ΠΑΝΕΠΙΣΤΗΜΙΟ

ΠΕΛΟΠΟΝΝΗΣΟΥ

Σημειώσεις για το λειτουργικό σύστημα UNIX

Κ. Βασιλάκης, Α. Σωτηροπούλου, Α. Καλόξυλος, Δ. Γούσκος, Μ. Νικολαΐδου

Περιεχόμενα

1 Εισαγωγή στο UNIX

1.1 Το Κέλυφος ή φλοιός (shell)

Ο κύριος σκοπός του φλοιού είναι να διαβάζει και να μεταφράζει τις εντολές μας καθώς μας συνδέει με το UNIX. Αυτό σημαίνει ότι δέχεται τις εντολές που δίνουμε στο τερματικό μας, ελέγχει τη σύνταξή τους, καλεί τις κατάλληλες εσωτερικές ή εξωτερικές εντολές του UNIX και επαναφέρει τον έλεγχο στον χρήστη όταν οι εντολές ολοκληρωθούν ορθά.

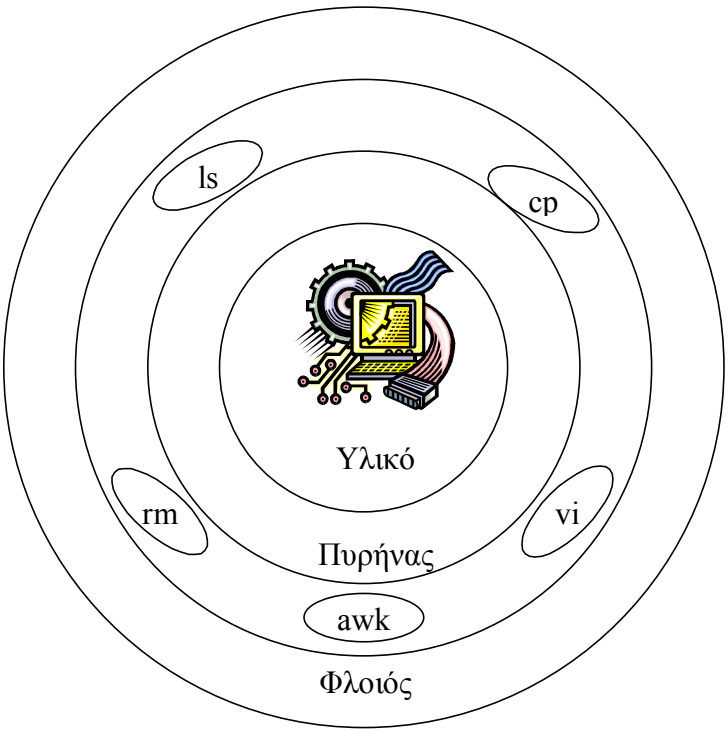
Είναι αλληλεπιδραστικό (interactive) επειδή απαντά άμεσα στις οδηγίες σας και αναφέρει λάθη και αποτελέσματα.

Ο φλοιός προσφέρει και άλλες δυνατότητες:

- Επιτρέπει στο χρήστη να φτιάξει το δικό του σύστημα (ή περιβάλλον) που να ταιριάζει στον τρόπο εργασίας του.
- Δίνει τη δυνατότητα για σύντμηση μεγάλων ονομάτων, εντολών και ακολουθιών εντολών.
- Αποθηκεύει τις παλιότερες εντολές ώστε να μπορούμε να τις επαναλαμβάνουμε, αναθεωρήσουμε ή να χρησιμοποιήσουμε ένα τμήμα από αυτές τις εντολές.
- Έχει τη δυνατότητα διαχείρισης εργασιών (job controI), ώστε να μπορούμε να δημιουργήσουμε νέες εργασίες, να τις εκτελέσουμε με ή χωρίς διαλογική επικοινωνία από το τερματικό μας και να τις τερματίσουμε.
- Μέσω του φλοιού γράφουμε προγράμματα τα οποία ονομάζονται *scripts* και εκτελούνται ισότιμα με τις ενσωματωμένες εντολές του λειτουργικού συστήματος Unix. Αυτά μπορεί απλώς να εκτελούν εντολές σειριακά ή να χρησιμοποιούν δομές όπως επαναλήψεις (loops) και επιλογή αποφάσεων (decisions). Μπορούν να έχουν ορίσματα (arguments) και να επιτρέπουν αλληλεπιδράσεις (interactions).
- Παρέχει τη δυνατότητα ανακατεύθυνσης της εισόδου, των αποτελεσμάτων και των μηνυμάτων σφάλματος εντολών, δίνοντας τη δυνατότητα ανάγνωσης εισόδου και εγγραφής αποτελεσμάτων/μηνυμάτων σφάλματος σε αρχεία, καθώς και σύνδεση εντολών μεταξύ τους προκειμένου η μία εντολή να επεξεργάζεται τα αποτελέσματα της άλλης.

Ο φλοιός του Unix βρίσκεται στο εξωτερικό επίπεδο της αλληλεπίδρασης λειτουργικού συστήματος-χρηστών, όπως φαίνεται στο σχήμα που ακολουθεί. Αξίζει να σημειωθεί ότι στα σύγχρονα συστήματα Unix η τυπική τους διανομή ενσωματώνει πάνω από έναν φλοιούς, με κυριότερους εκπροσώπους το Bourne Shell (sh), το C-

Shell (csh και την επέκτασή του tcsh), το Korn Shell (ksh), το GNU Bourne-Again Shell (bash) κ.λπ. Οι φλοιοί αυτοί έχουν διαφορές μεταξύ τους, σε ό,τι αφορά τη σύνταξη που πρέπει να χρησιμοποιηθεί για να εκτελεστούν ορισμένες ενέργειες, αλλά συνολικά προσφέρουν ισοδύναμη λειτουργικότητα.



Σχήμα 1 – Στρώματα συστήματος Unix

1.2 Ο πυρήνας του UNIX

Ο πυρήνας (kernel) του UNIX είναι η καρδιά του λειτουργικού συστήματος. Ελέγχει την προσπέλαση στον υπολογιστή και τα αρχεία, επιμερίζει τη χρήση των τμημάτων του συστήματος μεταξύ των διαφόρων διεργασιών που γίνονται μέσα στον υπολογιστή, συντηρεί το σύστημα αρχείων και διαχειρίζεται τη μνήμη του υπολογιστή. Παρ' όλο που οι απλοί χρήστες σπάνια έχουν άμεση επαφή μαζί του, ο πυρήνας είναι το κυριότερο μέρος του λειτουργικού συστήματος.

Ο διαχειριστής του συστήματος (system administrator) εκτελεί μια εργασία που λέγεται διαμόρφωση (configuration) του πυρήνα, για να προσαρμόσει τα λειτουργικά χαρακτηριστικά του συστήματος. Με τη διαμόρφωση του πυρήνα, ρυθμίζονται οι ορισμένες εσωτερικές παράμετροι του συστήματος και δημιουργούνται οι οδηγοί (drivers), οι οποίοι ελέγχουν τις περιφερειακές μονάδες που είναι συνδεδεμένες με τον υπολογιστή. Όμως, ακόμα και ένας διαχειριστής συστήματος δεν είναι σε θέση να τροποποιήσει τη δομή του πυρήνα γιατί, για να μπορέσει να το κάνει αυτό θα πρέπει να γράψει το ίδιο το UNIX από την αρχή.

1.3 Διεργασίες (processes)

Κατά τη διάρκεια της λειτουργίας του UNIX συμβαίνουν ταυτόχρονα πολλές δραστηριότητες οι οποίες ονομάζονται διεργασίες (processes). Μπορεί κανείς να φανταστεί ότι αυτές οι διεργασίες συμβαίνουν παράλληλα, παρόλο που οι περισσότεροι υπολογιστές με λειτουργικό σύστημα UNIX δεν υποστηρίζουν πραγματική παράλληλη επεξεργασία παρά μόνο για ορισμένες εξειδικευμένες εργασίες όπως είναι η εκτύπωση.

Κάθε στιγμή υπάρχει τουλάχιστον μια διεργασία του πυρήνα και μία για κάθε χρήστη που έχει μπει (έχει κάνει login) στο σύστημα. Στην πραγματικότητα για τον πυρήνα εκτελούνται πάντα πολλές διεργασίες ταυτόχρονα ενώ κάθε χρήστης μπορεί να ξεκινήσει και αυτός διεργασίες πέρα από τη διεργασία σύνδεσης. Στην πραγματικότητα, αν ο χρήστης συνδεθεί μέσα από παραθυρικό περιβάλλον (CDE, KDE, Gnome), εκτελούνται για λογαριασμό του αρκετές διεργασίες. Με την εντολή ps (process status) μπορείτε να δείτε έναν κατάλογο με όλες τις διεργασίες που τρέχουν. Για να εκτελέσετε αυτή την εντολή πληκτρολογήστε `ps -ef`.

Η διαχείριση των διεργασιών γίνεται από τον πυρήνα, ο οποίος επιτρέπει την εκτέλεσή τους διαδοχικά, ανάλογα με τις προτεραιότητες και τις ανάγκες της κάθε μίας. Μια διεργασία μπορεί να δημιουργήσει άλλες διεργασίες. Αυτή ονομάζεται τότε γονική (parent) διεργασία και εκείνες που δημιουργούνται ονομάζονται θυγατρικές (child) διεργασίες. Υπάρχουν επίσης κάποιες διεργασίες που ονομάζονται «δαίμονες» (daemons), οι οποίες παραμένουν λίγο-πολύ μόνιμα στο σύστημα γιατί εκτελούν συνεχιζόμενες εργασίες (π.χ., η διαχείριση του ταχυδρομείου, προγραμματισμών δουλειών που εκτελούνται σε τακτά χρονικά διαστήματα, δρομολογούν τα προς εκτύπωση αρχεία από την ουρά εκτύπωσης στους εκτυπωτές).

Ο χρήστης μπορεί να έχει πολλαπλά προγράμματα που να εκτελούνται ταυτόχρονα, δημιουργώντας πολλαπλές διεργασίες. Για παράδειγμα έστω ότι έχετε ένα πρόγραμμα που κάνει πολύ ώρα να ολοκληρωθεί. Αυτό το πρόγραμμα μπορείτε να το τρέξετε στο παρασκήνιο (background) με μικρή προτεραιότητα και όσο εκτελείται αυτό, εσείς να κάνετε άλλα πράγματα στο τερματικό σας. Σημειώστε ότι μερικές φορές δεν είναι απίθανο μια διεργασία που τρέχει να κολλήσει και να πρέπει να την απαλείψετε. Αν η διεργασία αυτή δεν ελέγχεται από το τερματικό σας, μπορείτε να βρείτε τον αριθμό της με την εντολή ps και να την απαλείψετε με την εντολή kill.

1.4 Το σύστημα αρχείων του UNIX

Το σύστημα αρχείων του UNIX αποτελείται από ένα σύνολο αρχείων και καταλόγων. Κάθε αρχείο/κατάλογος έχει ένα ή περισσότερα ονόματα αρχείων (file names). Υπάρχουν αρκετά είδη αρχείων με βασικότερα τα ακόλουθα:

- Τα κοινά αρχεία τα οποία περιέχουν δεδομένα

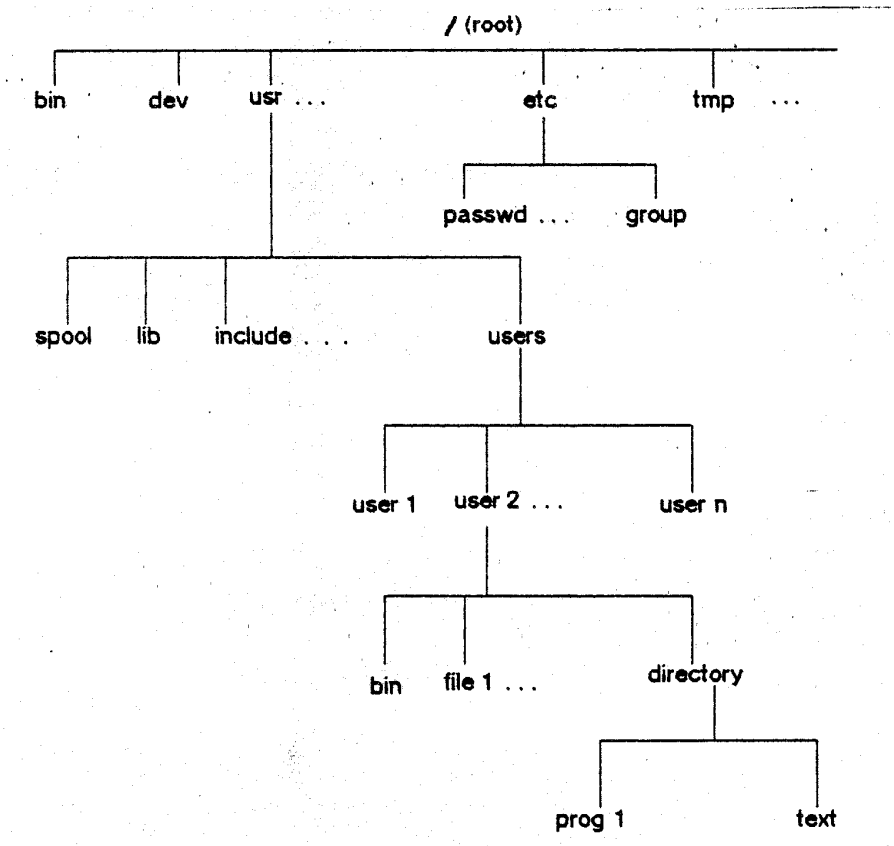
- Τα ειδικά αρχεία, με τα οποία γίνεται η προσπέλαση στις περιφερειακές μονάδες, π.χ., τα τερματικά και τους εκτυπωτές, αλλά έχουν και άλλους σκοπού. Τα ειδικά αρχεία με τη σειρά τους διαχωρίζονται σε *ειδικά αρχεία χαρακτήρων* και *ειδικά αρχεία μπλοκ*.
- Οι κατάλογοι αρχείων, που περιέχουν τις πληροφορίες που αφορούν μια ομάδα αρχείων και χρησιμεύουν για να μπορούμε να εντοπίσουμε ένα αρχείο με το όνομα του.
- Οι *σύνδεσμοι*, που επιτρέπουν την αναφορά σε αρχεία με πολλαπλά ονόματα ή/και από διαφορετικές θέσεις στην ιεραρχία καταλόγων.
- Τα *αρχεία σωληνώσεων*, που επιτρέπουν την επικοινωνία μεταξύ διεργασιών.

Το σύστημα αρχείων του UNIX είναι ιεραρχικό, διακλαδώνεται από ένα μοναδικό σημείο, τη ρίζα (root). Μπορεί να περιέχει πολλούς καταλόγους οι οποίοι με τη σειρά τους να περιέχουν υποκαταλόγους ή και αρχεία. Κάθε χρήστης του UNIX έχει το δικό του σημείο εκκίνησης (node) μέσα στο σύστημα αρχείων το οποίο καλείται *προσωπικός κατάλογος* (home directory). Οτιδήποτε κάτω από τον κατάλογο αυτό συνήθως ανήκει στον συγκεκριμένο χρήστη.

Αυτός ο κατάλογος αποτελεί και το σημείο εκκίνησης κατά την είσοδο του χρήστη στο UNIX. Όταν δηλαδή συνδεόμαστε στο σύστημα τότε πάντα μπαίνουμε στον προσωπικό μας κατάλογο, ο οποίος τίθεται και ως *κατάλογος εργασίας*. Η έννοια του καταλόγου εργασίας είναι προκειμένου να συντάσσουμε τις αναφορές μας σε αρχεία με πιο συνοπτικό τρόπο. Στο C-shell, στο TC-shell και στο bash, ο προσωπικός κατάλογος συμβολίζεται με το σύμβολο ~.

Χρησιμοποιώντας εντολές του UNIX ένας χρήστης μπορεί να δημιουργήσει αρχεία και υποκαταλόγους, διαμορφώνοντας έτσι ένα προσωπικό περιβάλλον. Στο υπόλοιπο σύστημα αρχείων (πάνω και έξω από το περιβάλλον ενός χρήστη) ο χρήστης έχει περιορισμένη πρόσβαση. Ένας χρήστης μπορεί να κινηθεί οπουδήποτε μέσα στο σύστημα αρχείων αλλάζοντας το τρέχοντα κατάλογο εργασίας.

Για να καθοριστεί ή να αναφερθεί ένα αρχείο UNIX, απλώς γράφουμε το όνομα του αρχείου όσο αυτό βρίσκεται στον τρέχοντα κατάλογο εργασίας (current working directory). Αν όμως το αρχείο βρίσκεται οπουδήποτε αλλού μέσα στο σύστημα αρχείων τότε πρέπει να προσδιοριστεί πέραν του ονόματός του και ο κατάλογος στον οποίο αυτό βρίσκεται.



Διαδρομή (path) ονομάζουμε μία ακολουθία καταλόγων που πρέπει να διασχίσουμε προκειμένου να φτάσουμε σε ένα αρχείο ή κατάλογο που επιθυμούμε. *Ο καθορισμός της θέσης* ενός αρχείου ή καταλόγου περιλαμβάνει τη διαδρομή προς το αρχείο ή κατάλογο και το όνομά του.

Το ανωτέρω σχήμα απεικονίζει τη δενδρική μορφή ενός μέρους του συστήματος αρχείων του UNIX. Μία διαδρομή μπορεί να ξεκινά είτε από τη ρίζα είτε από τον τρέχοντα κατάλογο εργασίας (current working directory).

Οι καθορισμοί αρχείων μπορεί να είναι τριών ειδών:

1) *Απόλυτος καθορισμός* (absolute specification) που πάντα ξεκινά από τη ρίζα

   π.χ. */usr/users/user2/directory/prog1*

   Προσοχή στο ότι ο πρώτος χαρακτήρας σε έναν απόλυτο καθορισμό είναι ο /

2) *σχετικός καθορισμός* (relative specification) που δείχνει πως θα φθάσουμε στον προορισμό μας από τον τρέχοντα κατάλογο εργασίας.

   Π.χ. αν ο τρέχων κατάλογος εργασίας είναι ο `/users/user2` τότε ο καθορισμός `directory/text` είναι σχετικός καθορισμός που αναφέρεται στο αρχείο `/users/user2/directory/prog1`. Ο σχετικός καθορισμός δεν ξεκινά *ποτέ* με τον χαρακτήρα /

   Μια δυνατότητα που έχουμε στους σχετικούς καθορισμούς είναι να διατρέξουμε *προς τα πάνω* την ιεραρχία των καταλόγων, μέσω του

συμβολισμού `..` (δύο τελείες χωρίς κενό ανάμεσα). Ο συμβολισμός των δύο τελειών αναφέρεται κάθε φορά στον γονικό κατάλογο. Για παράδειγμα, αν ο τρέχων κατάλογος εργασίας είναι ο `/users/user2/bin` τότε ο σχετικός καθορισμός `../directory/prog1` αναφέρεται στο αρχείο `/users/user2/directory/prog1` (με τις δύο τελείες «ανεβαίνουμε» στον κατάλογο `/users/user2` και από εκεί «κατεβαίνουμε» στον κατάλογο `directory` και εν συνεχεία πηγαίνουμε στο αρχείο `prog1`), ο καθορισμός `../../user1` αναφέρεται στον κατάλογο `/users/user1` (με τις δύο τελείες «ανεβαίνουμε» στον κατάλογο `/users/user2`, από εκεί ανεβαίνουμε με τις επόμενες δύο τελείες στον κατάλογο `/users` και από εκεί «κατεβαίνουμε» στον κατάλογο `user1`), και ο καθορισμός ../.. αναφέρεται στον κατάλογο `/users`.

3) *απλό όνομα αρχείου* (filename), που πάντα διερμηνεύεται σε σχέση με τον τρέχοντα κατάλογο εργασίας

   π.χ. αν ο τρέχων κατάλογος εργασίας είναι ο `/users/user2/directory`, ο καθορισμός `prog1` είναι καθορισμός με απλό όνομα αρχείου και αναφέρεται στο αρχείο `/users/user2/directory/prog1`

Το τελευταίο τμήμα του pathname (μετά το τελευταίο /) καλείται "ουρά" (tail) ή βασικό όνομα, και το υπόλοιπο μέρος του ονόματος διαδρομής (pathname) καλείται κεφαλή (head).

Προσοχή:

- Τα σχετικά ονόματα διαδρομής <u>ποτέ</u> δεν αρχίζουν με /
- Με το σύμβολο . συμβολίζουμε τον τρέχοντα κατάλογο.
- Με το σύμβολο .. συμβολίζουμε τον γονικό κατάλογο του τρέχοντος π.χ- *../ user1/ file2*
- Με το σύμβολο ../.. αναφέρομαι στο προηγούμενο του parent directory, δηλαδή σε δύο επίπεδα πιο πάνω.
- Σε μερικές παλιές εκδόσεις του Unix, τα ονόματα των αρχείων και καταλόγων μπορούν να φθάνουν μέχρι 14 χαρακτήρες μήκος. Στις σύγχρονες εκδόσεις, το μέγιστο μήκος είναι 256 χαρακτήρες.
- Κεφαλαία και μικρά γράμματα διαφέρουν για το σύστημα αρχείων του Unix.
- Στην ονομασία των αρχείων και καταλόγων χρησιμοποιούνται γράμματα, αριθμοί και τα σύμβολα – (παύλα), _ (κάτω παύλα), και αποφεύγονται τα ειδικά σύμβολα. Είναι επίσης καλό να αποφεύγετε τα Ελληνικά ονόματα αρχείων.

2 Βασικές εντολές UNIX

2.1 Εργασία με αρχεία και καταλόγους

Τα πάντα στο Unix απεικονίζονται σε αρχεία τα οποία είναι οργανωμένα σε καταλόγους που σχηματίζουν μία ιεραρχία. Για να δούμε σε ποιον κατάλογο βρισκόμαστε χρησιμοποιούμε την εντολή **pwd**.

Μπορούμε να κινηθούμε στην ιεραρχία των καταλόγων χρησιμοποιώντας την εντολή **cd**. Ο προκαθορισμένος κατάλογος όταν μπαίνουμε στο σύστημα έχει τη μορφή **/home/users/*login_name***, όπου ***login_name*** είναι ο κωδικός μας. Ανάλογα με τις τοπικές συμβάσεις διαχείρισης το η διαδρομή στον καθορισμό του προσωπικού μας καταλόγου μπορεί να είναι διαφορετική. Μπορούμε ανά πάσα στιγμή να μεταφερθούμε στον κατάλογο αυτό δίνοντας **cd** (χωρίς ορίσματα) ή **cd ~** (ο χαρακτήρας «~» δηλώνει πάντα τον κατάλογο του χρήστη). Μπορούμε να αναφερθούμε στον τρέχοντα κατάλογο με τον χαρακτήρα "." και στο γονικό κατάλογο με τους χαρακτήρες ".." Δίνοντας **cd *dirname*** θέτουμε ως τρέχοντα κατάλογο τον κατάλογο ***dirname***. Η θέση του ***dirname*** θεωρείται ότι είναι σχετική με τον τρέχοντα κατάλογο αν ο πρώτος χαρακτήρας δεν είναι ο **/** ή σχετική με τη ρίζα (root directory) αν ξεκινά με /

Παράδειγμα: Έστω ότι βρισκόμαστε στον κατάλογο `/home/users/user1`. Αν δώσουμε την εντολή `cd bin` θα μεταφερθούμε στον κατάλογο `/home/users/user1/bin` (εφ' όσον υπάρχει ο κατάλογος αυτός). Αν δώσουμε την εντολή `cd /bin` τότε θα μεταφερθούμε στον κατάλογο `/bin`.

Για να δημιουργήσουμε έναν κατάλογο χρησιμοποιούμε την εντολή **mkdir *dirname***.

Ένας κατάλογος περιέχει πολλά αρχεία και για να δούμε τα ονόματά τους χρησιμοποιούμε την εντολή **ls**. Η εντολή **ls** χωρίς ορίσματα είναι ισοδύναμη με την εντολή "**ls .**". Αν δώσουμε ορίσματα, τότε αν κάποιο από αυτά είναι αρχείο, η `ls` θα μας παρουσιάσει το όνομά του, ενώ αν είναι κατάλογος θα μας δώσει τα ονόματα των αρχείων και καταλόγων που υπάρχουν σε αυτόν. Στην εντολή `ls` εκτός από ορίσματα μπορούμε να παραθέσουμε και τις κάτωθι ενδείξεις:

-a : Αναγκάζει την `ls` να δείξει και τα αρχεία των οποίων το όνομα ξεκινά με τελεία. Κανονικά τα αρχεία αυτά δεν εμφανίζονται (κρυφά αρχεία)

-A : Όμοια με την `-a` αλλά δεν δείχνει τους καταλόγους "." και ".."

-d : Αν κάποιο όρισμα είναι κατάλογος τότε παρουσιάζει μόνο το όνομα του καταλόγου και όχι τα αρχεία που περιέχει

-F : Δίνει συνοπτικές πληροφορίες για τον τύπο των αρχείων: οι κατάλογοι έχουν στο τέλος του ονόματός τους ένα `/`, τα εκτελέσιμα αρχεία ένα `*` και οι συμβολικοί σύνδεσμοι ένα `@`

-l : Δίνει περισσότερες πληροφορίες για κάθε αρχείο: δικαιώματα προσπέλασης, αριθμό συνδέσμων, ιδιοκτήτη, μέγεθος και χρόνο τελευταίας τροποποίησης

-r : Η σειρά με την οποία παρουσιάζονται τα αρχεία αντιστρέφεται

-R : Εκτελεί αναδρομικά την `ls` σε κάθε κατάλογο που βρίσκει

-t : Τυπώνει τα αρχεία κατά σειρά που καθορίζεται από το χρόνο τροποποίησης.

Μπορούμε να δώσουμε περισσότερα από ένα option, αλλά πρέπει πάντα να είναι το πρώτο όρισμα.

Παράδειγμα: Δώστε την εντολή `ls -a`. Θα δείτε τα αρχεία `.login` και `.cshrc`.

Τα αρχεία `.login` και `.cshrc` είναι ειδικά αρχεία. Το `.login` εκτελείται κάθε φορά που συνδέεστε (κάνετε login) στο σύστημα. Το αρχείο `.cshrc` εκτελείται κάθε φορά που αρχίζει η εκτέλεση του φλοιού csh (C-shell) ή του φλοιού tcsh. Ο φλοιός είναι το περιβάλλον που σας επιτρέπει να επικοινωνείτε με το λειτουργικό σύστημα. Κάθε χρήστης μπορεί να επιλέξει σε ποιον φλοιό θέλει να δουλεύει. Ο εξ ορισμού φλοιός ποικίλει, ανάλογα με τις τοπικές ρυθμίσεις. Άλλοι φλοιοί είναι ο Bourne-shell (sh) (με την εκκίνηση του οποίου εκτελείται το αρχείο `.profile`, αν υπάρχει), ο ksh κλπ.

Άλλα ειδικά αρχεία που μπορεί να υπάρχουν είναι το αρχείο `.logout` που εκτελείται κάθε φορά που αποσυνδέεστε από το σύστημα, το αρχείο `.forward` που καθορίζει σε ποια διεύθυνση προωθούνται τα μηνύματά σας κ.λπ.

Για να δούμε τα περιεχόμενα ενός αρχείου κειμένου χρησιμοποιούμε την εντολή

cat [-*options*] [*filename(s)*]

(οι χαρακτήρες [] χρησιμοποιούνται για να δηλώσουν την προαιρετική ύπαρξη κάποιου ορίσματος). Options που μπορούμε να καθορίσουμε είναι τα ακόλουθα:

-b : Αγνοεί τις κενές γραμμές και τυπώνει αριθμούς γραμμών

-n : Όμοια με το `-b` αλλά δεν αγνοεί τις κενές γραμμές

-s : Τυπώνει ένα κενό όπου υπάρχουν πολλά διαδοχικά

-v : Δείχνει τους χαρακτήρες που δεν μπορούν να τυπωθούν σαν αριθμούς

Αν η **cat** δεν έχει ονόματα αρχείων στα ορίσματά της τότε διαβάζει από το *τυπικό ρεύμα εισόδου*, που συνήθως είναι το τερματικό - άρα παρουσιάζει ξανά ό,τι γράφεται στο τερματικό. Για να δηλώσετε σε κάποια εντολή ότι έχει τελειώσει η είσοδος από το τερματικό, πιέζεται `CTRL+D`. Γενικά το `CTRL+D` στο Unix δηλώνει το τέλος του αρχείου (end-of-file).

Ορισμένα αρχεία έχουν μέγεθος μεγαλύτερο απ' όσο χωράει σε μία οθόνη. Για να μπορέσουμε να δούμε τα περιεχόμενα των αρχείων αυτών χρησιμοποιούμε την εντολή **more**, η σύνταξη της οποίας είναι:

more [-n*lines*] [*filename(s)*]

Αν υπάρχει η ένδειξη `-n`*lines* (π.χ. `more -20 myfile`) το more θα τυπώσει *lines* γραμμές και μετά θα περιμένει να πιέσουμε το πλήκτρο space για να τυπώσει τις επόμενες *lines* γραμμές ή return για να τυπώσει την επόμενη γραμμή. Αν δεν υπάρχει αυτή η ένδειξη, τότε θα τυπώσει τόσες γραμμές όσες χωράνε στην οθόνη. Όπως και με την `cat` αν δεν της δώσουμε για ορίσματα ονόματα αρχείων θα παρουσιάζει ότι υπάρχει στο τυπικό ρεύμα εισόδου.

Μπορούμε να αλλάξουμε το όνομα ενός αρχείου με την εντολή **mv**. Η σύνταξή της είναι:

mv *file1 file2*

ή

mv *file1 file2 … filen directory*

Στην πρώτη περίπτωση το *file1* αλλάζει όνομα και γίνεται *file2*. Στη δεύτερη περίπτωση τα αρχεία *file1, file2 ... filen* μεταφέρονται (χωρίς να αλλάξουν όνομα) στον κατάλογο με το όνομα *directory*.

Παράδειγμα: η εντολή

mv ~/lex_programs/lex.yy.c ~/c_programs/paslex.c

μεταφέρει το αρχείο `lex.yy.c` από τον κατάλογο `~/lex_programs` στον κατάλογο `~/c_programs` και το μετονομάζει σε `paslex.c`

Μπορούμε να αντιγράψουμε αρχεία χρησιμοποιώντας την εντολή **cp**. Η σύνταξή της είναι μία από τις ακόλουθες

cp *f1 f2*

cp *f1 f2 …fn dir*

cp –r *dir1 dir2*

Στην πρώτη περίπτωση το αρχείο *f1* αντιγράφεται στο αρχείο *f2*, στη δεύτερη περίπτωση στον κατάλογο *dir* δημιουργούνται αντίγραφα των αρχείων *f1, f2, ..., fn* και στην τρίτη περίπτωση δημιουργείται ο κατάλογος *dir2* (αν δεν υπάρχει ήδη) και αντιγράφονται σε αυτόν αναδρομικά όλα τα αρχεία και οι υποκατάλογοι του καταλόγου *dir1*. Στην τελευταία αυτή περίπτωση αν το *dir2* είναι υποκατάλογος του *dir1* τότε η εντολή αυτή δεν θα τελειώσει ποτέ.

Μπορούμε να αναφερόμαστε σε ένα αρχείο (ή έναν κατάλογο) με πολλά ονόματα ή από διαφορετικούς καταλόγους, χωρίς να χρειάζεται να φτιάξουμε πολλά φυσικά αντίγραφα του αρχείου. Αυτό επιτυγχάνεται δημιουργώντας συνδέσμους σε αυτό με την εντολή **ln**, της οποίας η σύνταξη είναι

ln [-s] *f1 f2*

όπου *f1* είναι το αρχείο που υπάρχει, ενώ *f2* είναι το νέο όνομα του αρχείου, που θα δημιουργηθεί. Όποιες αλλαγές γίνονται στο ένα από τα δύο αρχεία θα αντικατοπτρίζονται στο άλλο αρχείο και (στην περίπτωση που δεν έχουμε χρησιμοποιήσει την ένδειξη `-s`) όταν σβηστεί το ένα το άλλο συνεχίζει να υπάρχει. Στην περίπτωση που θέλουμε να συνδέσουμε καταλόγους ή αρχεία που βρίσκονται σε διαφορετικά συστήματα αρχείων (πρακτικά σε διαφορετικούς διίσκους ή διαφορετικές διαμερίσεις του ίδιου δίσκου), τότε είναι υποχρεωτικό να χρησιμοποιούμε την επιλογή `-s`. Αν έχει χρησιμοποιηθεί η επιλογή `-s` και σβήσουμε το αρχικό αρχείο, ο σύνδεσμος είναι πλέον άχρηστος.

Μπορούμε να διαγράψουμε ένα ή περισσότερα αρχεία χρησιμοποιώντας την εντολή **rm**, η σύνταξη της οποίας είναι:

rm [-irf] ***f1 … fn***

η οποία σβήνει τα αρχεία *f1 … fn*. Οι επιλογές έχουν το εξής νόημα:

-i : (interactive) μας ρωτάει αν θέλουμε να προχωρήσουμε σε διαγραφή του αρχείου

-f : (force) προχωρά στη διαγραφή του αρχείου χωρίς να μας ρωτήσει

-r : (recursive) διαγράφει ολόκληρη ιεραρχία από καταλόγους

Ένας κατάλογος αν είναι άδειος και δεν είναι ο τρέχων κατάλογος μπορεί να διαγραφεί χρησιμοποιώντας την εντολή **rmdir *dirname***.

2.2 Ανακατευθύνσεις και σωληνώσεις

Οι πιο πολλές εντολές του UNIX διαβάζουν από το τυπικό ρεύμα εισόδου και γράφουν στο τυπικό ρεύμα εξόδου. Όταν ένας χρήστης δουλεύει σε κέλυφος που είναι αλληλεπιδραστικό, όπως συμβαίνει συνήθως, το τυπικό ρεύμα εισόδου και εξόδου αντιστοιχούν με το τερματικό του χρήστη. Ένα πρόγραμμα παίρνει τα στοιχεία εισόδου από το τερματικό μας (πληκτρολόγιο) και στέλνει τα στοιχεία εξόδου επίσης στο τερματικό μας (οθόνη). Επίσης, υπάρχει ένα *τυπικό ρεύμα σφαλμάτων* που χρησιμοποιείται για τα μηνύματα λαθών και για τις άλλες πληροφορίες που σχετίζονται με εκτέλεση μιας εντολής. Οι πληροφορίες που στέλνονται στο τυπικό ρεύμα σφαλμάτων καταλήγουν επίσης στο τερματικό μας.

Τα προαναφερθέντα τυπικά ρεύματα είναι στην ουσία προσδιοριστές αρχείων (file descriptors) που αναγνωρίζονται από όλες τις διεργασίες:

- **stdin** (standard input – τυπικό ρεύμα εισόδου)
- **stdout** (standard output – τυπικό ρεύμα εξόδου) και
- **stderr** (standard error – τυπικό ρεύμα σφαλμάτων)

Το **stdin** ανακατευθύνεται χρησιμοποιώντας το συμβολισμό **< *filename***, το **stdout** με το συμβολισμό **> *filename*** ή **>> *filename*** και αν θέλουμε να ενώσουμε το **stdout** και το **stderr** και να στείλουμε το αποτέλεσμα σε ένα αρχείο τότε χρησιμοποιούμε το συμβολισμό: **>& *filename*** ή **>>& *filename***. Όταν χρησιμοποιούμε τους συμβολισμούς `> filename` και `>& filename` δημιουργείται το αρχείο *filename*, ενώ αν τυχόν υπάρχει, τα περιεχόμενά του χάνονται. Αν χρησιμοποιήσουμε τους συμβολισμούς `>> filename` ή `>>& filename`, τότε η έξοδος θα προστεθεί στο τέλος του αρχείου, αν αυτό υπάρχει, ενώ αν δεν υπάρχει τότε θα δημιουργηθεί.

Για παράδειγμα, η εντολή ***cat < test > result*** αντιγράφονται τα περιεχόμενα του αρχείου test στο αρχείο result. Όπως προαναφέραμε, βάζοντας >> μπροστά από το όνομα του αρχείου, τα στοιχεία γράφονται μέσα στο αρχείο αυτό μετά το τέλος των περιεχομένων που υπάρχουν ήδη μέσα στο αρχείο. Στο προηγούμενο παράδειγμα, τα περιεχόμενα του test θα προσαρτηθούν μετά το τέλος του περιεχομένου του result με την εντολή ***cat < test >> result***

Μπορούμε επίσης να συνδέσουμε τυπικό ρεύμα εξόδου μίας διεργασίας με τυπικό ρεύμα εισόδου μίας άλλης χρησιμοποιώντας διαύλους ή σωληνώσεις (pipes). Μία σωλήνωση δηλώνεται ως

```
command1 | command2
```

Μπορούμε βέβαια να φτιάξουμε σωληνώσεις με περισσότερες εντολές, π.χ. `cmd1 | cmd2 | cmd3 | … | cmdn`. Αν σε κάποιο σημείο θέλουμε να αποθηκεύσουμε την ως τότε έξοδο σε κάποιο αρχείο αλλά και να συνεχίσουμε να τη διοχετεύουμε κατά μήκος της σωλήνωσης, μπορούμε να χρησιμοποιήσουμε την εντολή tee *filename* π.χ.

```
cmd1 | cmd2 | … | cmdm | tee my_file | cmdn
```

Στην περίπτωση αυτή η έξοδος της εντολής `cmdm` θα γραφτεί στο αρχείο `my_file` και στη συνέχεια θα περάσει ως είσοδος στην εντολή `cmdn`.

Ως παράδειγμα των παραπάνω θεωρείστε την εντολή ***grep "Smith" phones |sort***. Η εντολή αυτή καλεί το πρόγραμμα grep το οποίο επιλέγει από το αρχείο phones όλες τι εγγραφές που περιέχουν την ακολουθία χαρακτήρων "Smith" και δίνει αυτές τις εγγραφές στο τυπικό ρεύμα εξόδου του. Αυτές οι εγγραφές στη συνέχεια κατευθύνονται μέσω της σωλήνωσης στο πρόγραμμα `sort` το οποίο τις αντιμετωπίζει σαν ήταν το τυπικό ρεύμα εισόδου του. Το τελικό αποτέλεσμα είναι ένας ταξινομημένος πίνακας στην οθόνη με όλες τις γραμμές του αρχείου phones οι οποίες περιέχουν το "Smith".

Σε μία γραμμή εντολών, που περιέχει τόσο ανακατευθύνσεις όσο και σωληνώσεις, οι ανακατευθύνσεις έχουν μεγαλύτερη προτεραιότητα. Δηλαδή πρώτα γίνεται η ανακατεύθυνση των εντολών και μετά σχετίζονται με τις σωλωνώσεις. Π.χ., η γραμμή εντολών ***grep "Smith" phones | sort > hangups*** ξεδιαλέγει από το αρχείο phones τις εγγραφές που περιέχουν το "Smith" όπως και προηγουμένως, τις ταξινομεί με το sort και κατατάσσει τις διατεταγμένες εγγραφές στο αρχείο hangups.

Η δημιουργία μιας σωλήνωσης προϋποθέτει τη δημιουργία ενός ζεύγους διεργασιών. Η πρώτη διεργασία θα δημιουργήσει την πληροφορία, η οποία στη συνέχεια θα περάσει από τη σωλήνωση για να δοθεί ως είσοδος στη δεύτερη διεργασία.

2.3 Μεταχαρακτήρες

Μπορούμε να χρησιμοποιήσουμε ειδικούς χαρακτήρες για να αναφερθούμε σε αρχεία με παρόμοιο όνομα χωρίς να χρειάζεται να πληκτρολογήσουμε όλα τα ονόματα των αρχείων αυτών. Οι χαρακτήρες αυτοί είναι οι εξής:

*	:	Μία οποιαδήποτε ακολουθία χαρακτήρων
?	:	Ένας οποιοσδήποτε χαρακτήρας
[c1c2…cn]	:	Ένας χαρακτήρας από τους c1 c2 … cn
{word1, word2, …, wordn}	:	Μία οποιαδήποτε από τις λέξεις word1, word2, … wordn

Παραδείγματα:

Παράδειγμα	**Επεξήγηση**
~/*	όλα τα αρχεία στον προσωπικό κατάλογο του χρήστη
~/b*	όλα τα αρχεία στον προσωπικό κατάλογο του χρήστη που ξεκινάνε με το γράμμα b

Παράδειγμα	Επεξήγηση
*.c	όλα τα αρχεία στον τρέχοντα κατάλογο που τελειώνουν με .c
chap	όλα τα αρχεία στον τρέχοντα κατάλογο που περιέχουν τη λέξη chap
chap?.doc	όλα τα αρχεία που αρχίζουν με chap, τελειώνουν σε .doc και ανάμεσα στο chap και το .doc υπάρχει μόνο ένας χαρακτήρας.
chap[0123456789].doc	τα αρχεία chap1.doc, chap2.doc, chap3.doc, chap4.doc, chap5.doc, chap6.doc, chap7.doc, chap8.doc, chap9.doc, chap0.doc
chap[0-9].doc	το ίδιο με το προηγούμενο
/students/{cst, tst}/*	τους προσωπικούς καταλόγους όλων των χρηστών στα τμήματα υπολογιστών και τηλεπικοινωνιών

Για να δούμε τα αρχεία που ξεκινούν από ., πρέπει να δώσουμε κάποιο pattern που να αρχίζει με τελεία. Δηλαδή οι μεταχαρακτήρες δεν αντιστοιχούν ποτέ στην αρχική τελεία.

Πρέπει να τονίσουμε εδώ ότι ο φλοιός αντικαθιστά τους μεταχαρακτήρες με τα ονόματα των αρχείων που ταιριάζουν με τα patterns πριν εκτελέσει την εντολή. Έτσι για παράδειγμα αν στον κατάλογο υπάρχουν τα αρχεία `file1`, `file2`, `file3`, τότε πληκτρολογώντας `command *`, είναι το ίδιο με το να εισάγουμε `command file1 file2 file3`. Έτσι, εντολές του τύπου

```
cp *.c *.bak
```

δεν έχουν ως αποτέλεσμα την αντιγραφή όλων των αρχείων με κατάληξη `.c` σε ομώνυμα με κατάληξη `*.bak` (όπως θα είχαν π.χ. στο DOS). Αν έχουμε στον κατάλογο τα αρχεία `a.c b.c και c.c` τότε η εντολή θα είναι ισοδύναμη με `cp a.c b.c c.c` που είναι συντακτικά λάθος, ενώ αν υπάρχουν μόνο τα `a.c` και `b.c` τότε θα είναι ισοδύναμη με `cp a.c b.c`, οπότε θα καταστρέψει το αρχείο `b.c` αντιγράφοντας πάνω του το αρχείο `a.c`.

Ο φλοιός εκλαμβάνει ως λάθος την περίπτωση να μην υπάρχει κανένα αρχείο του οποίου το όνομα να ταιριάζει με τα patterns που δώσαμε και εμφανίζει το μήνυμα "No match". Αν θέλουμε να μην εκλαμβάνεται η κατάσταση αυτή ως σφάλμα πρέπει να δώσουμε την εντολή

```
set nonomatch
```

2.4 Χρήσιμες εντολές

2.4.1 Αλλαγή προνομίων προσπέλασης σε αρχεία

Η εντολή chmod χρησιμοποιείται για την αλλαγή προνομίων προσπέλασης. Η σύνταξή της είναι

`chmod` *`τριψήφιος-οκταδικός-αριθμός filename`*

ή

`chmod [user_specifier] mode_change_specifier filename`

Στην πρώτη περίπτωση υπάρχουν οι εξής κανόνες: το πρώτο ψηφίο αναφέρεται στον ιδιοκτήτη, το δεύτερο στο group και το τρίτο στους υπόλοιπους. Σε κάθε ψηφίο προσθέτουμε 4 αν θέλουμε να δώσουμε δικαίωμα ανάγνωσης, 2 αν θέλουμε να δώσουμε δικαίωμα εγγραφής και 1 αν θέλουμε να δώσουμε δικαίωμα εκτέλεσης.

Στη δεύτερη περίπτωση ο (προαιρετικός) *user_specifier* καθορίζει σε ποιον εφαρμόζεται η αλλαγή των προνομίων (u = owner, g = group, o = other) και το *mode_change_specifier* δείχνει ποια αλλαγή θα επέλθει (+r, -r, +w, -w, +x, -x).

Παραδείγματα:

`chmod 740 myfile`

δίνουμε όλα τα δικαιώματα στον ιδιοκτήτη, δικαίωμα ανάγνωσης στο group και κανένα δικαίωμα στους υπόλοιπους.

`chmod +x myfile`

δίνουμε δικαίωμα εκτέλεσης σε όλους (ή μόνο στον ιδιοκτήτη) στο αρχείο myfile. Το δικαίωμα εκτέλεσης σε έναν κατάλογο έχει το νόημα ότι μπορούμε να κινηθούμε μέσα σε αυτό.

`chmod o-w myfile`

αφαιρούμε το δικαίωμα εγγραφής από τους χρήστες που δεν ανήκουν στο δικό μου group από το αρχείο myfile.

Με τις εντολές chown και chgrp μπορούμε να μεταβιβάσουμε σε άλλους την ιδιοκτησία ενός χρήστη ή μιας ομάδας χρηστών πάνω σε μια ομάδα αρχείων και καταλόγων. Η σύνταξη των εντολών αυτών έχει τη μορφή:

chown owner file

chgrp group file

όπου file είναι μια ομάδα αρχείων και καταλόγων που θέλετε να αλλάξετε την ιδιοκτησία τους.

2.4.2 Σύγκριση αρχείων

Η εντολή `cmp f1 f2` συγκρίνει τα αρχεία f1 και f2. Η εντολή `diff f1 f2` κάνει την ίδια δουλειά με την cmp αλλά δίνει διαφορετικού τύπου έξοδο. Με την cmp μπορείτε να εξακριβώσετε πολύ γρήγορα αν υπάρχουν διαφορές μεταξύ των αρχείων αλλά δεν παίρνετε πολλές πληροφορίες για τις διαφορές τους. Αντίθετα με το diff δημιουργείται μια όσο το δυνατό πιο σύντομη λίστα με τις διαφορές και οδηγίες που δείχνουν τι πρέπει να γίνει στο ένα αρχείο για να μετασχηματιστεί στο άλλο.

2.4.3 Εντολές πληροφοριών & βασικών πράξεων με αρχεία

`date`: Δίνει την ημερομηνία και την ώρα

`who`: Δίνει ποιοι χρήστες είναι συνδεδεμένοι στον υπολογιστή

`finger`: Δίνει πληροφορίες για κάποιο συγκεκριμένο χρήστη ή όλους τους χρήστες που έχουν συνδεθεί στο σύστημα.

`file filename(s)`: Δίνει πληροφορίες για το περιεχόμενο του αρχείου

`head -no_lines [filename]`: Τυπώνει τις *no_lines* πρώτες γραμμές του αρχείου *filename*

`tail -no_lines [filename]`: Τυπώνει τις *no_lines* τελευταίες γραμμές του αρχείου *filename*

`tail +no_lines [filename]`: Παραλείπει τις γραμμές *1 – no_lines* και τυπώνει τις υπόλοιπες.

`wc [-lwc] [filename(s)]`: μετρά γραμμές, λέξεις ή χαρακτήρες ενός αρχείου (l = lines, w = words, c = characters)

`echo [-n] string`: Τυπώνει τη συμβολοσειρά. Αν υπάρχει το `-n` τότε δεν αλλάζει γραμμή. Χρήσιμο σε προγράμματα φλοιού.

`sort [-nrtxu] [+pos] [filename(s)]`: Ταξινομεί τις γραμμές των αρχείων. Οι ενδείξεις έχουν την ακόλουθη σημασία

`-n` : θεωρεί ότι τα πεδία σύγκρισης είναι αριθμοί και όχι συμβολοσειρές

`-r` : αντιστρέφει τη σειρά ταξινόμησης

`-t`*`x`* : ο χαρακτήρας *x* θεωρείται ως διαχωριστής πεδίων

`-u` : αν υπάρχουν διπλές γραμμές τυπώνει τη μία από αυτές

`+pos` : όπου το *pos* έχει τη μορφή m.n. Το m είναι ο αριθμός πεδίων που θα παραληφθούν (μετρώντας από την αρχή της γραμμής) και n είναι ο αριθμός των χαρακτήρων από την αρχή του επόμενου πεδίου που θα παραλειφθούν. Για τη σύγκριση χρησιμοποιείται η υπόλοιπη γραμμή.

`tr [-cd] str1 [str2]`: Αλλάζει τους χαρακτήρες του *str1* στους αντίστοιχους του *str2*. Αν δοθεί η ένδειξη `-c` τότε αλλάζουν οι χαρακτήρες που ***δεν*** βρίσκονται στο *str1*. Αν υπάρχει η ένδειξη `-d` τότε οι χαρακτήρες του *str1* διαγράφονται. Η εντολή δεν δέχεται ως ορίσματα ονόματα αρχείων αλλά λαμβάνει είσοδο από το τυπικό ρεύμα εισόδου και δίνει έξοδο στο τυπικό ρεύμα εξόδου. Τα `str1` και `str2` μπορούν να περιέχουν συμβολισμούς της μορφής $[c_1\text{-}c_2]$ οι οποίοι υποδηλώνουν όλους τους χαρακτήρες μεταξύ c_1 και c_2 π.χ. ο συμβολισμός `0-9` υποδηλώνει τους χαρακτήρες `0123456789`.

`find pathname_list options_and_commands`: Ψάχνει στη λίστα των διαδρομών που δίνουμε όλη την υπο-ιεραρχία των καταλόγων για να βρει αρχεία που πληρούν κάποιες προϋποθέσεις και εκτελεί κάποιες εντολές. Η αποτίμηση της λίστας των ενδείξεων (options) γίνεται από αριστερά προς τα δεξιά και σταματά όταν κάποια αποτύχει. Υπάρχουν οι ακόλουθες επιλογές:

`-name fname` : Επιτυγχάνει αν το όνομα του αρχείου ταιριάζει με το pattern *fname*

`-type c` : Οπου c = d ή c = f ή c = l. Επιτυγχάνει αν το αρχείο είναι κατάλογος (d), απλό αρχείο (f) ή σύνδεσμος (l)

`-size [+ ή -] size` : Επιτυγχάνει αν το αρχείο έχει μέγεθος *size*. Αν υπάρχει το + τότε το αρχείο πρέπει να είναι μεγαλύτερο από *size*, ενώ αν υπάρχει το – τότε πρέπει να είναι μικρότερο από *size*. Το μέγεθος μετράται σε μπλοκ των 512 bytes

`-print` : Τυπώνει το όνομα του αρχείου – επιτυγχάνει πάντα

`-exec` *command* : Εκτελεί την εντολή *command* και επιτυγχάνει αν αυτή επιστρέψει τον κωδικό 0. Στο τέλος της εντολής πρέπει να βάλουμε τους χαρακτήρες {}\; ενώ το όνομα του αρχείου που βρέθηκε δηλώνεται με τους χαρακτήρες {}

`-OK` *command* : Ομοια με την exec αλλά ζητά επιβεβαίωση πριν προχωρήσει στην εκτέλεση της εντολής

`grep [-v]` *regular_expression* `[`*filename(s)*`]`**:** Ψάχνει για μία κανονική έκφραση (*regular expression*) μέσα στα αρχεία ή στο τυπικό ρεύμα εισόδου. Η κανονική έκφραση αποτελείται από τα εξής συστατικά:

`c`	:	Ο χαρακτήρας c ταιριάζει με τον εαυτό του
`^`	:	Αρχή της γραμμής
`$`	:	Τέλος της γραμμής
`.`	:	Οποιοσδήποτε χαρακτήρας
`[c1c2c3 … cn]`	:	Οποιοσδήποτε από τους χαρακτήρες c1 c2 c3 … cn
`[^c1c2c3 … cn]`	:	Οποιοσδήποτε χαρακτήρας ΕΚΤΟΣ από τους c1, c2, c3, …, cn
`r*`	:	Μηδέν ή περισσότερες εμφανίσεις της κανονικής έκφρασης r
`r+`	:	Μία ή περισσότερες εμφανίσεις της κανονικής έκφρασης r
`r?`	:	Μηδέν ή μία εμφανίσεις της κανονικής έκφρασης r
`r1r2`	:	Η κανονική έκφραση r1, ακολουθούμενη από την κανονική έκφραση r2

Η grep διαβάζει διαδοχικά τις γραμμές των αρχείων και τυπώνει αυτές που περιέχουν την κανονική έκφραση. Αν υπάρχει το option –v τότε τυπώνει τις γραμμές που ***δεν*** περιέχουν την κανονική έκφραση. Υπάρχει και η εντολή egrep, με σύνταξη παρόμοια με την grep, αλλά επιτρέπει ομαδοποίηση των κανονικών εκφράσεων με παρενθέσεις και υποστηρίζει και τον τελεστή διάζευξης που είναι ο χαρακτήρας | (δηλαδή το r1 | r2 σημαίνει η κανονική έκφραση r1 ή η κανονική έκφραση r2).

`sed [-n]` *command(s)* `[`*filename(s)*`]`**:** Διαβάζει το περεχόμενο των αρχείων και εφαρμόζει σ' αυτό μετασχηματισμούς που ορίζονται από τις εντολές (commands). Όλες οι γραμμές τυπώνονται, εκτός αν παρατεθεί η ένδειξη *–n*,οπότε τυπώνονται μόνο οι γραμμές για τις οποίες προσδιορίζεται εντολή εκτύπωσης. Η κάθε εντολή έχει τη γενική μορφή:

```
qualifier action
```

όπου ο *επιλογέας* (qualifier) καθορίζει αν θα εκτελεστεί η ενέργεια (action) στη γραμμή ή όχι. Οι επιλογείς προσδιορίζονται ως εξής:

Τύπος επιλογέα	**Παράδειγμα**	**Αποτέλεσμα**
Αριθμός γραμμής	`10`	Η $10^{η}$ γραμμή του αρχείου

Τύπος επιλογέα	**Παράδειγμα**	**Αποτέλεσμα**
Περιοχή γραμμών	`10-30`	Οι γραμμές του αρχείου από τη $10^{η}$ έως την $30^{η}$
Κανονική έκφραση	`/abcd/` `/def.*z/`	Οι γραμμές που ταιριάζουν με την κανονική έκφραση (περιέχουν τη συμβολοσειρά *abcd* ($1^{ο}$ παράδειγμα ή τη συμβολοσειρά *def* ακολουθούμενη από οποιουσδήποτε χαρακτήρες και τον χαρακτήρα *z* ($2^{ο}$ παράδειγμα)
Περιοχή κανονικών εκφράσεων	`/abcd/-/xyz/`	Βρίσκουμε στο αρχείο τη γραμμή που ταιριάζει με την $1^{η}$ κανονική έκφραση. Από το σημείο αυτό και μέχρι να βρούμε τη $2^{η}$ κανονική έκφραση, όλες οι γραμμές ταιριάζουν, συμπεριλαμβανομένης της γραμμής που ταιριάζει με τη $2^{η}$ κανονική έκφραση. Αν μετά την εμφάνιση της $2^{ης}$ κανονικής έκφρασης εμφανιστεί εκ νέου η $1^{η}$, ο κύκλος ταιριάσματος συνεχίζεται.
$	`$`	Η τελευταία γραμμή του αρχείου

Μεικτή περιοχή	`1-/abcd/`
	`/abcd/-20`
	`100-$`
	`/xyz/-$`
		Βρίσκουμε τη γραμμή που ταιριάζει με τον $1^{ο}$ επιλογέα. Από το σημείο αυτό και μέχρι να βρούμε τη γραμμή που ταιριάζει με τον $2^{ο}$ επιλογέα, όλες οι γραμμές ταιριάζουν

Καθόλου επιλογέας		Όλες οι γραμμές ταιριάζουν

Η *ενέργεια* (action) καθορίζεται ως ακολούθως:

Ενέργεια	**Παράδειγμα**	**Αποτέλεσμα**
`d`	`d`	Η γραμμή διαγράφεται (δεν εμφανίζεται στην έξοδο)
`p`	`p`	Η γραμμή τυπώνεται
`w filename`	`w myfile`	Η γραμμή προστίθεται στο τέλος του αρχείου *filename*
`q`	`q`	(quit) Η εκτέλεση του sed τερματίζεται – οι υπόλοιπες γραμμές δεν εξετάζονται.
`=`	`=`	Τυπώνεται ο αριθμός γραμμής
`y/str1/str2/`	`y/abcd/ABCD/`	Οι χαρακτήρες της συμβολοσειράς *str1* αντικαθίστανται με τους χαρακτήρες της συμβολοσειράς *str2*. Η ενέργεια του παραδείγματος θα αντικαταστήσει τους χαρακτήρες a, b, c, d με τους αντίστοιχους κεφαλαίους.

Ενέργεια	Παράδειγμα	Αποτέλεσμα
`s/old/new/`	`s/seven/eight/`	Η συμβολοσειρά *old* αντικαθίσταται από τη συμβολοσειρά *new*. Η ενέργεια του παραδείγματος αντικαθιστά *την 1η εμφάνιση της συμβολοσειράς seven*, σε κάθε γραμμή, με τη συμβολοσειρά *eight*. Μετά τη 2η κάθετο μπορούμε να παραθέσουμε τους κάτωθι προσδιοριστές: • *g* για να αντικατασταθούν *όλες* οι εμφανίσεις της 1ης συμβολοσειράς με τη 2η • *p* για να τυπωθεί η γραμμή • *w filename* Η γραμμή προστίθεται στο τέλος του αρχείου *filename* Οι προσδιοριστές μπορούν να συνδυαστούν, π.χ. *gp*.

`alias str word(s)`: Αν δώσουμε την εντολή *str* αυτή θα αντικατασταθεί με τα *word(s)*. Η δυνατότητα χρήσης ψευδωνύμων αναπτύσσεται εκτενέστερα σε επόμενο εδάφιο.

2.5 Μεταβλητές

2.5.1 Απλές μεταβλητές

Το C-shell υποστηρίζει μεταβλητές. Ο χειρισμός των μεταβλητών είναι κυρίως προσανατολισμένος σε τιμές συμβολοσειρών, αν και υπάρχουν δυνατότητες για χειρισμό ακέραιων τιμών. Η απόδοση τιμής σε μεταβλητή γίνεται με την εντολή set:

```
set varname=value
```

Με την εντολή αυτή στη μεταβλητή `varname` αποδίδεται η τιμή `value`. Η απόδοση τιμής είναι ταυτόχρονα και δήλωση της μεταβλητής – δεν είναι απαραίτητο να δηλωθεί μια μεταβλητή πριν τη χρήση της, ούτε υπάρχει σχετικός μηχανισμός.

Για να αναφερθούμε στην τιμή μιας μεταβλητής χρησιμοποιούμε τον συμβολισμό

```
$varname
```

Έτσι η ακολουθία εντολών

```
set myMessage=hello
echo $myMessage
```

θα τυπώσει στην οθόνη τη συμβολοσειρά `hello` καθώς η πρώτη εντολή αποδίδει στη μεταβλητή `myMessage` την τιμή `hello` και η δεύτερη εντολή τυπώνει αυτή την τιμή.

2.5.2 Μεταβλητές λίστας λέξεων

Οι μεταβλητές που ορίζονται στο C-shell με τον ανωτέρω τρόπο έχουν *απλές τιμές*. Το C-shell υποστηρίζει επίσης και μεταβλητές τύπου *λίστας λέξεων* (ένα είδος πίνακα) που ορίζονται με τον ακόλουθο τρόπο:

```
set varname=(value1 value2 … valuen)
```

π.χ.

```
set fruit=(apple orange apricot melon)
```

Για τις μεταβλητές τύπου λίστας λέξεων, ο συμβολισμός `$varname` αναφέρεται στη συνολική τιμή της μεταβλητής, όλες δηλαδή τις λέξεις που την αποτελούν. Μπορούμε όμως για τις μεταβλητές αυτές να χρησιμοποιήσουμε και τους κάτωθι πρόσθετους συμβολισμούς:

Συμβολισμός	**Παράδειγμα**	**Σχόλια**
`$varname[index]`	`$fruit[2]`	Το index είναι αριθμητικός δείκτης. Το αποτέλεσμα είναι η υπ' αριθμόν index λέξη της τιμής (η αρίθμηση ξεκινά από το 1). Αν το index είναι μεγαλύτερο από το πλήθος λέξεων της μεταβλητής, η τιμή της έκφρασης είναι η κενή συμβολοσειρά.
`$#varname`	`$#fruit`	Δίνει το πλήθος των λέξεων που απαρτίζουν την τιμή της μεταβλητής.
`$varname[index1-index2]`	`$fruit[2-3]`	Τα index1 και index2 είναι αριθμητικοί δείκτες με index1 ≤ index2. Το αποτέλεσμα είναι οι λέξεις με αρίθμηση από index1 έως index2 (το index2 συμπεριλαμβάνεται). Αν το index1 παραλειφθεί εννοείται το 1, αν το index2 παραλειφθεί εννοείται το $#varname.

Για τις μεταβλητές τύπου λίστας λέξεων μπορεί να χρησιμοποιηθεί η εντολή

```
shift varname
```

η οποία έχει ως αποτέλεσμα να αποβάλλεται η πρώτη λέξη της τιμής της μεταβλητής δηλ. η νέα τιμή της μεταβλητής να απαρτίζεται από τις λέξεις με αρίθμηση 2, 3 κ.λπ. Η ανωτέρω εντολή είναι ισοδύναμη με την εντολή

```
set varname=($varname[2-])
```

2.5.3 Μεταβλητές αριθμητικού τύπου

Οι μεταβλητές αριθμητικού τύπου απαιτούν στο C-shell ειδικό χειρισμό. Μπορούμε να τους αποδώσουμε τιμή με την εντολή set, ως ανωτέρω, αλλά για να εκτελέσουμε αριθμητικές πράξεις σ' αυτές απαιτείται στη θέση του set να χρησιμοποιήσουμε τον χαρακτήρα `@`. Έτσι, η ακολουθία εντολών

```
set x=2
@ x=$x+1
```

αποδίδει στη μεταβλητή `x` την τιμή 3, ενώ η ακολουθία εντολών

```
set x=2
```

```
set x = $x + 1
```

θα αποδώσει ως τιμή στη μεταβλητή `x` τη συμβολοσειρά `2+1`. Η εντολή `@` υποστηρίζει τους ακόλουθους τελεστές:

```
+ (μοναδιαίο) + (δυαδικό) - (μοναδιαίο) - (δυαδικό)* / % (υπόλοιπο)
&& (λογική σύζευξη) || (λογική διάζευξη)
>> (δεξιά ολίσθηση) << (αριστερή ολίσθηση) ~ (συμπλήρωμα ως προς 1)
& (σύζευξη επιπέδου bit) | (διάζευξη επιπέδου bit)
^ (αποκλειστική διάζευξη επιπέδου bit)
++ (αύξηση) -- (μείωση)
+= -= *= /=
```

Σημειώσεις

1. το κενό είναι απαραίτητο μεταξύ του χαρακτήρα `@` και του ονόματος της μεταβλητής.
2. Τα κενά μεταξύ του ονόματος της μεταβλητής και του «=» καθώς και μεταξύ του «=» και της έκφρασης είναι προαιρετικά.
3. Τα τμήματα της έκφρασης πρέπει να διαχωρίζονται με κενά π.χ. γράφουμε `@ x=2 + 1` και όχι `@ x=2+1`
4. Αν η έκφραση περιέχει τους χαρακτήρες <, >, `&` ή `|` πρέπει όλη η έκφραση (ή τουλάχιστον το τμήμα που περιέχει τους χαρακτήρες) να εσωκλειστεί σε παρενθέσεις.

2.5.4 Διαχείριση μεταβλητών

Αν δώσουμε ως τιμή την κενή συμβολοσειρά σε μία μεταβλητή με την εντολή

```
set x=
```

τότε η μεταβλητή εξακολουθεί να υπάρχει έχοντας ως τιμή την κενή συμβολοσειρά. Για να πάψει συνολικά να υπάρχει η μεταβλητή, δίνουμε την εντολή

```
unset x
```

Για να ελέγξουμε αν μία μεταβλητή υπάρχει ή όχι, μπορούμε να χρησιμοποιήσουμε την έκφραση

```
$?x
```

που αποτιμάται σε 1 (αληθές) αν η μεταβλητή έχει ορισθεί και σε 0 (ψευδές) αν η μεταβλητή δεν έχει ορισθεί.

Δίνοντας την εντολή

```
set
```

ή

```
@
```

χωρίς ορίσματα, τυπώνονται όλες οι μεταβλητές που έχουν ορισθεί και οι αντίστοιχες τιμές τους.

2.5.5 Ειδικές μεταβλητές και συμβολισμοί

Υπάρχουν κάποιες μεταβλητές που έχουν ειδική σημασία για τον φλοιό:

Μεταβλητή	**Σημασία**
`path`	Λίστα λέξεων που υποδεικνύει σε ποιους καταλόγους θα ψάξει το σύστημα για να βρει τις εντολές που εισάγουμε
`prompt`	Το σύμβολο που τυπώνει ο φλοιός για να μας δείξει ότι είναι έτοιμος να δεχθεί τις εντολές μας.
`cwd`	Τρέχων κατάλογος εργασίας
`status`	Κωδικός εξόδου της τελευταίας εντολής
`argv`	Οι παράμετροι που έχουν μεταβιβασθεί στον φλοιό – χρήσιμο σε προγράμματα φλοιού

Χρήσιμοι είναι επίσης οι κάτωθι συμβολισμοί:

Μεταβλητή	**Σημασία**
`$0`	Σε προγράμματα φλοιού είναι το όνομα του προγράμματος
`$1, $2, …`	Σε προγράμματα φλοιού αναπαριστούν την $1^η$, $2^η$, κ.ο.κ. παράμετρο. Η πιο σωστή αναπαράσταση στο C-Shell είναι `$argv[1]`, `$argv[2]` κ.ο.κ.
`$$`	Η ταυτότητα της τρέχουσας διεργασίας
`$<`	Διαβάζει μια γραμμή από την είσοδο (συνήθως τερματικό) – π.χ. η εντολή `set x=$<` διαβάζει μία συμβολοσειρά από την είσοδο και την καταχωρεί ως τιμή της μεταβλητής x

2.5.6 Μεταβλητές φλοιού έναντι μεταβλητών περιβάλλοντος

Είναι επίσης σημαντικό να γνωρίζουμε ότι αν μέσα από ένα πρόγραμμα φλοιού καλέσουμε ένα άλλο, οι μεταβλητές που έχουμε ορίσει με εντολές `set` και `@` στο $1^ο$ πρόγραμμα, *δεν θα είναι ορατές στο $2^ο$ πρόγραμμα.* Οι μεταβλητές που ορίζονται με `set` και `@` είναι ορατές *μόνο* στο φλοιό που τους ορίζει. Για να ορίσουμε μεταβλητές που θα είναι ορατές *και στα προγράμματα που καλούμε* πρέπει αντί για `set` ή `@` να χρησιμοποιήσουμε την εντολή `setenv`, της οποίας η σύνταξη είναι:

```
setenv varname value
```

χωρίς = μεταξύ του ονόματος και της τιμής. Η setenv δεν υποστηρίζει αριθμητικές πράξεις – αν επιθυμούμε αριθμητικές πράξεις πρέπει πρώτα να υπολογίσουμε την τιμή με τον τελεστή `@` και στη συνέχεια να χρησιμοποιήσουμε τη `setenv` π.χ.

```
@ x=2 + 1
setenv RESULT $x
```

Για να πάψει να υπάρχει μία μεταβλητή που έχουμε ορίσει με `setenv` χρησιμοποιούμε την εντολή `unsetenv` π.χ.

```
unsetenv RESULT
```

Κατά σύμβαση, χρησιμοποιούμε πεζούς χαρακτήρες για τις μεταβλητές που ορίζονται με `set` και `@` (*κοινές μεταβλητές*) και κεφαλαίους χαρακτήρες για τις μεταβλητές που ορίζονται με `setenv` (*μεταβλητές περιβάλλοντος*).

2.5.7 Απομόνωση ονομάτων μεταβλητών

Για όλες της μεταβλητές μπορεί να χρησιμοποιηθεί ο συμβολισμός

```
${varname}
```

για να διαχωρίσει σαφώς το όνομα της μεταβλητής από τα λοιπά συμφραζόμενα του προγράμματος. Για παράδειγμα, έστω ότι έχουμε ορίσει μία μεταβλητή με το όνομα `word` που περιέχει μία αγγλική λέξη στον ενικό και θέλουμε να προσθέσουμε σ' αυτή τον χαρακτήρα `s` για να σχηματίσουμε τον πληθυντικό της. Η σύνταξη

```
set plural=$words
```

δεν θα έχει το επιθυμητό αποτέλεσμα διότι ο φλοιός θα ψάξει για τη μεταβλητή `words`. Αντιθέτως η σύνταξη

```
set plural=${word}s
```

θα μας δώσει το επιθυμητό αποτέλεσμα.

2.5.8 Μεταβλητές και μεταχαρακτήρες

Η συμβολοσειρά που αποδίδουμε ως τιμή σε μία μεταβλητή μπορεί να περιέχει μεταχαρακτήρες (`*, ?, [], {}`). Στην περίπτωση αυτή ο φλοιός θα αποδώσει ως τιμή στη μεταβλητή μία *λίστα λέξεων*, ανεξάρτητα αν έχουμε περικλείσει την τιμή σε παρενθέσεις. Κάθε στοιχείο της λίστας λέξεων θα είναι και ένα όνομα αρχείου που θα ταιριάζει με τον μεταχαρακτήρα.

2.6 Μηχανισμός Ιστορικού

Ο φλοιός απομνημονεύει τις πιο πρόσφατες εντολές που έχετε δώσει (το πλήθος τους ορίζεται μέσω της μεταβλητής `history`) τις οποίες μπορείτε να επανεκτελέσετε, χωρίς να χρειάζεται να τις πληκτρολογήσετε. Αυτό γίνεται με τους εξής συμβολισμούς:

`!!` : η αμέσως προηγούμενη εντολή

`!n` : η εντολή n

`!-n` : η εντολή που δώσατε πριν από n πατήματα του "Enter"

`!str` : η πιο πρόσφατη εντολή που αρχίζει με `str`

`!?str` : η πιο πρόσφατη εντολή που περιέχει το `str`

Μπορούμε να συμπληρώσουμε τις εντολές αυτές με κάποια συμβολοσειρά, πληκτρολογώντας τη στο τέλος των συμβολισμών. Π.χ. αν δώσουμε `ls` και μετά `!! bin`, η τελευταία εντολή θα είναι ισοδύναμη με `ls bin`.

Η εντολή history μας τυπώνει τις εντολές που έχει απομνημονεύσει ο φλοιός.

2.7 Προστασία ειδικών χαρακτήρων

Αρκετοί χαρακτήρες (π.χ. `*`. ?, `[`, `]`, `{`, `}`, `$`, `!`, ~, κενό) έχουν ειδική σημασία για το φλοιό. Για να αναιρέσουμε την ειδική σημασία αυτών μπορούμε:

- να βάλουμε μπροστά τους μία αντίστροφη κάθετο - backslash (`\`)
- να τους εσωκλείσουμε σε μονά εισαγωγικά (`' '`)
- να τους εσωκλείσουμε σε διπλά εισαγωγικά (`" "`)

Οι χαρακτήρες (`\`, `'`, `"`) είναι επίσης ειδικοί και πρέπει να προστατεύονται. Τα μονά εισαγωγικά επιτρέπουν σε όλους τους χαρακτήρες που εσωκλείουν να περνάνε χωρίς μετάφραση. Το ίδιο ισχύει και για τα διπλά εισαγωγικά με την εξαίρεση ότι αντικαθίστανται οι συμβολισμοί `$...` που εσωκλείονται, και επίσης αντικαθίστανται και οι αντίστροφες κάθετοι. Τα εισαγωγικά χρησιμεύουν επίσης για να μεταβιβάζουμε πολλές λέξεις σαν μία παράμετρο. Π.χ. αν θέλουμε να βρούμε τις γραμμές που περιέχουν τη συμβολοσειρά “Hello there” στο αρχείο myfile θα πρέπει να δώσουμε την εντολή

```
grep "Hello there" myfile
```

γιατί η εντολή

```
grep Hello there myfile
```

θα έχει ως αποτέλεσμα να αναζητηθεί η συμβολοσειρά Hello στα αρχεία there και myfile.

Ειδικοί χαρακτήρες είναι ακόμα οι `(`, `)`, `´`, `|`, >, <, `&`, `` ` ``, tab.

2.8 Εκτέλεση διεργασιών στο φλοιό

Μπορούμε να δώσουμε περισσότερες από μία εντολές σε μια μόνο γραμμή, χωρίζοντάς της με ελληνικά ερωτηματικά (`;`). Οι εντολές αυτές θα εκτελεστούν σειριακά. Αν αντί του `;` χρησιμοποιήσουμε το `&&`, τότε η δεύτερη εντολή θα εκτελεστεί μόνο αν επιτύχει η πρώτη (επιστρέψει αναφορά κατάστασης 0), ενώ αν χρησιμοποιήσουμε `||` τότε η δεύτερη εντολή θα εκτελεστεί αν η πρώτη αποτύχει (επιστρέψει αναφορά κατάστασης διάφορο από μηδέν).

Μπορούμε να καθορίσουμε ότι θέλουμε μία διεργασία να τρέχει στο παρασκήνιο βάζοντας στο τέλος της εντολής το χαρακτήρα `&` (π.χ. `command &`). Οι διεργασίες που εκτελούνται στο παρασκήνιο δεν μπορούν να ζητήσουν είσοδο από το τερματικό (αν προσπαθήσουν η εκτέλεσή τους αναστέλλεται), μπορούν όμως να τυπώσουν στο τερματικό. Αν δεν επιθυμούμε έξοδο από διεργασίες παρασκηνίου να εμφανίζονται στην οθόνη μας, δίνουμε την εντολή

```
stty tostop
```

Αφού δώσουμε την εντολή αυτή, διεργασίες παρασκηνίου που επιχειρούν να τυπώσουν στο τερματικό αναστέλλονται και πρέπει να συνεχιστούν στο προσκήνιο. Για να επιτρέψουμε ξανά στις διεργασίες παρασκηνίου να τυπώνουν στο τερματικό δίνουμε την εντολή

```
stty -tostop
```

Μπορούμε να σταματήσουμε τη διεργασία που τρέχει στο προσκήνιο διατηρώντας το δικαίωμα να τη συνεχίσουμε μετά από εκεί που σταμάτησε πατώντας Ctrl+Z.

Υπάρχουν οι εξής εντολές ελέγχου διεργασιών:

`jobs` : μας δείχνει τις διεργασίες του παρασκηνίου

`ps` : μας δείχνει όλες τις διεργασίες. Μπορεί να πάρει διάφορα ορίσματα που διαφέρουν σε συστήματα BSD και System V

`kill [-`*`signal`*`] `*`job_specifier`*: στέλνει ένα σήμα *(signal)* στη διεργασία που καθορίζουμε. Το εξ ορισμού σήμα είναι το `TERM` (terminate), ενώ άλλα χρήσιμα signals είναι τα `HUP` (hangup, δηλαδή logout αν αναφερόμαστε στο φλοιό, ή επανεκκίνηση της διεργασίας για άλλες), `KILL`, `CONT` (continue), `STOP` και `QUIT`.

`fg %`*`job_no`*: Συνεχίζει στο προσκήνιο την εργασία *job_no*

`bg %`*`job_no`*: Συνεχίζει την εκτέλεση της εργασίας *job_no* στο παρασκήνιο

`nohup `*`command`*` &`: Συνεχίζει την εκτέλεση της εντολής *command* ακόμη και αν αποσυνδεθούμε από το σύστημα

`wait`: περιμένει να τελειώσουν όλες οι διεργασίες στο παρασκήνιο

Ο *job_specifier* που αναφέρεται στην εντολή `kill` μπορεί να είναι είτε *`process_id`* (τον αριθμό που δίνει η `ps`) είτε `%`*`job_no`* (τον αριθμό που επιστρέφει η `jobs`). Εναλλακτικά αντί για `%`*`job_no`* μπορούμε να χρησιμοποιούμε τους συμβολισμούς `%`*`str`* ή `%?`*`str`* που αντιστοιχούν στην εργασία της οποίας η γραμμή εντολών αρχίζει ή περιλαμβάνει, αντίστοιχα, το *str*, ή τους συμβολισμούς `%%` και `%+` που αναφέρονται στην τρέχουσα εργασία ή `%-` που αναφέρεται στην προηγούμενη εργασία.

Για να μας ειδοποιεί το C-shell για τις αλλαγές στην κατάσταση μίας διεργασίας που εκτελείται στο παρασκήνιο χρησιμοποιούμε την εντολή `notify %`*`job_no`*. Αν δώσουμε `set notify` τότε το C-shell θα μας ειδοποιεί για τις αλλαγές στην κατάσταση όλων των διεργασιών του background. Το αποτέλεσμα της `set notify` αίρεται με την εντολή `unset notify`.

2.9 Μηχανισμός ψευδωνύμων

Ο φλοιός C-shell μας δίνει τη δυνατότητα να ορίζουμε συμβολικά ονόματα για εντολές ή ακολουθίες εντολών, μέσω του μηχανισμού ψευδωνύμων. Τα ψευδώνυμα χρησιμοποιούνται από τον χρήστη ισοδύναμα προς όλες τις εντολές του συστήματος και μπορούν επίσης να συνδυαστούν σε ακολουθίες εντολών, σωληνώσεις, να εκτελεστούν στο παρασκήνιο, να περιέχουν μεταβλητές κ.ά., ενώ μπορούν τέλος να περιέχουν και άλλα ψευδώνυμα.

Για να ορίσουμε ένα ψευδώνυμο (alias) χρησιμοποιούμε την εντολή `alias` ως ακολούθως:

```
alias name definition
```

όπου *name* είναι το ψευδώνυμο που επιθυμούμε να ορίσουμε και *definition* οι εντολές στις οποίες αντιστοιχεί το ψευδώνυμο αυτό. Για παράδειγμα θα μπορούσαμε να ορίσουμε τα κάτωθι ψευδώνυμα:

```
alias h history
alias l ls
```

Έχοντας δώσει τους ανωτέρω ορισμούς, όταν πληκτρολογούμε

```
h
```

ο φλοιός θα αντικαθιστά το `h` με την εντολή `history` και όταν πληκτρολογούμε

```
l
```

ο φλοιός θα το αντικαθιστά με την εντολή `ls`

Σημειώνουμε ότι η αντικατάσταση λαμβάνει χώρα *μόνο όταν η συμβολοσειρά υπέχει θέση εντολής*, π.χ. αν πληκτρολογήσουμε

```
echo l
```

ο φλοιός *δεν* θα αντικαταστήσει το `l` με `ls`.

Παράμετροι που τυχόν χρησιμοποιούμε σε εντολές που έχουν ορισθεί μέσω του μηχανισμού ψευδωνύμων μεταβιβάζονται κανονικά στις εντολές που τελικά εκτελούνται, π.χ. αν δώσουμε

```
l /etc
```

αυτό είναι ισοδύναμο με το να πληκτρολογούμε

```
ls /etc
```

Τα ψευδώνυμα μπορούν να περιλαμβάνουν και πάνω από μία λέξεις στον ορισμό, π.χ.:

```
alias cx chmod +x
alias ls ls -aF
```

Η πρώτη εντολή μας δίνει τη συντομογραφία `cx` για την εντολή `chmod` με παράμετρο `+x`. Τα αρχεία στα οποία αναφέρεται η εκτέλεση εντολής θα τα παρέχει ο χρήστης κατά την κλήση της `cx` π.χ.

```
cx myscript1 myscript2
```

που είναι ισοδύναμο με

```
chmod +x myscript1 myscript2
```

Στη δεύτερη εντολή (`alias ls ls -aF`) παρατηρούμε ότι είναι δυνατόν να επανορίζουμε μέσω του μηχανισμού ψευδωνύμων την εξ ορισμού συμπεριφορά εντολών του Unix. Όταν δίνουμε την εντολή `ls` ο φλοιός αντικαθιστά την εντολή μας με `ls -aF` *πριν* καλέσει την εντολή, συνεπώς η εντολή `ls` εκτελείται τελικά με τις σχετικές ενδείξεις, δίνοντας κατάλληλα διαμορφωμένο αποτέλεσμα. Τυχόν παράμετροι που δίνουμε στην `ls` μεταβιβάζονται και πάλι κανονικά, π.χ. δίνοντας

```
ls –l /bin
```

είναι ισοδύναμο με το να δώσουμε

```
ls –aF –l /bin
```

(εδώ οι ενδείξεις που έχουμε καθορίσει στον ορισμό του ψευδωνύμου συνενώνονται με αυτές που δίνουμε κατά την κλήση της εντολής). Ένα χρήσιμο σχετικό ψευδώνυμο είναι το

```
alias rm rm -i
```

Η εντολή `alias` χωρίς ορίσματα τυπώνει τη λίστα με τους ορισμούς ψευδωνύμων, ενώ αν θέλουμε να ακυρώσουμε έναν ορισμό ψευδωνύμου δίνουμε την εντολή

```
unalias name
```

π.χ.

```
unalias cx
```

Αν θέλουμε *για μία μόνο εκτέλεση* να καλέσουμε μια ενσωματωμένη εντολή του Unix παρακάμπτοντας τον μηχανισμό ψευδωνύμων έχουμε δύο επιλογές:

1. να τοποθετήσουμε μία αντίστροφη κάθετο *αμέσως πριν* την εντολή χωρίς ενδιάμεσο κενό π.χ.

   ```
   \ls /bin
   \rm –rf mytemp
   ```

2. Να δώσουμε την πλήρη διαδρομή προς την εντολή π.χ.

   ```
   /bin/ls /bin
   /usr/ucb/vi myfile
   ```

Βάσει των όσων αναφέρθησαν προηγουμένως, οι παράμετροι που εισάγουμε κατά την κλήση μιας εντολής που έχει ορισθεί με ψευδώνυμο τοποθετούνται πάντα *στο τέλος του ορισμού*. Είναι ωστόσο πιθανό να μην είναι αυτή η επιθυμητή λειτουργικότητα π.χ. να θέλουμε να επανορίσουμε την εντολή cd ώστε να έχει την εξής συμπεριφορά:

1. Να αλλάζει τον τρέχοντα κατάλογο εργασίας σ' αυτόν που καθορίζεται από τον χρήστη
2. Να τυπώνει στην οθόνη τον νέο κατάλογο εργασίας

άρα η εντολή

```
cd dirname
```

να είναι ισοδύναμη με την ακολουθία εντολών

```
cd dirname; echo $cwd
```

Ο ορισμός

```
alias cd cd \; echo \$cwd
```

δεν έχει το επιθυμητό αποτέλεσμα διότι η κλήση

```
cd /etc
```

θα αντικατασταθεί από την εντολή

```
cd; echo $cwd /bin
```

η οποία θα έχει ως αποτέλεσμα να επιστρέψουμε στον προσωπικό μας κατάλογο και κατόπιν να τυπωθεί ο κατάλογος αυτός και η συμβολοσειρά `/bin`.

Ο μηχανισμός ιστορικού μας παρέχει τους συμβολισμούς !^ και !* με τους οποίους μπορούμε να υποδείξουμε *την πρώτη παράμετρο στην εντολή* και *όλες τις παραμέτρους της εντολής* αντίστοιχα. Έτσι η επιθυμητή λειτουργικότητα της `cd` επιτυγχάνεται με τον ορισμό

```
alias cd cd \!^ \; echo \$cwd
```

ή με τον ορισμό

```
alias cd cd \!\* \; echo \$cwd
```

Ο δεύτερος ορισμός πλεονεκτεί του πρώτου, διότι ο πρώτος *απαιτεί* να δώσουμε μία παράμετρο στην εντολή `cd`, αποκλείοντας την χρήση της χωρίς παραμέτρους για επιστροφή στον προσωπικό μας κατάλογο.

Παρατηρήστε επίσης τη χρήση των αντίστροφων καθέτων στις προηγούμενες εντολές για προστασία των ειδικών χαρακτήρων. Η αντίστροφη κάθετος, όπου χρησιμοποιείται, ορίζει ότι ο ειδικός χαρακτήρας θα διερμηνευτεί *κατά την επέκταση του ψευδωνύμου*, όχι κατά τη στιγμή του ορισμού του (όταν δηλαδή δίνουμε την εντολή alias). Πιο αναλυτικά:

1. αν παραλείψουμε την αντίστροφη κάθετο πριν το `\$cwd`, τότε η μεταβλητή `cwd` θα επεκταθεί στην τιμή της *πριν* την εκτέλεση της εντολής `alias`. Αν ο τρέχων κατάλογος μας έτσι είναι ο `/etc`, η εντολή θα είναι ισοδύναμη με

   ```
   alias cd cd \!\* \; echo /etc
   ```

   η οποία θα μας μεταφέρει στον σχετικό κατάλογο (τον προσδιορισθέντα ή τον προσωπικό), και θα τυπώνει τη συμβολοσειρά `/etc`.

2. Αν παραλείψουμε την αντίστροφη κάθετο πριν το ερωτηματικό, η εντολή μας θα είναι ισοδύναμη με την ακολουθία εντολών

   ```
   alias cd cd \!\*
   echo /etc
   ```

   που δεν θα αλλάξει τίποτε ουσιαστικό στην λειτουργικότητα της `cd`, και θα τυπώσει στην οθόνη μας τη συμβολοσειρά `/etc`.

3. Αν παραλείψουμε την αντίστροφη κάθετο πριν το θαυμαστικό, η ακολουθία `!*` θα διερμηνευτεί από τον μηχανισμό ιστορικού και έτσι στη θέση της θα τοποθετηθούν οι παράμετροι της προηγούμενης εντολής.

4. Αν παραλείψουμε την αντίστροφη κάθετο πριν το αστεράκι, ο φλοιός θα προσπαθήσει να τον ταιριάξει με ονόματα αρχείων, διερμηνεύοντάς τον ως ειδικό χαρακτήρα.

Η προστασία των ειδικών χαρακτήρων μπορεί να γίνεται με οποιοδήποτε δόκιμο τρόπο που καλύπτει τον ειδικό χαρακτήρα που θέλουμε κατά περίπτωση να προστατέψουμε.

2.10 Αντικατάσταση εντολών από το αποτέλεσμά τους

Σε πολλές περιπτώσεις είναι χρήσιμο να χρησιμοποιήσουμε τα αποτελέσματα μιας εντολής (π.χ. τα αποτελέσματα της εντολής `ls` μας δίνουν σε μία μεταβλητή τη λίστα των αρχείων, τα αποτελέσματα της εντολής `who` μας δίνουν τη λίστα των συνδεδεμένων χρηστών κ.λπ.). Στο φλοιό αυτό είναι δυνατόν χρησιμοποιώντας τα αντίστροφα εισαγωγικά (backquotes - `` ` ``). Απλώς παραθέτουμε εντός των αντίστροφων εισαγωγικών την εντολή που επιθυμούμε – ο φλοιός θα εκτελέσει την εντολή και θα τοποθετήσει στη συγκεκριμένη θέση τα αποτελέσματα.

Παραδείγματα

```
1. set x=`date`; echo Time is $x[4]
```

Το αποτέλεσμα της date καταχωρείται στη μεταβλητή x ως λίστα λέξεων. Ακολούθως επιλέγεται και τυπώνεται η ώρα.

2. `echo There are `who | wc -l` users logged in`

Τυπώνει τον αριθμό των χρηστών που είναι συνδεδεμένοι στο σύστημα: το αποτέλεσμα της who διοχετεύεται μέσω σωλήνωσης στην εντολή wc -l. Η έξοδος της τελευταίας εντολής τυπώνεται στη θέση του `who | wc -l`

3. `set c_progs=`find ~ -name *.c -print``

Η έξοδος της εντολής find είναι τα ονόματα αρχείων κάτω από τον προσωπικό μας κατάλογο που έχουν κατάληξη .c. Στη μεταβλητή c_progs καταχωρούνται τα ονόματα αυτά.

2.11 Ενδείξεις τροποποίησης λέξεων

Οι ενδείξεις τροποποίησης λέξεων χρησιμοποιούνται για να εξάγουμε διαδρομές, ονόματα ή καταλήξεις από λέξεις, για μετατροπή απλών μεταβλητών σε λίστες λέξεων και για προστασία κενών που υπάρχουν σε λίστες λέξεων. Αναλυτικότερα, έστω η μεταβλητή filespec με τιμή /usr/users/user1/myfile.ext, όπως θα μπορούσε να ορισθεί με την εντολή

```
set filespec=/usr/users/user1/myfile.ext
```

Τότε είναι δυνατόν να χρησιμοποιήσουμε τους κάτωθι συμβολισμούς:

Ένδειξη	**Παράδειγμα**	**Αποτέλεσμα**
:r	$filespec:r	/usr/users/user1/myfile
:h	$filespec:h	/usr/users/user1
:t	$filespec:t	myfile.ext
:e	$filespec:e	ext

Αν έχουμε μία μεταβλητή τύπου λίστας λέξεων, είναι δυνατόν να εφαρμόσουμε μία ένδειξη τροποποίησης σε κάθε στοιχείο της χρησιμοποιώντας τους συμβολισμούς :gr, :gh, :gt και :ge. Παραδείγματος χάριν:

```
set x=(/usr/ucb/vi /usr/include/stdio.h)
echo $x:gr
αποτέλεσμα: /usr/ucb/vi /usr/include/stdio.h
echo $x:gh
αποτέλεσμα: /usr/ucb /usr/include
echo $x:gt
αποτέλεσμα: vi stdio.h
```

Αν η μεταβλητή x είναι λίστα λέξεων, τότε ο συμβολισμός $x:q είναι ισοδύναμος με τον "$x" δηλαδή η μεταβλητή αντικαθίσταται από τις λέξεις που την αποτελούν, αλλά θεωρείται ολόκληρη ως μία παράμετρος.

Αν έχουμε μία απλή μεταβλητή που περιέχει κενά και ειδικούς χαρακτήρες και θέλουμε να τη μετατρέψουμε σε λίστα λέξεων όπου το κάθε στοιχείο της θα είναι ένα τμήμα της απλής μεταβλητής από αυτά που διαχωρίζονται με κενά, αλλά τους ειδικούς χαρακτήρες αμετάφραστους, τότε θα πρέπει να χρησιμοποιήσουμε τον συμβολισμό :x. Επί παραδείγματι, έστω ο ορισμός της μεταβλητής var ως εξής

```
set var="*.c myfile[1-4]"
```

Θέλουμε να φτιάξουμε μία λίστα λέξεων που το πρώτο στοιχείο της να είναι `*.c` και το δεύτερο `myfile[1-4]`. Η εντολή

```
set x1=($var)
```

δεν θα έχει το αποτέλεσμα που επιθυμούμε, διότι το C-shell θα επεκτείνει τους μεταχαρακτήρες `*` και `[]`. Η εντολή

```
set x1="$var"
```

επίσης δεν θα έχει το επιθυμητό αποτέλεσμα, διότι ολόκληρη η τιμή της μεταβλητής `var` θα εκληφθεί ως ένα στοιχείο. Η εντολή που θα πρέπει να δώσουμε είναι η

```
set x1=$var:x
```

3 Προγράμματα φλοιού

Οι φλοιοί του Unix έχουν τη δυνατότητα να εκτελούν *δέσμες εντολών*. Μία δέσμη εντολών αποθηκεύεται σε ένα αρχείο και από εκεί υποβάλλεται προς εκτέλεση. Τα προγράμματα που συγγράφονται με τον τρόπο αυτό καλούνται *προγράμματα φλοιού* (shell scripts). Τα προγράμματα φλοιού μπορούν να περιέχουν, πέραν των κανονικών εντολών του Unix, και συγκεκριμένες *δομές ελέγχου*, που μας παρέχουν τη δυνατότητα για υπό συνθήκη εκτέλεση, επανάληψη κ.λπ. Η σύνταξη των δομών ελέγχου διαφέρει από φλοιό σε φλοιό. Στη συνέχεια του παρόντος εδαφίου θα χρησιμοποιήσουμε σύνταξη που είναι έγκυρη στο φλοιό *csh*.

3.1 Συγγραφή και εκτέλεση προγραμμάτων φλοιού

Τα προγράμματα φλοιού συγγράφονται με χρήση οποιουδήποτε κειμενογράφου και αποθηκεύονται σε κάποιο αρχείο. Το αρχείο (έστω `myscript`) μπορεί να υποβληθεί για εκτέλεση στον φλοιό `csh` γράφοντας

```
csh myscript
```

ή

```
csh myscript param1 param2 …
```

Στη δεύτερη περίπτωση οι τιμές `param1 param2 …` μεταβιβάζονται ως παράμετροι στο πρόγραμμά μας.

Μία δεύτερη δυνατότητα είναι να δώσουμε *δικαιώματα εκτέλεσης* στο αρχείο που περιέχει πρόγραμμά μας και να το εκτελούμε απλά παραθέτοντας το όνομά του:

```
chmod +x myscript
./myscript
```

ή

```
./myscript param1 param2 …
```

Όταν χρησιμοποιούμε τη δυνατότητα αυτή είναι σημαντικό η *πρώτη* γραμμή του αρχείου να περιέχει τη συμβολοσειρά

```
#
```

ή

```
#! /bin/csh
```

Αν δεν ισχύει τίποτε από αυτά, το πρόγραμμα θα υποβληθεί προς εκτέλεση στον φλοιό `sh`, ο οποίος έχει διαφορετική σύνταξη σε αρκετά σημεία, με συνέπεια να οδηγηθούμε σε σφάλματα και ανεπιτυχή εκτέλεση.

Σημειώσεις

1. Αν γράφουμε το πρόγραμμά μας με επιμελητή κειμένου (text editor) των Windows, υπάρχει πιθανότητα το πρόγραμμά μας να μην εκτελεστεί σωστά, καθώς τα προγράμματα των Windows τερματίζουν τις γραμμές χρησιμοποιώντας την ακολουθία ειδικών χαρακτήρων <*επιστροφή δρομέα, αλλαγή γραμμής*>, η οποία δεν είναι καταληπτή από όλες τις εκδόσεις του φλοιού `csh` (ο οποίος αναζητά τον χαρακτήρα <*αλλαγή γραμμής*> μόνο).

2. Η χρήση του προσδιορισμού καταλόγου `./` στην κλήση της εντολής (`./myscript`) είναι απαραίτητη όταν ο τρέχων κατάλογος (`.`) δεν περιλαμβάνεται στη μεταβλητή `path`.
3. Μερικοί χρήστες φτιάχνουν έναν κατάλογο `bin` κάτω από τον προσωπικό τους κατάλογο και τοποθετούν τα δικά τους εκτελέσιμα προγράμματα φλοιού στον κατάλογο αυτό. Φυσικά ρυθμίζουν τα δικαιώματα των προγραμμάτων φλοιού κατά κατάλληλα (προσθέτουν το δικαίωμα εκτέλεσης) ενώ επίσης φροντίζουν να περιλαμβάνεται ο συγκεκριμένος κατάλογος στη μεταβλητή `path` του περιβάλλοντός τους, τοποθετώντας την ακόλουθη εντολή στο τέλος του αρχείου `.cshrc` στον προσωπικό τους κατάλογο:

   ```
   set path=($path ~/bin)
   ```

3.2 Δομές εκτέλεσης υπό συνθήκη

3.2.1 Η δομή ελέγχου if

Η σύνταξη της δομής ελέγχου `if` είναι

```
if (expression1) then
     command(s)1
else if (expression2)
     commands(s)2
[else if (expression3)
     commands(s)3 …]]
[else
     commands]
endif
```

Οι συνθήκες στις εκφράσεις `expression1`, `expression2`, ... εξετάζονται διαδοχικά, μέχρι να βρεθεί κάποια που να αληθεύει. Αν βρεθεί κάποια συνθήκη που να αληθεύει εκτελούνται οι σχετικές εντολές, ενώ αν καμία δεν αληθεύει και είναι παρόν το τμήμα `else`, τότε εκτελούνται οι εντολές στο τμήμα αυτό. Σημειώνεται ότι *το πολύ ένα σύνολο εντολών θα εκτελεστεί* (ενδεχομένως και κανένα, αν δεν αληθεύει καμία συνθήκη και δεν είναι παρόν το τμήμα `else`. Ανεξαρτήτως του πλήθους των τμημάτων `else if` που παραθέτουμε, στο τέλος θα πρέπει να χρησιμοποιήσουμε μόνο μία `endif`. Επίσης, είναι σημαντικό να χρησιμοποιούμε αλλαγές γραμμών όπως φαίνεται στην ανωτέρω σύνταξη, διαφορετικά οι εντολές δεν θα ερμηνευθούν σωστά.

Στις συνθήκες μπορούν να χρησιμοποιηθούν οι κλασικοί τελεστές σύγκρισης (==, <, >, <=, >=, !=) καθώς επίσης και οι ακόλουθες συντακτικές δομές:

Συντακτική δομή	*Παράδειγμα*	*Ερμηνεία*
`string =~ pattern`	`$x =~ [Ll]emon*`	Αληθεύει όταν η αριστερή συμβολοσειρά ταιριάζει με το πρότυπο που παρατίθεται δεξιά. Το ταίριασμα γίνεται με τον ίδιο τρόπο που χρησιμοποιούνται οι μεταχαρακτήρες για ταίριασμα αρχείων. Στο παράδειγμα, η σύγκριση αληθεύει αν η μεταβλητή `x` έχει τις τιμές `Lemon`, `lemon`, `lemonade` κ.λπ.
`string !~ pattern`	`$x !~ [Ll]emon*`	Αληθεύει όταν δεν αληθεύει η `string =~ pattern`
`-d filename`	`-d mydir`	Αληθεύει όταν το `filename` υπάρχει και είναι κατάλογος
`-f filename`	`-f myfile`	Αληθεύει όταν το `filename` υπάρχει και είναι αρχείο
`-e filename`	`-e myfile`	Αληθεύει όταν το `filename` υπάρχει (μπορεί να είναι αρχείο, κατάλογος ή οτιδήποτε άλλο αντικείμενο συστήματος αρχείων)
`-o filename`	`-o myfile`	Αληθεύει όταν ο χρήστης που εκτελεί το πρόγραμμα είναι ιδιοκτήτης του αρχείου
`-r filename`	`-r myfile`	Αληθεύει όταν ο χρήστης που εκτελεί το πρόγραμμα έχει δικαίωμα ανάγνωσης στο αρχείο
`-w filename`	`-w myfile`	Αληθεύει όταν ο χρήστης που εκτελεί το πρόγραμμα έχει δικαίωμα εγγραφής στο αρχείο
`-x filename`	`-x myfile`	Αληθεύει όταν ο χρήστης που εκτελεί το πρόγραμμα έχει δικαίωμα εκτέλεσης στο αρχείο
`-z filename`	`-z myfile`	Αληθεύει όταν το αρχείο υπάρχει και έχει μηδενικό μέγεθος
`{ command }`	`{ grep -q ab f1 }`	Αληθεύει όταν η εντολή εντός των αγκίστρων έχει κατάσταση εξόδου ίση με μηδέν. Σε αντίθετη περίπτωση είναι ψευδής. Το παράδειγμα αληθεύει όταν το αρχείο `f1` υπάρχει και περιέχει τη συμβολοσειρά `ab`.

3.2.2 Η δομή ελέγχου switch

Η σύνταξη της εντολής `switch` είναι:

```
switch (string)
      case pattern1:
            commands1
            breaksw
      [ case pattern2:
            commands2
            breaksw
      … ]
      [ default:
            commandsn
            breaksw ]
endsw
```

Η συμβολοσειρά `string` συγκρίνεται διαδοχικά με τα πρότυπα *pattern1*, *pattern2* κ.ο.κ. Αν η συμβολοσειρά ταιριάζει με το πρότυπο, εκτελούνται οι αντίστοιχες εντολές (`commandsi`). Αν η συμβολοσειρά δεν ταιριάζει με κανένα πρότυπο και είναι παρούσα η πρόταση `default`, τότε εκτελούνται οι εντολές `commandsn`, ενώ αν η συμβολοσειρά δεν ταιριάζει με κανένα πρότυπο και *δεν* είναι παρούσα η πρόταση `default`, τότε δεν εκτελείται καμία εντολή.

Τα πρότυπα δεν είναι απαραίτητο να περικλείονται σε εισαγωγικά. Επίσης, είναι σημαντικό να χρησιμοποιούμε αλλαγές γραμμών όπως φαίνεται στην ανωτέρω σύνταξη, διαφορετικά οι εντολές δεν θα ερμηνευθούν σωστά.

3.3 Δομές επανάληψης

3.3.1 Η εντολή foreach

Η σύνταξη της εντολής `foreach` είναι:

```
foreach var (word1 word2 …)
      commands
end
```

Το αποτέλεσμα της εντολής είναι να τίθεται η μεταβλητή var διαδοχικά στις τιμές `word1 word2…` και να εκτελούνται οι σχετικές εντολές.

3.3.2 Η εντολή while

Η σύνταξη της εντολής `while` είναι:

```
while (condition)
      commands
end
```

Αρχικά αποτιμάται η συνθήκη `condition`. Αν είναι αληθής, εκτελούνται οι εντολές `commands` και η διαδικασία επαναλαμβάνεται. Αν η συνθήκη είναι ψευδής, η εντολή `while` τερματίζεται.

3.3.3 Οι εντολές break και continue

Μεταξύ των εντολών `commands` στις εντολές επανάληψης (`foreach, while`) μπορούν να χρησιμοποιηθούν οι εντολές `break` και `continue`. Η εντολή `break`,

όταν εκτελείται προκαλεί τον συνολικό τερματισμό της εντολής επανάληψης. Η εντολή `continue`, όταν εκτελείται προκαλεί την άμεση έναρξη του επόμενου βρόχου εκτέλεσης για την περιέχουσα εντολή επανάληψης.

3.4 Άλλες εντολές σε προγράμματα φλοιού

3.4.1 Η εντολή exit

Η σύνταξη της εντολής `exit` είναι

```
exit [status]
```

όπου το `status`, αν παρατίθεται, πρέπει να είναι ακέραια αριθμητική τιμή. Η εκτέλεση της εντολής `exit` προκαλεί τον άμεσο τερματισμό του προγράμματος φλοιού, θέτοντας τον κωδικό κατάστασης εξόδου στην τιμή του `status`. Αν το `status` *δεν* παρατίθεται, η εντολή είναι ισοδύναμη με την

```
exit $?
```

που θέτει τον κωδικό κατάστασης εξόδου στην τιμή του κωδικού κατάστασης εξόδου που επέστρεψε η τελευταία εντολή στο πρόγραμμα φλοιού πριν την `exit`.

3.4.2 Η εντολή goto

Η σύνταξη της εντολής `goto` είναι

```
goto label
```

όπου `label` είναι μία *ετικέτα* που πρέπει να ορίζεται μέσα στο πρόγραμμα φλοιού με τη σύνταξη

```
label:
```

Ο ορισμός της ετικέτας πρέπει να βρίσκεται σε μία γραμμή μόνος του. Η εκτέλεση της εντολής `goto` μεταφέρει άμεσα τη ροή εκτέλεσης στην εντολή που ακολουθεί τον ορισμό της ετικέτας.

3.4.3 Η εντολή sleep

Η σύνταξη της εντολής `sleep` είναι

```
sleep numseconds
```

όπου `numseconds` πρέπει να είναι ακέραια αριθμητική τιμή. Η εκτέλεση της εντολής προκαλεί την παύση εκτέλεσης του προγράμματος για τον προσδιορισθέντα αριθμό δευτερολέπτων.

3.4.4 Σχόλια

Οι γραμμές που ξεκινάνε με τον χαρακτήρα # θεωρούνται ως σχόλια και αγνοούνται κατά την εκτέλεση. Αν ο χαρακτήρας # βρίσκεται στη μέση μίας γραμμής, τότε το τμήμα *αριστερά* του χαρακτήρα εκτελείται κανονικά και το τμήμα *δεξιά* του χαρακτήρα θεωρείται σχόλιο. Ο χαρακτήρας # δεν θεωρείται ως σήμανση σχολίου αν περικλείεται σε εισαγωγικά ή αν τοποθετήσουμε *ακριβώς πριν από αυτόν* την ανάστροφη κάθετο.

3.5 Παράμετροι σε προγράμματα φλοιού

Κάθε πρόγραμμα φλοιού έχει πρόσβαση στη μεταβλητή `argv`, στην οποία ο φλοιός τοποθετεί τις παραμέτρους που ορίζουμε στη γραμμή εντολών ότι επιθυμούμε να μεταβιβαστούν στο πρόγραμμα. Η μεταβλητή είναι *λίστα λέξεων*, συνεπώς μπορούμε να χρησιμοποιήσουμε όλους τους συμβολισμούς που είναι διαθέσιμες για τον τύπο αυτό μεταβλητών. Η τιμή του πρώτου στοιχείου της λίστας λέξεων είναι η πρώτη παράμετρος, η τιμή του δεύτερου στοιχείου της λίστας λέξεων είναι η δεύτερη παράμετρος κ.ο.κ.

Αν για παράδειγμα καλέσουμε ένα πρόγραμμα φλοιού με την εντολή

```
myscript param1 param2 … paramn
```

τότε θα ισχύουν:

```
$#argv = n
$argv[1] = param1
$argv[2] = param2
…
$argv[n] = paramn
```

Μπορεί να χρησιμοποιηθεί επίσης και ο συμβολισμός `$0` που υποδηλώνει το όνομα του προγράμματος που εκτελείται.

Παράδειγμα

Το πιο κάτω πρόγραμμα τυπώνει τις παραμέτρους του, μία σε κάθε γραμμή.

```
#! /bin/csh
@ count = 0
while ($count <= $#argv)
      echo $argv[$count]
      @ count++
end
exit 0
```

3.6 Διοχέτευση εισόδου σε εντολές εντός του προγράμματος φλοιού

Υπάρχουν περιστάσεις κατά τις οποίες επιθυμούμε να εκτελέσουμε μέσα από ένα πρόγραμμα φλοιού μία εντολή, παρέχοντάς της συγκεκριμένη είσοδο. Μολονότι αυτό είναι δυνατόν να το επιτύχουμε αποθηκεύοντας την είσοδο σε ένα αρχείο π.χ. `inputfile` και να καλέσουμε την εντολή με

```
cmd < inputfile
```

(όπου `cmd` είναι η εντολή), η μέθοδος αυτή έχει τα μειονεκτήματα ότι (α) διατηρούμε δύο αρχεία και (β) αν καλέσουμε το πρόγραμμα από κάποιον άλλο κατάλογο, το αρχείο `inputfile` μπορεί να μην είναι διαθέσιμο. Μια λύση στο πρόβλημα αυτό είναι η χρήση του συμβολισμού << ο οποίος επιτρέπει να παραθέσουμε την είσοδο που επιθυμούμε να παράσχουμε στην εντολή εντός του προγράμματος φλοιού. Η σύνταξη είναι:

```
cmd << word
γραμμή εισόδου 1
...
γραμμή εισόδου ν
word
```

Ο συμβολισμός << ακολουθείται από μία λέξη (word) που οριοθετεί το τέλος των γραμμών εισόδου. Η λέξη, για να οριοθετεί το τέλος των γραμμών εισόδου, πρέπει να βρίσκεται σε μία γραμμή μόνη της. Αν οι γραμμές εισόδου περιέχουν μεταβλητές (συμβολισμούς $...) ή εντολές σε αντίστροφα εισαγωγικά (`) τότε εκτελούνται οι σχετικές αντικαταστάσεις και το αποτέλεσμα των αντικαταστάσεων μεταβιβάζεται ως είσοδος στην εντολή.

Παράδειγμα

```
/usr/ucb/mail -s Hello $user << END
This is a message for user $user
End of message
END
```

Αν η τιμή της μεταβλητής user είναι root, τότε θα σταλεί στον χρήστη root ένα ηλεκτρονικό μήνυμα με το περιεχόμενο

```
This is a message for user root
End of message
```

3.7 Περιβάλλον εκτέλεσης προγραμμάτων φλοιού

Όταν καλούμε ένα πρόγραμμα φλοιού, τότε αυτόματα δημιουργείται ένα νέο *στιγμιότυπο φλοιού* ή *υποφλοιός* (subshell) στο περιβάλλον του οποίου θα εκτελεστεί το πρόγραμμα. Η εκτέλεση του προγράμματος σε ξεχωριστό στιγμιότυπο φλοιού έχει τις εξής επιπτώσεις:

- Μεταβλητές που έχουμε ορίσει στο τρέχον στιγμιότυπο φλοιού μέσω των εντολών set και @ δεν είναι διαθέσιμες στο νέο στιγμιότυπο φλοιού
- Ψευδώνυμα που έχουμε ορίσει στο τρέχον στιγμιότυπο φλοιού δεν είναι διαθέσιμες στο νέο στιγμιότυπο φλοιού.
- Αν αλλάξουμε εντός του προγράμματος την τιμή μιας μεταβλητής και υπάρχει μεταβλητή με ίδιο όνομα στον τρέχοντα φλοιό, η αλλαγή δεν θα είναι ορατή μετά τον τερματισμό του προγράμματος
- Αν ορίσουμε κάποιο ψευδώνυμο εντός του προγράμματος, το ψευδώνυμο δεν θα είναι διαθέσιμο μετά τον τερματισμό του προγράμματος
- Αν αλλάξουμε τρέχοντα κατάλογο εργασίας εντός του προγράμματος, μετά τον τερματισμό του θα βρεθούμε ξανά στον τρέχοντα κατάλογο εργασίας που βρισκόμαστε και πριν την εκτέλεση του προγράμματος.

Αν επιθυμούμε μία μεταβλητή να είναι ορατή σε προγράμματα που καλούμε από τον φλοιό, θα πρέπει να την ορίσουμε με την εντολή setenv αντί της set. Δεν υπάρχει τρόπος να ορίσουμε μεταβλητή σε ένα πρόγραμμα φλοιού και αυτή να είναι ορατή στον φλοιό που κάλεσε το πρόγραμμα.

Μπορούμε εντός ενός προγράμματος να ορίσουμε ότι κάποιες εντολές θα εκτελεστούν σε ένα ξεχωριστό *στιγμιότυπο φλοιού*, περικλείοντάς τες σε παρενθέσεις. Η σύνταξη αυτή χρησιμοποιείται κυρίως για τον καθορισμό εισόδου-εξόδου, δεδομένου ότι οι εντολές που ομαδοποιούνται σε παρενθέσεις έχουν κοινή είσοδο και έξοδο. Έτσι η σύνταξη

```
date; ls > myfile
```

θα τυπώσει την ημερομηνία στο τερματικό και θα γράψει το αποτέλεσμα της `ls` στο αρχείο `myfile`, ενώ η εντολή

```
(date; ls) > myfile
```

θα γράψει στο αρχείο `myfile` την ημερομηνία ακολουθούμενη από το αποτέλεσμα της `ls`. Επίσης, η εντολή

```
(cmd1; cmd2) < myfile
```

υποδεικνύει ότι η εντολή `cmd1` θα διαβάζει την είσοδό της από το αρχείο `myfile` και η είσοδος της `cmd2` θα ξεκινήσει από το σημείο του `myfile` όπου σταμάτησε να διαβάζει η `cmd1`. Αντιθέτως η σύνταξη

```
cmd1 < myfile; cmd2 < myfile
```

θα έχει ως αποτέλεσμα και οι δύο εντολές να διαβάζουν είσοδο από την αρχή του αρχείου `myfile`. Η χρήση παρενθέσεων είναι επιβεβλημένη για την ανακατεύθυνση της εισόδου όταν χρησιμοποιούμε τον συμβολισμό `$<` για την ανάγνωση της τιμής μιας μεταβλητής από την κανονική είσοδο. Πιο συγκεκριμένα η σύνταξη

```
set x=$< < myfile
```

δεν προκαλεί την ανάγνωση της τιμής της μεταβλητής από το αρχείο `myfile`: η ανάγνωση γίνεται κανονικά από την κανονική είσοδο του προγράμματος. Η ανακατεύθυνση μπορεί να επιτευχθεί χρησιμοποιώντας παρενθέσεις, με την ακόλουθη σύνταξη:

```
(set x=$< ) < myfile
```

η οποία όμως, μια και εκτελείται σε ξεχωριστό στιγμιότυπο φλοιού δεν έχει επίδραση στο τρέχον στιγμιότυπο φλοιού και έτσι η μεταβλητή `x` δεν ορίζεται. Η σύνταξη που μπορούμε να χρησιμοποιήσουμε να ορίσουμε μία μεταβλητή χρησιμοποιώντας ανακατεύθυνση εισόδου είναι:

```
set x=`(echo "$<") < myfile`
```

Παρατηρήστε τη χρήση των διπλών εισαγωγικών που προστατεύουν κενά και μεταχαρακτήρες από την ερμηνεία τους, στην οποία θα τους υπέβαλε ο φλοιός.

Τέλος, με χρήση παρενθέσεων είναι δυνατόν να διαχωρίσουμε την έξοδο μιας εντολής από την έξοδο σφαλμάτων της:

```
(command > myoutput) >& myerrors
```

Αν επιθυμούμε να εκτελεστεί ένα αρχείο φλοιού στον τρέχοντα φλοιό και όχι σε καινούργιο στιγμιότυπο φλοιού, μπορούμε να χρησιμοποιήσουμε την εντολή `source`. Δίνοντας

```
source myfile
```

το πρόγραμμα `myfile` εκτελείται στον τρέχοντα φλοιό και όχι σε καινούργιο στιγμιότυπο φλοιού.

3.8 Χειρισμός διακοπών χρήστη σε προγράμματα φλοιού

Αν κατά τη διάρκεια εκτέλεσης ενός προγράμματος φλοιού ο χρήστης πατήσει CTRL-C, η εκτέλεση του προγράμματος διακόπτεται. Η συμπεριφορά αυτή μπορεί να αλλάξει χρησιμοποιώντας την εντολή `onintr` ως ακολούθως:

Εντολή	*Αποτέλεσμα*
`onintr -`	Το πάτημα του CTRL-C αγνοείται
`onintr label`	Όταν πατηθεί CTRL-C εκτελείται η εντολή `goto label`. Η ετικέτα `label` πρέπει να έχει ορισθεί όπως και στην εντολή `goto`.
`onintr`	Επαναφέρεται η εξ ορισμού συμπεριφορά για το πάτημα του CTRL-C

3.9 Ανίχνευση σφαλμάτων σε προγράμματα φλοιού

Όταν υπάρχει συντακτικό σφάλμα σε πρόγραμμα φλοιού, ο φλοιός θα τυπώσει ένα μήνυμα που (συνήθως) δεν είναι ιδιαίτερα κατατοπιστικό για το ποιο είναι το συγκεκριμένο σφάλμα. Επίσης, δεν υπάρχει κάποιο βοηθητικό πρόγραμμα για τον εντοπισμό λογικών σφαλμάτων. Για να εντοπίσουμε συντακτικά ή λογικά σφάλματα σε προγράμματα φλοιού μπορούμε να χρησιμοποιήσουμε κάποιες από τις ενδείξεις εκτέλεσης του φλοιού ως ακολούθως:

```
csh –v script [param1 param2 …]
```

το πρόγραμμα `script` με παραμέτρους `param1 param2` ... εκτελείται κανονικά, με τις εκτελούμενες εντολές να τυπώνονται *πριν* αντικατασταθούν οι μεταβλητές με την πραγματική τιμή τους.

```
csh –x script [param1 param2 …]
```

το πρόγραμμα `script` με παραμέτρους `param1 param2` ... εκτελείται κανονικά, με τις εκτελούμενες εντολές να τυπώνονται *αφού* αντικατασταθούν οι μεταβλητές με την πραγματική τιμή τους.

```
csh –n script [param1 param2 …]
```

Ο φλοιός απλώς επιβεβαιώνει τη συντακτική ορθότητα των εντολών και δεν τις εκτελεί. Συνήθως η ένδειξη `–n` χρησιμοποιείται μαζί με την ένδειξη `–v`

```
csh –vn script [param1 param2 …]
```

Παράρτημα Α- Παραδείγματα προγραμμάτων φλοιού

Παράδειγμα 1: pick

Τυπώνονται οι παράμετροι μία προς μία και ζητάται από τον χρήστη να εισάγει “Yes” “No” ή “Quit”. Αν ο χρήστης εισάγει “Yes” το η παράμετρος τυπώνεται, αν εισάγει “Quit” η επεξεργασία τερματίζεται.

```
#! /bin/csh
if ($#argv == 0) then
      echo Usage is: $0 param1 param2 ...
      exit 1
endif

foreach arg ($argv:q)
      echo -n "$arg":\  > /dev/tty
      set response = $<
      if ( "$response" =~ [Yy]* ) then
            echo "$arg"
      else if ( "$response" =~ [Qq]* ) then
            break
      endif
end
exit 0
```

Παράδειγμα 2: pick2

Τυπώνονται οι παράμετροι μία προς μία και ζητάται από τον χρήστη να εισάγει “Yes” “No” ή “Quit”. Αν ο χρήστης εισάγει “Yes” το η παράμετρος τυπώνεται, αν εισάγει “Quit” η επεξεργασία τερματίζεται. Αν ο χρήστης δεν έχει παράσχει παραμέτρους, προτρέπεται για τις τιμές που διαβάζονται από την τυπική είσοδο και οι απαντήσεις διαβάζονται από το τερματικό.

```
#! /bin/csh
if ($#argv != 0) then
      foreach arg ($argv:q)
            echo -n "$arg":\  > /dev/tty
            set response = $<
            if ( "$response" =~ [Yy]* ) then
                  echo "$arg"
            else if ( "$response" =~ [Qq]* ) then
                  break
            endif
      end
else
      set item = "$<"
      while ("$item" != "")
            echo -n "$item":\ > /dev/tty
            set response = `(echo "$<") < /dev/tty`
            if ( "$response" =~ [Yy]* ) then
                  echo "$item"
            else if ( "$response" =~ [Qq]* ) then
                  break
            endif
            set item = "$<"
      end
endif
exit 0
```

Παράδειγμα 3: uterm

Βρίσκει σε ποιο τερματικό είναι συνδεδεμένος ένας συγκεκριμένος χρήστης και την ώρα σύνδεσής του

```
#! /bin/csh

if ($#argv != 1) then
      echo Usage is $0 username
      exit 1
endif

set users="`who | grep '^$1[  ]'`"
if ($#users == 0) then
      echo User $argv[1] is not logged in
else
      foreach user ($users:q)
            set user_data=($user:x)
            echo -n User $user_data[1] is logged in at teminal
$user_data[2]
            echo " Login time is $user_data[5]"
      end
endif
```

Παράδειγμα 4: wherecmd

Τυπώνει σε ποιο κατάλογο βρίσκονται οι εντολές που δίνουμε ως παραμέτρους.

```
#! /bin/csh

if ($#argv == 0) then
      echo Usage is $0 command\(s\)
      exit 1
endif

if (! $?path) then
      echo path variable not set
      exit 2
endif

foreach cmd ($argv)
      set found=0
      foreach dir ($path)
            if (-x $dir/$cmd) then
                  echo ${cmd}: $dir/$cmd
                  set found=1
                  break
            endif
      end
      if (! $found) echo ${cmd}: not found
end
```

Παράδειγμα 5: ftype

Τυπώνει το είδος των αρχείων (απλά αρχεία, κατάλογοι κ.λπ.) που δίνονται ως παράμετροι.

```
#! /bin/csh

if ($#argv == 0) then
      echo Usage is $0 file\(s\)
      exit 1
```

```
endif

foreach arg ($argv:q)
	# Use "ls" to handle metacharacters (stars, question marks etc)
	# double quotes preserve newlines, creating one element in the array
	# per line
	set files = "`ls -ld $arg`"
	foreach file ($files:q)
		# expand the line into a word list
		set file_data = ($file:x)
		set file_name = $file_data[9]
		if ("$file_data" =~ l*) then	# flag links
			set is_link = 1
		else
			set is_link = 0
		endif
		while ( "$file_data" =~ l*)
			set target_file = $file_data[11]
			if ( $target_file =~ /*) then # link to absolute path
				set file_data = `ls -ld $target_file`
				set arg=`dirname $target_file`
			else if ("$arg" =~ */*) then # arg contains a directory
				set file_data = `ls -ld $arg:h/$target_file`
			else
				set file_data = `ls -ld $target_file`
			endif
			if ("$file_data" == "") then
				set file_data = "N"
				break
			endif
		end
		echo -n $file_name\: \
		if ( $is_link ) echo -n link to \
		switch ( "$file_data" )
			case d* :
				echo directory
				breaksw
			case -* :
				echo plain file
				breaksw
			case N :
				echo Something that could not be found
				breaksw
			default:
				something else
				breaksw
		endsw
	end
end
exit 0
```

Παράδειγμα 6: lsr

Αναδρομική λίστα των καταλόγων που προσδιορίζονται. Αν δεν δίνονται κατάλογοι, τυπώνεται αναδρομική λίστα του τρέχοντος καταλόγου.

```
#! /bin/csh

if ($#argv == 0) set argv=( . )

foreach file ($argv)
	set dirlist = ()
	if (! -d $file && -e $file) then
		echo $file\: not a directory
	else if (! -e $file) then
		echo $file\: no such file or directory
	else
```

```
            set x="`ls -A1 $file`"
            @ no_files = $#x + 2
            echo "$file  (total $no_files)"
            foreach f1 ($x:q)
                  if (-d $file/$f1) set dirlist = ($dirlist $file/$f1)
                  echo $file/$f1
            end
            foreach fl ($dirlist)
                  echo ""
                  $0 $fl
            end
      end
exit 0
```

Παράδειγμα 7: overwrite

Η πρώτη παράμετρος θεωρείται ως όνομα αρχείου και οι επόμενες ως εντολή. Εφαρμόζεται η εντολή με είσοδο το αρχείο και το αρχείο αντικαθίσταται με το αποτέλεσμα την έξοδο της εντολής.

```
#! /bin/csh

if ($#argv < 2) then
      echo Usage is $0 filename command
      exit 1
endif

set file=$argv[1]
if (! -f $file:q) then
      echo $0\: $file\: not a file or nonexistent
      exit 1
else if (! -r $file:q || ! -w $file:q) then
      echo $0\: $file\: must have read and write permission
      exit 1
endif

shift # now argv contains only the command
onintr cleanup

# temporary file names
set old=/tmp/old.$$
set new=/tmp/new.$$
cp $file $old

$argv:x < $old > $new
if ($status != 0) then
      echo $0\: error running command - $file unchanged
else
      onintr -
      cp $new $file
endif

cleanup:
onintr -
rm -f $old $new >& /dev/null
```

Παράρτημα Β - Ο κειμενογράφος πλήρους οθόνης vi

Εντολή ví: Η εντολή για την προετοιμασία κειμένου μέσα στο αρχείο "όνομα αρχείου" είναι:

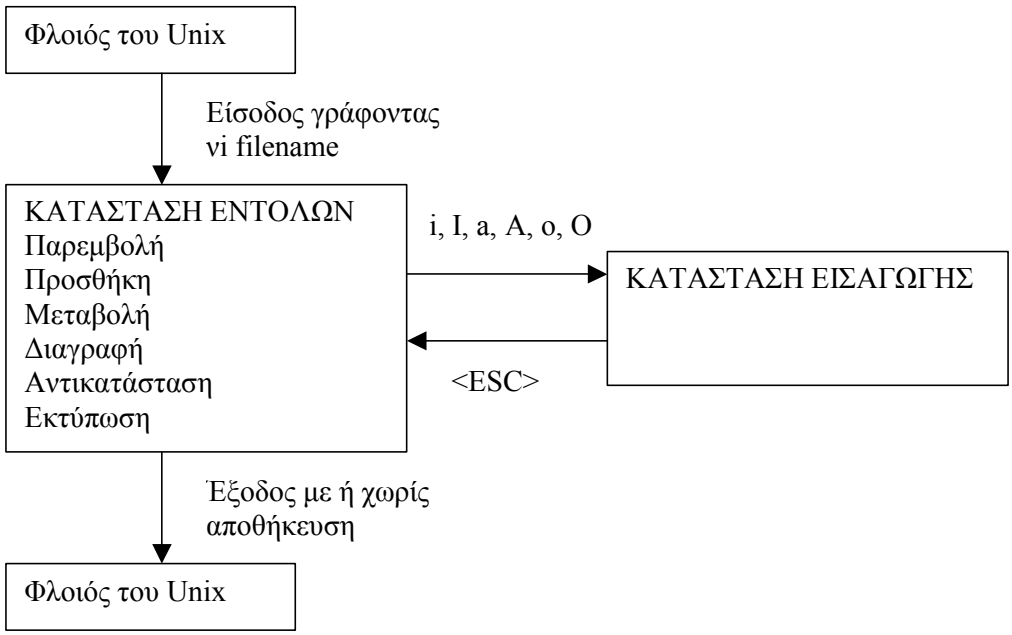
ví όνομα αρχείου

Αν δεν υπάρχει τέτοιο αρχείο, θα δημιουργηθεί όταν δοθεί αυτή η εντολή.

3.10 Περιβάλλον λειτουργίας του ví

Α) Κατάσταση εντολών

Για να εισέλθετε: Σας τοποθετεί στην κατάσταση εντολών όταν καλείτε τον vi και σας επιτρέπει να χρησιμοποιήσετε τις εντολές που περιγράφονται στη συνέχεια.



Β) Κατάσταση κειμένου

Για να εισέλθετε: Οποιαδήποτε από τις παρακάτω εντολές θα σας τοποθετήσει στην κατάσταση εισαγωγής κειμένου: a, i, o, O. και σας επιτρέπει να χρησιμοποιήσετε το πληκτρολόγιο για να εισάγετε κείμενο.

Για να εισέλθετε εκ νέου σε κατάσταση εντολών από την κατάσταση κειμένου, πατήστε το πλήκτρο ESC.

Οι εντολές του vi εφαρμόζονται στη θέση του cursor .Οι εντολές αυτές σας βοηθούν να τοποθετήσετε τον cursor στη θέση που θέλετε να βρίσκεται μέσα στο κείμενο. Ο cursor δεν θα μετακινηθεί πέρα από τα όρια του υπάρχοντος κειμένου. Τα βέλη

↑,↓,←,→, σας δίνουν τη δυνατότητα να μετακινηθείτε μέσα στο κείμενο. Ισοδύναμα ο cursor μπορεί να μετακινηθεί με τη χρήση των παρακάτω πλήκτρων (όταν βρισκόμαστε σε κατάσταση εντολών).

j: Μετακινεί τον cursor προς τα κάτω μία γραμμή.

k: Μετακινεί τον cursor προς τα πάνω μία γραμμή.

h: Μετακινεί τον cursor αριστερό ένα χαρακτήρα.

l: Μετακινεί τον cursor δεξιά ένα χαρακτήρα.

Αυτών των εντολών μπορεί να προηγείται ένας θετικός ακέραιος για να προσδιορίζει τον αριθμό των γραμμών ή χαρακτήρων μετακίνησης του cursor .

nG Μετακινεί τον cursor στην n-οστή γραμμή του αρχείου.

^ Μετακινεί τον cursor στην αρχή της τρέχουσας γραμμής.

$ Μετακινεί τον cursor στο τέλος της γραμμής.

3.11 Εντολές εισαγωγής κειμένου

a: Προσθέτει κείμενο μετά τη θέση του cursor

i: Παρεμβάλλει κείμενο πριν τη θέση του cursor.

o: Εισάγει μία νέα γραμμή κάτω από τη θέση του cursor.

O: Εισάγει μία νέα γραμμή πάνω από τη θέση του cursor.

3.12 Εντολές Διαγραφής κειμένου

x: Διαγράφει τον χαρακτήρα που βρίσκεται κάτω από τον cursor.

dd: Διαγράφει τη γραμμή που περιέχει τον cursor.

Επίσης ισχύει nx, ndd για διαγραφή πολλών χαρακτήρων ή γραμμών.

r: Αντικαθιστά τον χαρακτήρα που βρίσκεται κάτω από τον cursor με τον επόμενο χαρακτήρα που πληκτρολογείται.

s: Αντικαθιστά τον χαρακτήρα που βρίσκεται κάτω από τον cursor με το κείμενο που θα πληκτρολογήσετε μέχρι να πατήσετε <ESC>.

3.13 Αποθήκευση κειμένου και έξοδος από τον κειμενογράφο

Η προετοιμασία του κειμένου γίνεται σε μια προσωρινή περιοχή εργασίας και μπορεί να αποθηκευτεί σε ένα μόνιμο αρχείο.

<esc>: w Γράφει το τρέχον κείμενο μέσα στο μόνιμο αρχείο.

<esc> : q Εξέρχεται, αν δεν υπάρχουν μεταβολές από την τελευταία εντολή w.

<esc>: q! Εξέρχεται χωρίς να γράφονται οι μεταβολές στο κείμενο.

<esc> : wq Γράφει το αρχείο και εξέρχεται.

<esc>:zz Γρόφει και εξέρχεται

<esc>: fname Αλλάζει το όνομα του αρχείου σε «name».

<esc> : set nu Δείχνει τους αριθμούς γραμμών του κειμένου που γράφουμε