για παράδειγμα, αν θέλουμε να εμφανίσουμε το πρώτο πεδίο όλων γραμμών του αρχείου θα γράφαμε:

```
user@localhost:~ $ awk '{print $1}' cities
Athens
Kastoria
Larisa
Patra
Thessaloniki
```

Χρησιμοποιούμε την εντολή print και δίνουμε σαν παράμετρο τον αύξοντα αριθμό της στήλης (πεδίου) που θέλουμε να εκτυπώσουμε.

Μπορούμε να κάνουμε ταυτόχρονα αναζήτηση μιας συμβολοσειράς και να εκτυπώσουμε κάποια/κάποιες στήλη/ες.

```
user@localhost:~ $ awk '/Athens/{print $1,$2}' cities
Athens 5000000
```

Στο παραπάνω παράδειγμα, κάναμε αναζήτηση της λέξης Athens και από τη γραμμή η οποία περιέχει αυτή την λέξη εκτυπώσαμε μόνο την πρώτη και τη δεύτερη στήλη.

Έχουμε τη δυνατότητα να ορίσουμε τον δικό μας χαρακτήρα διαχωρισμού των πεδίων αν τα περιεχόμενα του αρχείου δεν είναι χωρισμένα με αυτόν. Για να πετύχουμε αλλαγή του χαρακτήρα διαχωρισμού εισάγουμε την επιλογή -F. Έτσι, αν ο χαρακτήρας διαχωρισμού είναι ο : τότε θα πρέπει να γράψουμε:

```
user@localhost:~ $ awk -F: '/Athens/{print $1,$2}' cities
```

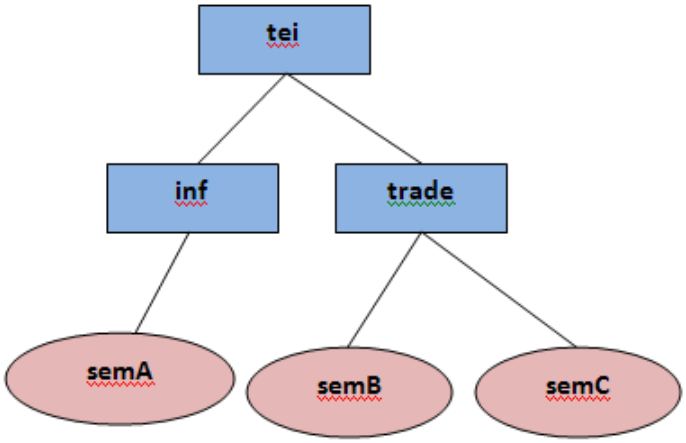
Τέλος, έχουμε τη δυνατότητα να κάνουμε πιο σύνθετες αναζητήσεις χρησιμοποιώντας αριθμητικούς τελεστές στο περιεχόμενο των πεδίων ενός αρχείου. Αν θέλαμε να εμφανίσουμε όλες τις εγγραφές οι οποίες έχουν το δεύτερο πεδίο τους μεγαλύτερο από 500000, θα γράφαμε:

```
user@localhost:~ $ awk '{if ($2 > 500000) print $0}' cities
Athens 5000000   Attikis
Thessaloniki 900000   Thessalonikis
```

Χρησιμοποιήσαμε τον τελεστή if και μέσα σε παρένθεση γράψαμε την συνθήκη που θέλουμε να ισχύει. Τέλος, γράφοντας print $0 τυπώνουμε όλα τα πεδία.

Ασκήσεις

1. Δημιουργήστε ένα κενό κανονικό αρχείο με όνομα file1.
2. Εμφανίστε την απόλυτη διαδρομή του προσωπικού σας φακέλου.
3. Να κάνετε τρέχοντα φάκελο τον φάκελο /etc.
4. Μετακινηθείτε στον προσωπικό σας φάκελο.
5. Δημιουργείστε τρία κενά αρχεία με ονόματα testfile*, testfile*1, testfile*2 στον προσωπικό σας φάκελο.
6. Δημιουργείστε την παρακάτω δενδρική δομή φακέλων και αρχείων μέσα στον προσωπικό σας φάκελο.



7. Να μετακινηθείτε στον φάκελο trade.
8. Να αντιγράψετε το αρχείο semA στον προσωπικό σας φάκελο.
9. Να μετονομάσετε το αρχείο semA, που βρίσκεται στον φάκελο inf, σε semester.
10. Να μετακινήσετε τα αρχεία semB και semC στον προσωπικό σας φάκελο.
11. Να διαγράψετε τα αρχεία semA, semB, semC που βρίσκονται στον προσωπικό σας φάκελο.
12. Να αντιγράψετε στον προσωπικό σας φάκελο, όλα τα αρχεία που το όνομά τους αρχίζει από i και τελειώνει σε .conf τα οποία βρίσκονται στον φάκελο /etc.
13. Να μετακινήσετε τα παραπάνω αρχεία στον φάκελο /tmp.
14. Να διαγράψετε το αρχείο test_file*.
15. Να προβάλλετε στην οθόνη σας όλους τους φακέλους που βρίσκονται στο φάκελο /etc.
16. Να δημιουργήσετε τέσσερα αρχεία με ονόματα file1, file2, file3 και file4 στον προσωπικό σας φάκελο.
17. Να δώσετε όλες τις άδειες πρόσβασης στα αρχεία file1 και file2 σε όλες τις κατηγορίες χρηστών.
18. Στα αρχεία file3 και file4 μόνο ο ιδιοκτήτης να έχει τις άδειες ανάγνωσης και εγγραφής. Οι υπόλοιποι να μην έχουν καμία άδεια.
19. Να αφαιρέσετε τις άδειες ανάγνωσης και εγγραφής από το φάκελο inf.
20. Να αντιγράψετε τα αρχεία file1 και file2 στο φάκελο inf.
21. Να αντιγράψετε όλα τα κρυφά αρχεία του προσωπικού σας φακέλου στο φάκελο inf.
22. Να διαγράψετε το φάκελο inf.
23. Υποθέτουμε ότι σε ένα φάκελο υπάρχουν τα αρχεία a??*, aa, bb, a1, a2, a1.1, b1 και b2. Επίσης, υπάρχουν και οι φάκελοι folder1 και folder2. Ποιο θα είναι το αποτέλεσμα των παρακάτω εντολών (η κάθε μία εντολή εκτελείται ξεχωριστά και δεν επηρεάζει τις υπόλοιπες):
    - cp a[1-2] folder1
    - ls *[1-2]

- mv *[!0-9] ~
- cp [a-zA-Z][!a-zA-Z]* folder1
- mv a??* folder2

24. Να αναζητήσετε το αρχείο date
25. Να αναζητήσετε το αρχείο free που βρίσκεται σε κάποιον υποφάκελο του φακέλου /usr.
26. Να δημιουργήσετε ένα συμβολικό σύνδεσμο στον προσωπικό σας φάκελο με όνομα share ο οποίος να δείχνει στο φάκελο /tmp.
27. Να βρείτε το πλήθος των αρχείων του φακέλου /etc.
28. Με ποια εντολή θα εμφανίζατε, από ένα αρχείο κειμένου, της γραμμές από την 5 μέχρι και την 10;
29. Να αποθηκεύσετε σε αρχείο με όνομα folders, όλους τους φακέλους του προσωπικού σας φακέλου στους οποίους είστε ιδιοκτήτης.
30. Να δημιουργήσετε ένα αρχείο με το παρακάτω περιεχόμενο:

```
gold     1    1986  USA                American Eagle
gold     1    1908  Austria-Hungary    Franz Josef 100   silver
10    1981  USA                ingot
gold     1    1984  Switzerland        ingot
gold     1    1979  RSA                Krugerrand
gold     0.5  1981  RSA                Krugerrand
gold     0.1  1986  PRC                Panda
silver   1    1986  USA                Liberty dollar
gold     0.25 1986  USA                Liberty 5-dollar   silver
0.5  1986  USA               Liberty 50-cent
silver   1    1987  USA                Constitution dollar
gold     0.25 1987  USA                Constitution 5-dollar
gold     1    1988  Canada             Maple Leaf
```

31. Να εμφανίσετε όλα τα νομίσματα από την USA.
32. Να βρείτε όλα τα νομίσματα που το όνομά τους τελειώνει σε r.
33. Να εμφανίσετε το πρώτο και το τελευταίο πεδίο του αρχείου.
34. Να εμφανίσετε όλα τα νομίσματα που κυκλοφόρησαν μετά από το 1980.

Προγραμματισμός στο φλοιό

Ο προγραμματισμός στο φλοιό αυξάνει τις δυνατότητες του λειτουργικού συστήματος, γιατί με μικρό σχετικά κόπο μπορούμε να δημιουργήσουμε νέες εντολές οι οποίες θα κάνουν πολύπλοκα πράγματα. Ουσιαστικά, όταν αναφερόμαστε στον προγραμματισμό φλοιού, μιλάμε για συγγραφή απλών προγραμμάτων (χρησιμοποιώντας υπάρχοντα προγράμματα, δομές ελέγχου και επανάληψης) που διερμηνεύονται απ' ευθείας από το φλοιό χωρίς να χρειάζεται η μεταγλώττισή τους.

Απλά προγράμματα φλοιού (scripts)

Για να δημιουργήσουμε ένα απλό πρόγραμμα στο φλοιό θα πρέπει να ανοίξουμε έναν επεξεργαστή κειμένου (για παράδειγμα nano) και να πληκτρολογήσουμε τις εντολές που επιθυμούμε και οι οποίες θα εκτελεστούν.

Αφού έχουμε μπει στον επεξεργαστή κειμένου και θέλουμε να δημιουργήσουμε ένα πρόγραμμα το οποίο θα εμφανίζει την τρέχουσα ημερομηνία και το όνομα χρήστη, γράφουμε:

```
#!/bin/bash
date
who am i
```

Όπως παρατηρούμε, η δεύτερη και η τρίτη γραμμή είναι γνωστές εντολές που ήδη έχουμε αναφέρει. Η πρώτη εντολή καθορίζει το φλοιό με τον οποίο θα γίνει η ερμηνεία των εντολών (είπαμε ότι υπάρχουν πολλοί φλοιοί με διαφορετική σύνταξη όσον αφορά τον προγραμματισμό).

Αφού αποθηκεύσουμε το αρχείο για να το εκτελέσουμε υπάρχουν δύο τρόποι:

- Με τον πρώτο τρόπο, θα πρέπει αρχικά να προσθέσουμε το δικαίωμα για εκτέλεση στο αρχείο με την εντολή chmod u+x όνομα_αρχείου και στη συνέχεια γράφοντας ./όνομα_αρχείου. Η τελεία και το slash μπροστά από το όνομα του αρχείου είναι υποχρεωτικά.
- Με το δεύτερο τρόπο, δε χρειάζεται να αλλάξουμε τα δικαιώματα χρήσης του αρχείου και αρκεί να γράψουμε bash όνομα_αρχείου.

Αν το όνομα του αρχείου που έχουμε δημιουργήσει είναι script1 για να το τρέξουμε με τον πρώτο τρόπο θα γράψουμε:

```
user@localhost:~ $ chmod u+x script1
user@localhost:~ $ ./script1
Tue Oct  5 09:44:48 EET 2002
theo     pts/0        2002-01-05 08:23
```

Με τον δεύτερο τρόπο αρκεί να γράψουμε:

```
user@localhost:~ $ bash script1
Tue Oct  5 09:44:48 EET 2002
theo     pts/0        2002-01-05 08:23
```

Μεταβλητές

Από τη στιγμή που μιλάμε για προγραμματισμό θα πρέπει να υπάρχουν μεταβλητές. Ο φλοιός υποστηρίζει δύο ειδών μεταβλητές, ακέραιες (integer) και συμβολοσειρές (string). Δεν χρειάζεται να ορίσουμε τον τύπο της μεταβλητής πριν την χρησιμοποιήσουμε. Για να εκχωρήσουμε μια τιμή σε μια μεταβλητή θα πρέπει απλά να γράψουμε:

```
user@localhost:~ $ var1=1234
```

Δεν υπάρχει κενό πριν και μετά από το σύμβολο =

Αυτή τη στιγμή υπάρχει η μεταβλητή var1 με τιμή 1234. Θα μπορούσαμε να καταχωρήσουμε μία συμβολοσειρά.

```
user@localhost:~ $ var2=abc
```

Για να δούμε (χρησιμοποιήσουμε) το περιεχόμενο μιας μεταβλητής θα πρέπει να εισάγουμε το σύμβολο του δολαρίου μπροστά από το όνομά της.

```
user@localhost:~ $ echo $var1
abc
```

Οι μεταβλητές έχουν τοπική εμβέλεια (μέσα στο πρόγραμμα). Αν θέλουμε μία μεταβλητή να είναι διαθέσιμη και σε άλλα προγράμματα τότε θα πρέπει να χρησιμοποιήσουμε την εντολή export.

```
user@localhost:~ $ export var1
```

Τώρα η μεταβλητή var1 είναι διαθέσιμη σε όλο το σύστημα και ονομάζεται μεταβλητή περιβάλλοντος.

Για να δούμε όλες τις μεταβλητές περιβάλλοντος χρησιμοποιούμε την εντολή env.

Μεταβλητές συστήματος

Μετά την εκκίνηση του συστήματος υπάρχουν κάποιες μεταβλητές που χρησιμοποιούνται από το σύστημα. Το όνομα αυτών των μεταβλητών είναι πάντα με κεφαλαίους χαρακτήρες και μερικές από αυτές είναι οι παρακάτω:

PATH	Οι φάκελοι μέσα στους οποίους γίνεται αναζήτηση για τις εντολές του συστήματος.
HOME	Ο προσωπικός φάκελος του χρήστη.
USER	Το όνομα χρήστη.
SHELL	Ο φλοιός που χρησιμοποιεί ο χρήστης.

Για παράδειγμα αν θέλουμε να δούμε το σύνολο των φακέλων που ψάχνει ο φλοιός για να βρει προγράμματα θα γράψουμε:

```
user@localhost:~ $ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games
```

Αριθμητικοί υπολογισμοί

Για την εκτέλεση αριθμητικών πράξεων στο φλοιό θα πρέπει να εισάγουμε την αριθμητική παράσταση μέσα σε διπλές παρενθέσεις:

```
user@localhost:~ $ echo $((14 + 45))
59
user@localhost:~ $ a=1
user@localhost:~ $ b=4
user@localhost:~ $ c=2
user@localhost:~ $ echo $(($a+$b-$c))
3
```

Οι πράξεις που υποστηρίζονται είναι οι παρακάτω:

+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Πηλίκο διαίρεσης
**	Εκθέτης
%	Υπόλοιπο διαίρεσης

Είσοδος δεδομένων

Υπάρχουν δύο τρόποι εισαγωγής δεδομένων από το χρήστη προς το πρόγραμμα. Ο πρώτος είναι με τη χρήση της εντολής read. Για παράδειγμα στο παρακάτω πρόγραμμα ζητάμε από το χρήση να εισάγει κάποιο αριθμό και τον τοποθετούμε στη μεταβλητή int1. Στη συνέχεια εκτυπώνουμε αυτόν τον αριθμό στην οθόνη:

```
#!/bin/bash
echo -n "Give a number"
read int1
echo $int1
```

Ο δεύτερος τρόπος εισαγωγής δεδομένων είναι με τη χρήση των παραμέτρων κατά την κλήση του προγράμματος όπως ακριβώς εισάγουμε παραμέτρους και στις εντολές του Linux. Σε κάθε μία παράμετρο αντιστοιχεί μία συγκεκριμένη μεταβλητή. Πιο συγκεκριμένα, η πρώτη παράμετρος αντιστοιχεί στη μεταβλητή $1, η δεύτερη στη μεταβλητή $2 και ούτω καθεξής. Δηλαδή, αν είχαμε δημιουργήσει ένα πρόγραμμα με όνομα script1 και η εκτέλεση του γινόταν με τον παρακάτω τρόπο

```
user@localhost:~ $ script1 45 23
```

τότε ο αριθμός 45 θα είχε εκχωρηθεί στην παράμετρο $1 και ο αριθμός 23 στην παράμετρο $2. Για παράδειγμα, το παρακάτω πρόγραμμα προσθέτει δύο αριθμούς οι οποίοι έχουν εισαχθεί σαν παράμετροι.

```
#!/bin/bash
echo "The summary is $(($1 + $2))"
```

Το πλήθος των παραμέτρων που έχει εισάγει ο χρήστης στο πρόγραμμά μας αποθηκεύεται στη μεταβλητή #.

```
#!/bin/bash
echo "Variables count: $#"
echo "The summary is $(($1 + $2))"
```

Εντολές ελέγχου

Οι εντολές ελέγχου μας επιτρέπουν αν εκτελούμε κάποιες εργασίες ανάλογα με τον αν μία συνθήκη είναι αληθής ή ψευδής.

Ο φλοιός του Linux, μέσω του κωδικού εξόδου, μας παρέχει τη δυνατότητα ελέγχου μιας συνθήκης. Αν ο κωδικός εξόδου έχει τιμή 0 τότε η συνθήκη θεωρείται αληθής, ενώ αν έχει οποιαδήποτε άλλη τιμή η συνθήκη θεωρείται ψευδής.

Να σημειώσουμε εδώ ότι κάθε φορά που εκτελείται μία εντολή, στο τέλος παράγεται ένας κωδικός εξόδου. Αν η εντολή ήταν επιτυχής (συντακτικά ή ως αποτέλεσμα) τότε ο κωδικός εξόδου είναι 0, ενώ σε διαφορετική περίπτωση ο κωδικός εξόδου παίρνει τιμή διαφορετική του μηδενός.

```
user@localhost: ~$ username=test_user
user@localhost: ~$ grep "$username" /etc/passwd
test_user:x:583:100::/home/test_user:/bin/bash
user@localhost: ~$ echo $?
0
user@localhost: ~$ username=test_user1
user@localhost: ~$ grep "$username" /etc/passwd
user@localhost: ~$ echo $?
1
```

Αρχικά, γίνεται ανάθεση της συμβολοσειράς test_user στην μεταβλητή username. Γίνεται αναζήτηση του περιεχομένου της μεταβλητής username στο αρχείο passwd. Η εντολή grep επιστρέφει ολόκληρη την γραμμή στην οποία βρέθηκε η συμβολοσειρά test_user. Ελέγχουμε τον κωδικό εξόδου της τελευταίας εντολής (grep) μέσω της μεταβλητής $?. Βλέπουμε ότι είναι 0 κάτι που σημαίνει ότι η προηγούμενη εντολή εκτελέστηκε με επιτυχία.

Στην μεταβλητή username ανατίθεται η συμβολοσειρά test_user1. Γίνεται αναζήτηση του περιεχομένου της μεταβλητής username στο αρχείο passwd. Η εντολή grep δεν επιστρέφει τίποτα, γιατί δεν βρέθηκε η συμβολοσειρά test_user1. Ελέγχουμε τον κωδικό εξόδου της τελευταίας εντολής (grep) μέσω της μεταβλητής $?. Βλέπουμε ότι είναι 1 κάτι που σημαίνει ότι είτε υπήρξε κάποιο λάθος στην σύνταξη της εντολής είτε δεν επέστρεψε κάποιο αποτέλεσμα.

Η εντολή If/Else

Η εντολή If είναι η απλούστερη μέθοδος για τον έλεγχο μιας συνθήκης. Η γενική σύνταξη της εντολής είναι:

```
If [ συνθήκη ]
then
    εντολή1
    εντολή2
    ….
    ….
fi
```

```
If [ συνθήκη ]
then
    εντολή1
    ….
else
    εντολή1
    ….
fi
```

Στην συνέχεια μπορείτε να δείτε όλες επιλογές που μπορούμε να γράψουμε σαν συνθήκη στην εντολή If:

Συνθήκη	Έλεγχος
[string1 == string2]	Ισότητα δύο συμβολοσειρών [“$username” == “$username1”]

[string1 != string2]	Μη ισότητα δύο συμβολοσειρών [“$username” != “$username1”]
[string]	Η συμβολοσειρά δεν είναι κενή (not null) [“$username”]
[! string1]	Μη ισότητα μιας συμβολοσειράς [! “$username” == “test”]
[έκφραση1 –a έκφραση2]	Και η έκφραση1 και η έκφραση2 να είναι αληθής [“$username” == “test” –a “$username1” == “test1”]
[έκφραση1 –o έκφραση2]	Ή η έκφραση1 ή η έκφραση2 να είναι αληθής [“$username” == “test” –o “$username1” == “test1”]
[-f όνομα αρχείου]	Ύπαρξη αρχείου
[-d όνομα φακέλου]	Ύπαρξη φακέλου
[int1 –eq int2]	Ισότητα δύο ακέραιων
[int1 –ne int2]	Ανισότητα δύο ακέραιων
[int1 –gt int2]	Ο αριθμός int1 είναι μεγαλύτερος από τον int2
[int1 –ge int2]	Ο αριθμός int1 είναι μεγαλύτερος ή ίσος από τον int2
[int1 –lt int2]	Ο αριθμός int1 είναι μικρότερος από τον int2
[int1 –le int2]	Ο αριθμός int1 είναι μικρότερος ή ίσος από τον int2

Πριν και μετά τις αγκύλες πρέπει να εισάγετε κενά. Το ίδιο ισχύει, πριν και μετά από κάθε τελεστή.

Στο παρακάτω πρόγραμμα, αρχικά δίνουμε μια τιμή στη μεταβλητή username και στη συνέχεια ελέγχουμε αν αυτήν η τιμή είναι ίδια με μια άλλη. Αν είναι εμφανίζει στην οθόνη το μήνυμα "Einai idia", ενώ σε διαφορετική περίπτωση εμφανίζει το μήνυμα "Den einai idia".

```
#!/bin/bash
# Έλεγχος ισότητας δύο συμβολοσειρών
username=testuser
if [ "$username" == "testuser" ]
then
    echo "Einai idia"
else
    echo "Den einai idia"
fi
```

Στο παρακάτω παράδειγμα, ελέγχουμε αν δύο αριθμοί είναι ίσοι.

```
#!/bin/bash
# Έλεγχος ισότητας δύο αριθμών
int1=10
int2=12
if [ "$int1" -eq "$int2" ]
then
    echo "They are equals"
else
     echo "They are not equals"
fi
```

Προσέξτε τους συμβολισμούς που χρησιμοποιούμε για τον έλεγχο αριθμών

Όπως παρατηρείτε, μέσα στις συνθήκες τα ονόματα των μεταβλητών τα περικλείουμε σε διπλά εισαγωγικά. Αυτό το κάνουμε για να αποφύγουμε το συντακτικό σφάλμα στην περίπτωση που κάποια μεταβλητή μας είναι κενή περιεχομένου. Πιο συγκεκριμένα, αν στο παραπάνω παράδειγμα η μεταβλητή int1 δεν είχε καμία τιμή και στην συνθήκη του If δεν είχαμε βάλει διπλά εισαγωγικά το αποτέλεσμα θα ήταν συντακτικό σφάλμα. Αυτό οφείλεται στο γεγονός ότι η εντολή If περιμένει σαν είσοδο τρεις παραμέτρους (δύο μεταβλητές και έναν τελεστή) ενώ εμείς της παρέχουμε μόνο δύο.

Στο παρακάτω παράδειγμα χρησιμοποιούμε το λογικό ΚΑΙ. Αν η τιμή της μεταβλητής int είναι ανάμεσα στο 5 και στο 15, τότε είναι αληθής ολόκληρη η συνθήκη.

```
#!/bin/bash
# Έλεγχος δύο συνθηκών. Πρέπει να ισχύουν και οι δύο.
int=10
if [ "$int" -ge 5 -a  "$int" -le 15 ]
then
    echo "Variable int is between 5 and 15"
else
    echo "Variable int is not between 5 and 15"
fi
```

Στο παρακάτω παράδειγμα ελέγχουμε το αποτέλεσμα μιας εντολής. Αν η εντολή δεν έχει συντακτικά λάθη και δώσει αποτελέσματα θα επιστρέψει κωδικό εξόδου 0, άρα η συνθήκη θα είναι αληθής (για το if). Δηλαδή, αν το περιεχόμενο της μεταβλητής username, το οποίο το χρησιμοποιούμε σαν λέξη αναζήτησης στην grep, βρεθεί μέσα στο αρχείο /etc/paswd το πρόγραμμα θα επιστρέψει το μήνυμα "There is a user with that name".

```
#!/bin/bash
# Έλεγχος επιστροφής αποτελέσματος από μία εντολή.
username=test_user
if cat /etc/passwd | grep "$username"
then
    echo "There is a user with that name"
```

```
else
    echo "There is not such a user"
fi
```

Γενικά, αν δεν θέλουμε να τυπώνεται το αποτέλεσμα μιας εντολής στην οθόνη θα πρέπει να κάνουμε ανακατεύθυνση της εξόδου σε ένα σημείο που αντιπροσωπεύει το κενό. Έτσι λοιπόν, θα πρέπει να γράψουμε:

cat /etc/passwd | grep "$username" > /dev/null

Η εντολή If/Elif

Με την εντολή If/Elif μπορούμε να κάνουμε πιο σύνθετους ελέγχους από ότι με την απλή If. Μπορείτε να δείτε την γενική σύνταξη της εντολής παρακάτω:

If [συνθήκη]
then
    εντολή1
    εντολή2
    ….
    ….
elif [συνθήκη]
    εντολή1
    εντολή2
    ….
    ….
else
    εντολή1
    εντολή2
    ….
    ….
fi

Στο παρακάτω παράδειγμα γίνεται έλεγχος αν δύο αριθμοί είναι ίσοι ή αν κάποιος είναι μεγαλύτερος από το άλλον.

```
# Έλεγχος δύο αριθμών.
int=5
int1=12
if [ "$int" -gt "$int1" ]
then
      echo "$int is greater than $int1"
elif [ "$int" -lt "$int1" ]
then
      echo "$int1 is greater than $int"
else
      echo "The two numbers are equal"
fi
```

Στο επόμενο παράδειγμα ελέγχουμε αν αυτό που έχει εισαχθεί σαν πρώτη παράμετρος στο πρόγραμμά μας είναι φάκελος ή αρχείο ή δεν υπάρχει.

```
# Έλεγχος ύπαρξης αρχείου ή φακέλου
```

```
if [ -f "$1" ]
then
      echo "$1 is a file"
elif [ -d "$1" ]
then
      echo "$1 is directory"
else
      echo "$1 does not exist"
fi
```

Η εντολή case

Η εντολή case είναι μία εντολή η οποία μας επιτρέπει να κάνουμε πολλαπλούς ελέγχους και μπορούμε να την χρησιμοποιήσουμε αντί της δομής If/elif. Μπορείτε να δείτε την σύνταξη της εντολής παρακάτω:

case μεταβλητή in
τιμή1)
εντολή1
εντολή2
….
;;
τιμή2)
εντολή1
εντολή2
….
;;
***)**
εντολή1
εντολή2
….
;;
esac

Στο επόμενο παράδειγμα δημιουργούμε ένα μενού επιλογών, μέσω του οποίου πατώντας το 1 εμφανίζεται το ημερολόγιο του τρέχοντα μήνα, πατώντας το 2 προβάλλεται η τρέχουσα ώρα και πατώντας το N γίνεται έξοδος από το πρόγραμμα.

```
# Δημιουργία ενός μενού επιλογών
echo “Press 1 to display current month’s calendar”
echo “Press 2 to display current time”
echo “Press N or n to leave the program”
echo “Give your command”
read choice
case “$choice” in
      1)
          echo “Month Calendar”
          cal
      ;;
```

```
      2)
          echo “Current time”
          date +%H:%M
      ;;
      “N” | “n”)
          echo “Goobye”
      ;;
      *)
           echo “Wrong choice”
      ;;
esac
```

Ο χαρακτήρας | στην περίπτωση της case χρησιμοποιείται σαν λογικό Ή

Παραδείγματα

1. Να γράψετε πρόγραμμα το οποίο θα δέχεται αποκλειστικά δύο παραμέτρους. Στην πρώτη παράμετρο θα γίνεται εισαγωγή ενός αριθμού, ενώ στην δεύτερη το όνομα ενός αρχείου. Το πρόγραμμα θα πρέπει να εμφανίζει τόσες γραμμές από την αρχή του αρχείου όσες δηλώνει ο αριθμός της πρώτης παραμέτρου. Αν το αρχείο έχει λιγότερες γραμμές από τον αριθμό της πρώτης παραμέτρου, τότε να εμφανίζεται μήνυμα λάθους και να τελειώνει το πρόγραμμα.

```
#!/bin/bash
if [ $# -eq 2 ] #Ελεγχος πλήθους παραμέτρων
then
    if [ -f $2 ]  #Έλεγχος ύπαρξη του αρχείου
    then
       lines=$(wc -l $2 | awk {'print $1}') #Πλήθος γραμμών
       if [ $1 -le $lines ]
       then
          head -$1 $2
       else
          echo "The file has less lines the number specified"
       fi
    else
       echo "File $2 does not exist!"
    fi
else
    echo "You must provide exactly two parameters to the program"
    echo "Example: prog 4 testfile"
fi
```

2. Να γράψετε πρόγραμμα το οποίο να εμφανίζει, από τον τρέχοντα φάκελο, μόνο τα αρχεία που έχουν μέγεθος μεγαλύτερο από 100 bytes.

```
#!/bin/bash
# Η εντολή grep φιλτράρει το αποτέλεσμα της εντολής ls και
# απορρίπτει οτιδήποτε δεν είναι αρχείο. Η εντολή awk
# εμφανίζει το όνομα των αρχείων που το μέγεθός τους (πέμπτη
# στήλη) έχει μέγεθος μεγαλύτερο από 100 bytes.
ls -l | grep ^- | awk '$5 > 100 {print $9}'
```

3. Να γράψετε πρόγραμμα το οποίο θα ζητά από τον χρήστη να εισάγει έναν μονοψήφιο αριθμό και στη συνέχεια θα τυπώνει αυτόν τον αριθμό με το λεκτικό του. Για παράδειγμα, αν ο χρήστης εισάγει τον αριθμό 1, τότε το πρόγραμμα θα εκτυπώνει One.

```
#!/bin/bash
echo "Please give a number"
read num
if [ $num -ge 0 -a $num -lt 10 ]
then
    case $num in
         0) echo "Zero";;
         1) echo "One";;
         2) echo "Two";;
         3) echo "Three";;
         4) echo "Four";;
         5) echo "Five";;
         6) echo "Six";;
         7) echo "Seven";;
         8) echo "Eight";;
         9) echo "Nine";;
   esac
else
   echo "Error! You must provide an one digit number"
fi
```

Ασκήσεις

1. Να γράψετε πρόγραμμα το οποίο θα ζητά από τον χρήστη να εισάγει έναν αριθμό και στη συνέχεια θα εκτυπώνει το μήνυμα “The nuber is odd” αν ο αριθμός είναι περιττός, ενώ θα εκτυπώνει το μήνυμα “The number is even” αν ο αριθμός είναι άρτιος.
2. Να βρείτε το πλήθος των αρχείων του φακέλου /etc
3. Να γράψετε πρόγραμμα το οποίο θα μετράει και θα εμφανίζει το πλήθος των αρχείων που το όνομά τους αρχίζει από k (η αναζήτηση θα πραγματοποιείται στον τρέχοντα φάκελο).
4. Να γράψετε πρόγραμμα το οποίο θα εκτυπώνει την τρέχουσα ώρα σε 12ωρη μορφή. Δηλαδή, αν η ώρα του συστήματος είναι 15:10, το πρόγραμμα να εκτυπώνει 3:10 μ.μ.
5. Να γράψετε πρόγραμμα το οποίο θα εκτυπώνει το λεκτικό του μονοψήφιου αριθμού που θα εισάγει ο χρήστης από το πληκτρολόγιο. Για παράδειγμα, αν πληκτρολογήσει τον αριθμό 1 να εμφανίζει One, αν πληκτρολογήσει τον αριθμό 2 να εμφανίζει Two και ούτω καθ’ εξής.
6. Να γράψετε πρόγραμμα το οποίο να εμφανίζει ένα μήνυμα με το οποίο θα ζητά από τον χρήστη να εισάγει την ημερομηνία γέννησής του και στη συνέχεια θα εκτυπώνει ένα από τα παρακάτω μηνύματα:
    - Αν η ηλικία του χρήστη είναι μικρότερη από 12 χρονών, το μήνυμα “Hello little boy”
    - Αν η ηλικία του χρήστη είναι μεταξύ 12 και 18, το μήνυμα “Hello teenager”
    - Αν η ηλικία του χρήστη είναι μεταξύ 19 και 45, το μήνυμα “Hello little mister”
    - Τέλος, αν ηλικία του χρήστη είναι μεγαλύτερη από 45, το μήνυμα “Hello mister”
7. Να γράψετε πρόγραμμα το οποίο να δέχεται μόνο μία παράμετρο και θα αναζητά το περιεχόμενο της παραμέτρου στο αρχείο /etc/passwd. Αν η αναζήτηση είναι επιτυχής να εκτυπώνει το μήνυμα “User xxxxx found”, ενώ σε διαφορετική περίπτωση να εκτυπώνει το μήνυμα “User not found”. Σε περίπτωση που ο χρήστης εισάγει λάθος πλήθος παραμέτρων, να εκτυπώνεται μήνυμα λάθους και να τερματίζεται το πρόγραμμα.
8. Να γράψετε πρόγραμμα το οποίο να δέχεται μόνο μία παράμετρο και θα αναζητά αν το περιεχόμενο της παραμέτρου είναι αρχείο ή φάκελος μέσα στον τρέχοντα φάκελο. Αν το περιεχόμενο της παραμέτρου είναι φάκελος να εκτυπώνει το μήνυμα “xxxx is a directory”, αν είναι αρχείο να εκτυπώνει το μήνυμα “xxxx is a file” και τέλος αν δεν υπάρχει να εκτυπώνει το μήνυμα “There is not such a file or directory”. Σε περίπτωση που ο χρήστης εισάγει λάθος πλήθος παραμέτρων, να εκτυπώνεται μήνυμα λάθους και να τερματίζεται το πρόγραμμα.
9. Να γράψετε πρόγραμμα το οποίο θα αντιγράφει το πιο πρόσφατο αρχείο του τρέχοντα φακέλου στο ίδιο φάκελο με όνομα που θα δίνεται ως παράμετρος στο πρόγραμμα. Σε περίπτωση που εισαχθούν περισσότερες ή λιγότερες παράμετροι θα πρέπει να εμφανίζεται μήνυμα λάθους και να τερματίζεται το πρόγραμμα. Θα πρέπει να γίνεται έλεγχος αν το όνομα αρχείου που έχει εισαχθεί ως παράμετρος υπάρχει ή όχι. Αν υπάρχει θα πρέπει να εμφανίζεται μήνυμα λάθους και να τερματίζεται το πρόγραμμα.

Εντολές επανάληψης

Στην προηγούμενη ενότητα είδαμε πώς μπορούμε να χειριζόμαστε τη ροή ενός προγράμματος, ελέγχοντας την τιμή μιας έκφρασης ή το αποτέλεσμα μιας εντολής. Σ' αυτήν την ενότητα, θα περιγράψουμε τις εντολές με τις οποίες έχουμε τη δυνατότητα να εκτελούμε επαναληπτικές διαδικασίες.

Η εντολή for

Πολλές φορές, χρειάζεται να εκτελέσουμε ένα σύνολο από εντολές μέχρι να ικανοποιηθεί μια συνθήκη (να εκτελέσουμε μια πράξη σε όλα τα αρχεία ενός φακέλου ή να κάνουμε μια πράξη σε όλες τις γραμμές ενός αρχείου κτλ).

Ο φλοιός μας παρέχει την εντολή for με την οποία μπορούμε να δημιουργήσουμε βρόχους. Η γενική σύνταξη της εντολής είναι η παρακάτω:

for μεταβλητή in λίστα_τιμών
do
    εντολή1
    εντολή2
    ….
    ….
done

Στο παρακάτω παράδειγμα εκτυπώνονται όλες οι πόλεις οι οποίες βρίσκονται στη θέση λίστα_τιμών, οι οποίες χωρίζονται με κενό η μια από την άλλη.

```
#!/bin/bash
for cityname in Athens London Paris Helsinki "New York"
do
   echo The next city is $cityname
done
```

Επειδή οι τιμές πρέπει να χωρίζονται με κενό, θα πρέπει να περικλείουμε με διπλά εισαγωγικά τις τιμές οι οποίες περιέχουν κενά (όπως το New York στο παραπάνω παράδειγμα).

Τη λίστα τιμών, έχουμε τη δυνατότητα να την «γεμίσουμε» με το αποτέλεσμα μια εντολής. Για παράδειγμα:

```
user@localhost: ~$ cat cities
Thessaloniki
Kozani
Kastoria
Kavala
Patra
```

```
#!/bin/bash
for cityname in $(cat cities)
do
   echo The next city is $cityname
done
```

Στο παραπάνω παράδειγμα, το αποτέλεσμα της εντολής cat cities δημιούργησε μια λίστα τιμών την οποία τοποθετήσαμε σαν λίστα τιμών στην εντολή for.

Επίσης, μπορούμε να εκτελέσουμε ένα σύνολο από εντολές στα αρχεία ενός φακέλου μέσω μιας εντολής for. Στο παρακάτω παράδειγμα, ελέγχουμε τον τύπο όλων των αρχείων του προσωπικού μας φακέλου.

```
#!/bin/bash
for file in /home/user/*
do
   if [ -d "$file" ]
   then
      echo "$file is a directory"
   elif [ -f "$file" ]
   then
      echo "$file is a file"
   fi
done
```

Ο φλοιός μας δίνει τη δυνατότητα να συντάξουμε την εντολή for όπως τη γνωρίζουμε από τη γλώσσα C.

for ((αρχική_τιμή;συνθήκη_λήξης;βήμα_αύξησης))
do
εντολή1
εντολή2
….
….
done

Στο παρακάτω παράδειγμα εκτυπώνονται οι αριθμοί από το 1 μέχρι και το 10.

```
#!/bin/bash
for (( i=1; i<=10; i++ ))
do
   echo $i
done
```

Η εντολή while

Η εντολή while μας επιτρέπει να εκτελούμε ένα σύνολο εντολών όσο είναι αληθής μια συνθήκη. Η γενική της σύνταξη είναι:

while [συνθήκη]
do
εντολή 1
εντολή 2
…….
done

```
#!/bin/bash
var1=10
while [ $var1 -gt 0 ]
do
echo $var1
```

```
var1=$(expr $var1 - 1)
done
```

Στο πιο πάνω παράδειγμα εκτυπώνονται οι αριθμοί από το 10 μέχρι το 1.

Μπορούμε να χρησιμοποιήσουμε τον τελεστή –a σαν λογικό ΚΑΙ ή τον τελεστή –o σαν λογικό Ή για δημιουργήσουμε σύνθετες εκφράσεις.

Η εντολή until

Η εντολή until λειτουργεί ακριβώς αντίστροφα από ότι η εντολή while. Όσο η έκφραση (που έχουμε εισάγει στην until) δεν είναι αληθής εκτελούνται οι εντολές που βρίσκονται στο σώμα, ενώ μόλις γίνει αληθής σταματάει η εκτέλεση. Η γενική της σύνταξη είναι:

until [συνθήκη]
do
    εντολή 1
    εντολή 2
    …….
done

```
#!/bin/bash
var1=100
until [ $var1 -eq 0 ]
do
echo $var1
var1=$(($var1 - 20))
done
```

Στο παραπάνω πρόγραμμα, εκτυπώνονται αριθμοί ξεκινώντας από το 100 και φτάνοντας μέχρι το 0 με βήμα 20.

Παραδείγματα

1. Να γράψετε πρόγραμμα το οποίο θα εμφανίζει το πλήθος των αρχείων και των φακέλων του τρέχοντα φακέλου.

```
#!/bin/bash
files=0  #Μετρητής αρχείων
dirs=0   #Μετρητής φακέλων
others=0  #Μετρητής άλλου τύπου αρχείων
for filename in *
do
  if [ -f $filename ]
  then
      files=$(($files + 1))
  elif [ -d $filename ]
  then
      dirs=$(($dirs + 1))
  else
      others=$((others + 1))
```

```
  fi
done
echo "Files: $files"
echo "Directories: $dirs"
echo "Other types: $others"
```

2. Να δημιουργήσετε πρόγραμμα το οποίο θα δέχεται μία παράμετρο. Να γίνεται έλεγχος του πλήθους των παραμέτρων και σε περίπτωση εισαγωγής περισσότερων ή λιγότερων να εμφανίζεται μήνυμα λάθους και να ολοκληρώνεται η εκτέλεση του προγράμματος. Αν το πλήθος των παραμέτρων είναι σωστό, τότε να εκτυπώνεται η προπαίδεια του αριθμού που έγινε εισαγωγή σαν παράμετρος. Η έξοδος του προγράμματος να είναι όπως παρακάτω (αν εισάγουμε το 2 σαν παράμετρο):

   1 * 2 = 2
   2 * 2 = 4
   3 * 2 = 6

```
#!/bin/bash
if [ $# -eq 1 ]
then
    for ((i=1;i<=10;i++))
    do
      echo "$i * $1 = $(($1 * $i))"
    done
else
   echo "You must insert only one parameter"
fi
```

3. Να δημιουργήσετε πρόγραμμα το οποίο θα επιτρέπει την εισαγωγή αριθμών από το πληκτρολόγιο και θα βρίσκει το άθροισμα τους. Η εισαγωγή αριθμών θα σταματάει μόλις ο χρήστης εισάγει έναν αριθμό μεγαλύτερο από 1000.

```
#!/bin/bash
sum=0
read num
while [ $num -le 1000 ]
do
  sum=$(($sum + $num))
  read num
done
echo "The summary is $sum"
```

4. Να δημιουργήσετε ένα πρόγραμμα, το οποίο θα υπολογίζει το συνολικό μέγεθος των κανονικών αρχείων του τρέχοντα φακέλου.

```
#!/bin/bash
total=0
for size in $(ls -l | grep ^- | awk {'print $5'})
do
```

```
  total=$(($total + $size))
done
echo "Total size of the file is: $total"
```

Ασκήσεις

1. Να γράψετε πρόγραμμα το οποίο να ζητάει από τον χρήστη να εισάγει ένα όνομα αρχείου. Όσο δεν υπάρχει το αρχείο το πρόγραμμα θα συνεχίζει να εκτελείται.
2. Να γράψετε πρόγραμμα το οποίο θα ζητάει από τον χρήστη να εισάγει έναν αριθμό από το 20 μέχρι το 30. Αν ο χρήστης έχει εισάγει αριθμό εκτός των ορίων να εμφανίζει μήνυμα λάθους και να εκτελείται από την αρχή.
3. Να δημιουργήσετε 150 εκτελέσιμα αρχεία με όνομα το οποίο θα αρχίζει από file και θα καταλήγει στον τρέχοντα αύξοντα αριθμό. Για παράδειγμα, file1, file2, file3,....,file150. Το περιεχόμενο του κάθε ενός αρχείου θα είναι ο αύξοντας αριθμός του ονόματός του.
4. Να γράψετε πρόγραμμα το οποίο θα υπολογίζει τον μέσο όρο των αμοιβών (δεύτερη στήλη) από το παρακάτω αρχείο:
   Bill 12000
   John 13200
   Jenny 32100
   George 32100
   Helen 14000
5. Να γράψετε πρόγραμμα το οποίο θα εμφανίζει το παρακάτω μενού επιλογών:
   1. Εμφάνιση ημερομηνίας
   2. Εμφάνιση ώρας
   3. Εμφάνιση ονόματος χρήστη
   Για έξοδο πατήστε είτε E είτε e.
   Πατώντας τον αριθμό 1 θα εμφανίζεται η ημερομηνία του συστήματος, πατώντας το 2 θα εμφανίζεται η ώρα και πατώντας το 3 θα εμφανίζεται το όνομα του χρήστη. Μετά την εκτέλεση μιας επιλογής, θα εμφανίζεται και πάλι το μενού. Το πρόγραμμα θα τελειώνει αν ο πατήσουμε είτε το γράμμα E είτε το e.

Όπως γνωρίζουμε, για να γράψουμε και να μεταγλωττίσουμε ένα πρόγραμμα σε γλώσσα C αρκεί να έχουμε έναν κειμενογράφο (για παράδειγμα το nano) και έναν μεταγλωττιστή, το gcc στην περίπτωση του Linux. Για παράδειγμα, μπορούμε να δημιουργήσουμε ένα αρχείο στο nano με όνομα hello.c και το κάτωθι περιεχόμενο:

```
#include <stdio.h>
main()
{
    printf("Hello, world!\n");
    return 0;
}
```

Η μεταγλώττιση του παραπάνω προγράμματος μπορεί να γίνει με την εντολή:

```
gcc hello.c
```

Θα παρατηρήσουμε ότι παράγεται ένα αρχείο με όνομα a.out το οποίο είναι το μεταγλωττισμένο και εκτελέσιμο αρχείο του προγράμματος hello.c. Το εξαγόμενο αρχείο πήρε ένα όνομα, γιατί δεν καθορίσαμε εμείς το όνομά του. Για να καθορίσουμε το όνομα του αρχείο εξαγωγής πρέπει να γράψουμε:

```
gcc –o hello hello.c
```

Με την επιλογή –o καθορίζουμε το όνομα του εκτελέσιμου αρχείου, το οποίο θα δημιουργηθεί στην περίπτωση που δεν υπάρχουν συντακτικά λάθη στο πρόγραμμά μας. Για να εμφανιστούν συντακτικά λάθη (errors) και προειδοποιήσεις (warnings) κατά τη διάρκεια της μεταγλώττισης θα πρέπει να εισάγουμε ακόμα μια παράμετρο:

```
gcc –Wall –o hello hello.c
```

Για την εκτέλεση του αρχείου του παραδείγματος πρέπει να γράψουμε ./hello

Παραδείγματα

Στο πρώτο παράδειγμα, διαβάζουμε έναν αριθμό από το πληκτρολόγιο, ο οποίος αντιπροσωπεύει βαθμούς Κελσίου και το μετατρέπουμε σε βαθμούς Φαρενάιτ.

```
#include <stdio.h>

int main (void)
{
        float far,cel;
        printf ("Please, give the temperature in Celsius: ");
        scanf ("%f", &cel);
        far=cel * 9/5 + 32;
        printf ("%f Celsius = %f Fahrenheit\n", cel, far);

        return 0;
}
```

Αν έχουμε ονομάσει το πρόγραμμα cel2far.c τότε για την μεταγλώττισή του θα γράψουμε gcc –Wall –o far2cel far2cel.c

Στο επόμενο παράδειγμα, βρίσκουμε τον μικρότερο μεταξύ τριών ακεραίων αριθμών.

```
#include <stdio.h>

int main (void)
{
        int num1, num2, num3;
        int smallest;

        printf ("Please insert three numbers ");
        scanf ("%d %d %d", &num1, &num2, &num3);

        if (num1 <= num2)
        {
                if (num1 <= num3)
                        smallest=num1;
                else
                        smallest=num3;
        }
        else
        {
                if (num2 <= num3)
                        smallest=num2;
                else
                        smallest=num3;
        }

        printf ("The smallest number is %d \n", smallest);
        return 0;
}
```

Στο παρακάτω πρόγραμμα εμφανίζουμε την προπαίδεια του αριθμού που θα έχει πληκτρολογήσει ο χρήστης.

```
#include <stdio.h>

int main (void)
{
        int i, num;

        printf ("Please give a number ");
        scanf ("%d", &num);
```

```
        for (i=1;i<=10;i++)
                printf ("%d * %d = %d \n", i, num, num * i);


        return 0;
}
```

Στο παρακάτω πρόγραμμα βρίσκουμε τον μεγαλύτερο, τον μικρότερο και τον μέσο όρο n πραγματικών αριθμών.

```
#include <stdio.h>

int main (void)
{     int j, n;
      double max, min, sum;

      printf ("Enter number of elements: ");
      scanf ("%d", &n);
       double a[n];
          printf ("Enter %d numbers: ", n);
          for (j = 0; j < n; j++)
                scanf ("%lf", &a[j]);
          sum = max = min = a[0];
          for (j = 1; j < n; j++)
              { if (a[j] > max)
                    max = a[j];
                if (a[j] < min)
                    min = a[j];
                sum += a[j];
              }
          printf ("Maximum:      %5.2f\n", max);
          printf ("Minimum:      %5.2f\n", min);
          printf ("Average:      %5.2f\n\n", sum/n);

          printf ("Enter the index (1..n) of the number to be
printed: ");
          scanf    ("%d", &j);
          printf ("The %d th number is %.2f\n", j, a[j-1]);
      return 0;
}
```

Στο παρακάτω πρόγραμμα αποθηκεύουμε σε ένα αρχείο τους αριθμούς από το 1 μέχρι και το 10.

```
#include <stdio.h>
```

```
int main (void)
{
    FILE   *file1;
    int i;

    file1 = fopen ("integers", "w");
    for (i = 1; i <= 10; i++)
           fprintf (file1, "%d\n", i);

    fclose (file1);
    return 0;
}
```

Στο επόμενο παράδειγμα γίνεται ανάγνωση και εκτύπωση των περιεχομένων του αρχείου integers.

```
#include <stdio.h>

int main (void)
{
    FILE *intfile;
    int i;

    printf ("The integers from 'integers' follow:\n");
    intfile = fopen ("integer.file", "r");

    while (fscanf (intfile, "%d", &i) != EOF)
          printf ("%10d\n", i);

    fclose (intfile);
    return 0;
}
```

Αν θέλουμε να εισάγουμε δεδομένα στο πρόγραμμά μας μέσω παραμέτρων (κατά την κλήση του προγράμματος), θα πρέπει να εισάγουμε στην συνάρτηση main του προγράμματος το argc (είναι ένας ακέραιος που περιέχει το πλήθος των παραμέτρων που έχει εισάγει ο χρήστης που κάλεσε το πρόγραμμα) και το argv (είναι ένα διάνυσμα το οποίο περιέχει τις παραμέτρους που έχουν εισαχθεί). Στο παρακάτω παράδειγμα, γίνεται έλεγχος αν έχει εισαχθεί κάποια παράμετρος στο πρόγραμμα και να έχει εισαχθεί, απλά την εμφανίζει.

```
#include<stdio.h>
main (int argc, char *argv[])
{
```

```
     if(argc==1)
     {
         printf("No parameter. \n");
         exit(0);
     }
     printf("You have inserted %s \n",argv[1]);
}
```

Στο επόμενο παράδειγμα βρίσκουμε το άθροισμα τριών αριθμών που έχουν εισαχθεί ως παράμετροι στο πρόγραμμα.

```
#include<stdio.h>
#include<stdlib.h>

main(int argc, char *argv[])
{
         int sum;

         if(argc !=4 )
         {
                  printf("You must input three numbers as command
line inputs. \n");
                  exit(0);
         }
         else
         {
                sum=atoi(argv[1])+atoi(argv[2])+atoi(argv[3]);
                printf("The sum of three values is =  %d\n",sum);
         }
}
```

Το στοιχείο argv[0] είναι το όνομα του αρχείου.

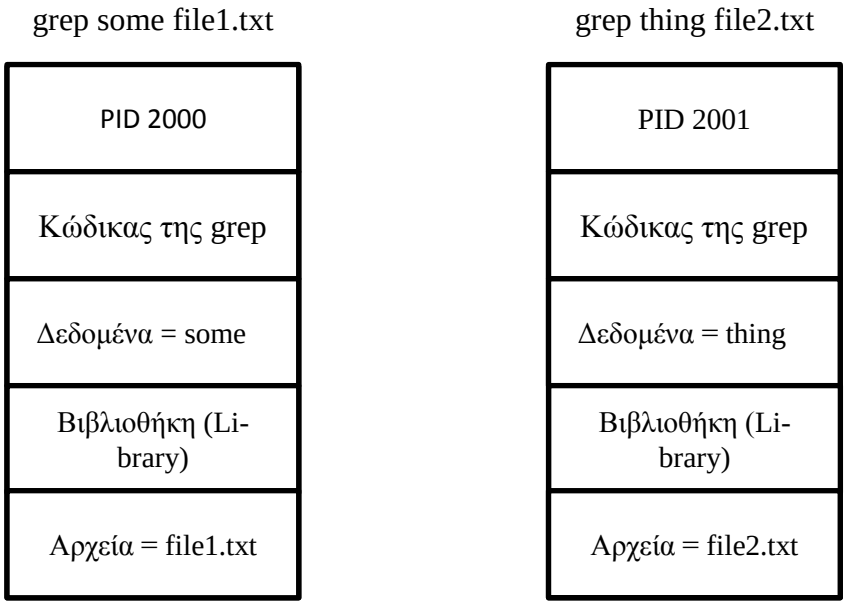
Η συνάρτηση atoi () μετατρέπει ένα αλφαριθμητικό σε ακέραιο αριθμό.

Διεργασίες

Οι διεργασίες είναι βασική έννοια στον κόσμο των λειτουργικών συστημάτων, γιατί μέσω αυτών εκτελούνται όλες οι εργασίες σε ένα υπολογιστικό σύστημα. Ως διεργασία ορίζεται ένας χώρος διευθύνσεων με ένα ή περισσότερα νήματα εκτέλεσης και όλους τους απαραίτητους πόρους συστήματος για αυτά τα νήματα (IEEE Std 1003.1, 2004 Edition). Ένα λειτουργικό σύστημα που υποστηρίζει πολυπρογραμματισμό μπορεί και εκτελεί παράλληλα πολλά προγράμματα. Το κάθε ένα από αυτά τα προγράμματα συνθέτουν μια διεργασία. Επίσης, σε ένα σύστημα πολλών χρηστών επιτρέπεται η εκτέλεση πολλών προγραμμάτων από πολλούς χρήστες ταυτόχρονα.

Δομή μιας διεργασίας

Αν υποθέσουμε ότι δύο χρήστες κάνουν μια αναζήτηση σε ένα αρχείο χρησιμοποιώντας την εντολή grep, μια εικόνα των δύο διεργασιών θα είναι η παρακάτω:



Με τη δημιουργία μιας διεργασίας, το ΛΣ της δίνει έναν μοναδικό θετικό ακέραιο αριθμό (PID – Process ID) που την κάνει να ξεχωρίζει από τις υπόλοιπες. Στη συνέχεια φορτώνεται στη μνήμη ο κώδικας του προγράμματος (στο παράδειγμά μας ο κώδικας της grep), τα δεδομένα εισόδου (στο πρώτο παράδειγμα some και οι απαραίτητες βιβλιοθήκες, οι οποίες μπορεί να είναι διαμοιραζόμενες (shared libraries). Επίσης, κάθε μια διεργασία έχει στην κατοχή της μια στοίβα (stuck) για την αποθήκευση των τοπικών μεταβλητών, των κλήσεων και δεδομένων επιστροφής των συναρτήσεων.

> Στην περίπτωση διαμοιραζόμενων βιβλιοθηκών, αυτές φορτώνονται μια φορά και χρησιμοποιούνται από όλες τις διεργασίες που τις χρειάζονται.

Παρακολούθηση διεργασιών

Για να δούμε πληροφορίες για τις διεργασίες που εκτελούνται στο σύστημά μας, χρησιμοποιούμε την εντολή ps. Αν θέλουμε να δούμε όλες τις διεργασίες που εκτελούνται στο σύστημα μπορούμε αν γράψουμε:

```
user@localhost:~$ ps -el
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  0  80   0 -   508 ?      ?        00:00:01 init
1 S     0     2     0  0  80   0 -     0 ?      ?        00:00:00 kthreadd
1 S     0     3     2  0 -40   - -     0 ?      ?        00:00:00 migration/0
1 S     0     4     2  0  80   0 -     0 ?      ?        00:00:00 ksoftirqd/0
5 S     0     5     2  0 -40   - -     0 ?      ?        00:00:00 watchdog/0
```

```
.................
0 S  1000  1730  1728  0  80   0 -  1455 -      pts/0    00:00:00 bash
1 S     0  1820     2  0  80   0 -     0 ?      ?        00:00:00 flush-8:0
0 R  1000  2017  1730  0  80   0 -   915 -      pts/0    00:00:00 ps
```

Στο αποτέλεσμα της εντολής, κάθε μια γραμμή αντιπροσωπεύει μια διεργασία, ενώ σε κάθε μια στήλη υπάρχει κάποια συγκεκριμένη πληροφορία.

Στήλη	Περιγραφή
S	Η κατάσταση που βρίσκεται η διεργασία, με τις παρακάτω δυνατές τιμές: • S (Sleeping), σε «ύπνωση» με δυνατότητα διακοπής. Δηλαδή, η διεργασία έχει σταματήσει την εκτέλεσή της και περιμένει κάποιο συμβάν για να συνεχίσει • R (Runnable), εκτελέσιμη. Δηλαδή, είτε εκτελείται είτε βρίσκεται στη λίστα με τις προς εκτέλεση διεργασίες • D (Sleeping Uninterruptible), σε ύπνωση χωρίς τη δυνατότητα διακοπής • T (Stopped), ακινητοποιημένη διεργασία ή υπό εκσφαλμάτωση (debugging). Δηλαδή, η διεργασία έχει σταματήσει την εκτέλεσή της είτε λόγω επέμβασης του χρήστη είτε λόγω επέμβασης μιας άλλης διεργασίας • Z (Zombie), δηλαδή η διεργασία έχει τελειώσει την εργασία της, αλλά η γονική της διεργασία δεν την έχει διαγράψει ακόμα
UID	Το προσδιοριστικό του χρήστη στον οποίο ανήκει η διεργασία (User ID).
PID	Το προσδιοριστικό της διεργασίας (Process Identifier).
PPID	Το προσδιοριστικό της γονικής διεργασίας.
PRI	Η δυναμική προτεραιότητα της διεργασίας. Οι τιμές που μπορεί να πάρει κυμαίνονται από -40 έως 90. Οι διεργασίες πραγματικού χρόνου παίρνουν τιμές από -40 μέχρι 59. Οι κανονικές διεργασίες μπορούν να πάρουν τιμές από 60 μέχρι 90, με αυτήν που έχει τιμή 60 να έχει μεγαλύτερη προτεραιότητα.
NI	Η τιμή niceness της διεργασίας.
TTY	Το τερματικό από το οποίο ελέγχεται η διεργασίας.
TIME	Ο συνολικός χρόνος που έχει απασχολήσει την ΚΜΕ η διεργασίας.

CMD	Το εκτελέσιμο πρόγραμμα που αντιστοιχεί στη διεργασία.

Έχουμε τη δυνατότητα να δούμε, σε πραγματικό χρόνο, τις διεργασίες που εκτελούνται στο σύστημά μας, καθώς επίσης και άλλες πληροφορίες με την εντολή top.

```
user@localhost:~$ top
top - 10:06:42 up  1:29,  2 users,  load average: 0.07, 0.02, 0.00
Tasks: 113 total,   1 running, 112 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.0%us,  1.0%sy,  0.0%ni, 98.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    987732k total,   400860k used,   586872k free,   103844k buffers
Swap:   392184k total,        0k used,   392184k free,   191352k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
  919 root      20   0 46844  21m 6444 S  1.7  2.2   0:18.91 Xorg
 1728 theo      20   0 83044  11m 9264 S  0.3  1.2   0:06.28 gnome-terminal
 2027 theo      20   0  2464 1168  896 R  0.3  0.1   0:00.04 top
    1 root      20   0  2032  704  612 S  0.0  0.1   0:01.16 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:00.02 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    6 root      20   0     0    0    0 S  0.0  0.0   0:00.09 events/0
    7 root      20   0     0    0    0 S  0.0  0.0   0:00.00 cpuset
    8 root      20   0     0    0    0 S  0.0  0.0   0:00.00 khelper
```

Για έξοδο από το περιβάλλον της top πρέπει να πατήσουμε το πλήκτρο q

Δημιουργία διεργασίας

Οι διεργασίες είναι οργανωμένες ιεραρχικά. Αρχικά, υπάρχει μόνο μια διεργασία (είναι η διεργασία init με PID=1) μέσα από την οποία δημιουργούνται όλες οι υπόλοιπες διεργασίες. Με αυτόν τον τρόπο δημιουργείται ένα δέντρο διεργασιών, το οποίο αποτελείται από γονικές διεργασίες και διεργασίες παιδιά, οι οποίες κληρονομούν τις ιδιότητες της γονικής τους διεργασίας.

Η κλήση συστήματος με την οποία μπορούμε να δημιουργήσουμε μια διεργασία είναι η fork (). Εκτελώντας την fork (), ουσιαστικά δημιουργούμε ένα πιστό αντίγραφο της γονικής διεργασίας με μόνη διαφορά το PID. Η διεργασία παιδί εκτελεί παράλληλα το ίδιο πρόγραμμα με την γονική διεργασία.

Όταν τελειώσει την εκτέλεσή της η διεργασία παιδί, στέλνει ένα μήνυμα στην γονική διεργασία για να την ενημερώσει και να εκτελέσει πιθανώς κάποια ενέργεια. Μια διεργασία η οποία περιμένει την γονική της διεργασία να αποδεχτεί το μήνυμα τερματισμού της, ονομάζεται zombie. Αν για οποιονδήποτε λόγο η γονική διεργασία τερματιστεί πριν από μια διεργασία παιδί (ορφανή διεργασία) τότε αυτήν η διεργασία υιοθετείται από την αρχική διεργασία του λειτουργικού συστήματος.

Παράδειγμα

```
#include <stdio.h>
#include <unistd.h>

int main(void)
 {
        printf("Hello World!\n");
        fork( );
        printf("I am after forking\n");
        printf("\tI am process %d.\n", getpid( ));
```

```
}
```

Αφού μεταγλωττίσουμε και εκτελέσουμε το παραπάνω πρόγραμμα, θα δούμε ότι αρχικά εμφανίζει το μήνυμα Hello World!. Στη συνέχεια, αφού εκτελεστεί η εντολή fork() θα δημιουργηθεί μια διεργασία. Όταν δημιουργείται μια διεργασία:

- Η καινούρια διεργασία είναι πιστό αντίγραφο της διεργασίας που την δημιούργησε (γονικής).
- Η εκτέλεση των δύο διεργασιών είναι εντελώς ανεξάρτητη.
- Η εκτέλεση της διεργασίας παιδί αρχίζει μετά την εκτέλεση της εντολής fork().
- Δεν γνωρίζουμε εκ των προτέρων πια διεργασία θα εκτελεστεί πρώτη.
- Η επιστρεφόμενη τιμή της εντολής fork(), μας επιτρέπει να ξεχωρίζουμε την γονική διεργασία από την διεργασία παιδί.

Παράδειγμα

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void) {
   pid_t childpid;

   childpid = fork();
   if (childpid == -1) {
      perror("Failed to fork");
      return 1;
   }
   if (childpid == 0)     /* child code */
      printf("I am child %d\n", getpid());
   else                   /* parent code */
      printf("I am parent %d\n", getpid());
   return 0;
}
```

Η συνάρτηση getpid() επιστρέφει το pid της διεργασίας. Η συνάρτηση perror() εμφανίζει κάποιο μήνυμα λάθους στην οθόνη.

Αν η επιστρεφόμενη τιμή της fork() είναι αρνητική, τότε σημαίνει ότι για κάποιον λόγο δεν ήταν δυνατή η δημιουργία της νέας διεργασίας. Αν είναι ίση με το 0, σημαίνει ότι η διεργασία παιδί δημιουργήθηκε και τρέχει ο κώδικάς της. Τέλος, αν η τιμή της είναι διαφορετική από το 0 τρέχει ο κώδικας της γονικής διεργασίας.

Αναμονή της γονικής διεργασίας

Όταν μια διεργασία δημιουργεί μια διεργασία παιδί, εκτελούνται τόσο η γονική όσο και η διεργασία παιδί. Όμως, η γονική διεργασία μπορεί να καλέσει την συνάρτηση wait() με σκοπό να περιμένει να εκτελεστεί πρώτα η διεργασία παιδί. Η συνάρτηση wait() αναγκάζει την διεργασία που την κάλεσε να μπει στην αναμονή, μέχρι να λάβει ένα σήμα. Συνήθως, αυτό είναι το σήμα τερματισμού της διεργασίας παιδί.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {
   pid_t childpid;
   int i, n, status;

   if (argc != 2) {
      fprintf(stderr, "Usage: %s n\n", argv[0]);
      return 1;
   }
   n = atoi(argv[1]);
   for (i = 1; i < n; i++)
      if ((childpid = fork()) <= 0)
         break;

   while(wait(&status) > 0) ; /* wait for all of your children */
   printf("i:%d  process ID:%d  parent ID:%d  child ID:%d\n", i,
getpid(), getppid());
   return 0;
}
```

Στο παραπάνω πρόγραμμα, εισάγοντας έναν αριθμό, σαν παράμετρο, δημιουργείται αντίστοιχος αριθμός διεργασιών. Στη συνέχεια, η γονική διεργασία περιμένει όλα τα παιδιά της να ολοκληρώσουν την εκτέλεσή τους και έπειτα ολοκληρώνει την εκτέλεσή της και η ίδια.

> Η status περιέχει την κατάσταση της διεργασίας παιδί. Μερικές από τις τιμές που μπορεί να πάρει είναι αν η διεργασία έχει ολοκληρωθεί ομαλά, αν έχει τερματιστεί από κάποιο σήμα, αν αντιμετώπισε κάποιο σφάλμα στην εκτέλεσή της κτλ)

Ορφανή διεργασία

Όταν ολοκληρώνεται η εκτέλεση της γονικής διεργασίας πριν την ολοκλήρωση της διεργασίας παιδί, τότε η διεργασία παιδί μένει ορφανή (orphan) και αυτόματα υιοθετείται από την αρχική διεργασία (init). Στο παρακάτω παράδειγμα, έχουμε εισάγει μια καθυστέρηση τεσσάρων δευτερολέπτων στον κώδικα της διεργασίας παιδί, με σκοπό να ολοκληρωθεί πρώτα η γονική διεργασία. Στο τέλος, αυτό που θα παρατηρήσουμε είναι ότι εμφανίζεται η αρχική διεργασία σαν γονική της διεργασίας παιδιού, αφού ο πραγματικός της γονέας ολοκληρώθηκε πριν από αυτήν.

```
#include <stdio.h>
#include <unistd.h>

int main(void)
 {
```

```
      pid_t childpid;

      printf("Original process with PID %d and PPID %d.\n",
getpid(), getppid());

      childpid = fork ( );
      if ( childpid != 0 )
      {
          printf("Parent with PID %d and PPID %d.\n", getpid(),
getppid()) ;
          printf("Child PID is %d\n", childpid ) ;
      }
      else
      {
          sleep(4);
          printf("Child with PID %d and PPID %d.\n", getpid(),
getppid()) ;
      }
      printf ("PID %d terminates.\n", getpid());
 }
```

Διεργασία zombie

Για να ολοκληρώσει την εκτέλεσή της μια διεργασία παιδί, θα πρέπει να «συμφωνήσει» και η γονική διεργασία, αποδεχόμενη τον ανάλογο σήμα από το παιδί. Μια διεργασία ονομάζεται zombie όταν ενώ έχει στείλει το σήμα τερματισμού στην γονική της διεργασία, αυτήν δεν το έχει αποδεχτεί. Στο παρακάτω πρόγραμμα δημιουργούμε μια διεργασία zombie.

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main ()
{
  pid_t child_pid;

  child_pid = fork ();
  if (child_pid > 0) {
    sleep (100);
  }
  else {
    exit (0);
  }
```

```
  return 0;
}
```

Αφού μεταφράσουμε το πρόγραμμα, θα το εκτελέσουμε στο παρασκήνιο εισάγοντας το σύμβολο & μετά το όνομα του εκτελέσιμου αρχείου. Δηλαδή, αν το όνομα του προγράμματος είναι fork4, για την εκτέλεση του στο παρασκήνιο θα γράφαμε fork4 &. Για να δούμε τις διεργασίες που εκτελούνται στο σύστημά μας θα εισάγουμε την εντολή ps –el. Κοντά στο τέλος του αποτελέσματος της προηγούμενης εντολής θα παρατηρήσουμε την διεργασία fork4 να αναφέρεται ως defunct ενώ στην στήλη της κατάστασης θα υπάρχει το Z, το οποίο μας ενημερώνει ότι η συγκεκριμένη διεργασία είναι zombie.

Εκτέλεση κώδικα από την διεργασία παιδί

Από τη στιγμή που δημιουργείται μια διεργασία παιδί, αυτήν αντιγράφει την γονική διεργασία και εκτελείται παράλληλα με αυτήν. Αν θέλουμε η διεργασία παιδί να εκτελεί άλλο κώδικα από αυτόν την γονικής διεργασίας πρέπει να χρησιμοποιήσουμε μια συνάρτηση από την οικογένεια συναρτήσεων exec. Δηλαδή, μέσω αυτών των συναρτήσεων θα φορτώνουμε και θα εκτελούμε διαφορετικό κώδικα στη διεργασία παιδί από ότι στη γονική της διεργασία.

Σε κανονικές συνθήκες, οι συναρτήσεις exec δεν επιστρέφουν τίποτα, ενώ σε περίπτωση σφάλματος επιστρέφουν τιμή -1

Στο παρακάτω παράδειγμα, δημιουργούμε μια διεργασία παιδί, η οποία εκτελεί την εντολή ls –l.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int  main(void) {
   pid_t childpid;

   childpid = fork();
   if (childpid == -1)  {
       perror("Failed to fork");
       return 1;
   }
   if (childpid == 0) {                    /* child code */
       execl("/bin/ls", "ls", "-l", NULL);
       perror("Child failed to exec ls");
       return 1;
   }
   if (childpid != wait(NULL)) {           /* parent code */
       perror("Parent failed to wait due to signal or error");
       return 1;
   }
```

```
    return 0;
}
```

Η συνάρτηση execl, αντικαθιστά τον κώδικα της διεργασίας με αυτόν που ορίζεται στην πρώτη της παράμετρο. Μπορούμε να προσθέσουμε πολλές παραμέτρους στην εντολή, ενώ θα πρέπει να τελειώνει με την παράμετρο NULL. Αν θα θέλαμε η διεργασία να ανοίγει ένα αρχείο με τον nano θα γράφαμε execl (“/usr/bin/nano”, “nano”, “/home/user/test.txt”, NULL)

Ασκήσεις

1. Εκτελέστε το παρακάτω πρόγραμμα:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void) {
   pid_t childpid;
   pid_t mypid;

   mypid = getpid();
   childpid = fork();
   if (childpid == -1) {
      perror("Failed to fork");
      return 1;
   }
   if (childpid == 0)
      printf("I am child %d, ID = %d\n", getpid(), mypid);
   else
      printf("I am parent %d, ID = %d\n", getpid(), mypid);
   return 0;
}
```

   Θα παρατηρήσετε ότι η μεταβλητή mypid παίρνει την ίδια τιμή και στην γονική και στην διεργασία παιδί. Γιατί συμβαίνει αυτό;

2. Εκτελέστε το παρακάτω πρόγραμμα:

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
    pid_t pid;
    char *message;
    int n;

    printf("fork program starting\n");
    pid = fork();
    switch(pid)
```

```
    {
         case -1:
            perror("fork failed");
            exit(1);
         case 0:
            message = "This is the child";
            n = 5;
            break;
         default:
            message = "This is the parent";
            n = 3;
            break;
     }

    for(; n > 0; n--) {
        puts(message);
        sleep(1);
     }
     exit(0);
}
```

Πόσες φορές θα εκτελεστεί η γονική διεργασία και πόσες φορές η διεργασία παιδί;

Βιβλιογραφία

- Bash Guide for Beginners, Machtelt Garrels, http://tldp.org/LDP/Bash-Beginners-Guide/html/index.html
- Introduction to Linux - A Hands on Guide, Machtelt Garrels, http://tldp.org/LDP/intro-linux/html/index.html
- A Sed and Awk Micro-Primer, http://tldp.org/LDP/abs/html/sedawk.html
- Programming in C, http://www.cs.cf.ac.uk/Dave/C/
- Advanced Linux Programming, http://www.advancedlinuxprogramming.com/alp-folder