Comenzi pentru prelucrarea fisierelor text

Numeroase intrucit Sistemul UNIX a fost considerat un sistem de dezvolatre de programe unde prelucrarea informatiilor (textelor) e importanta.

1. Afisarea primelor linii dintr-un fisier

Comanda: **head(antet)**

Sintaxa:

head [-n] [filename…]

Obs In absenta unor nume de fisiere fiserul standard de intrare este copiat in fiserul standard de iesire.

Optiunea **–n**, ($\overline{1,9999}$), reprezinta numarul de linii afisate. Valoarea implicita este 10.

Cand se specifica mai multe fisiere liniile fiecaruia sunt **precedate** de

===> file name < ===

2. Afisarea ultimei portiuni dintr-un fisier

Comanda: **tail (sfirsit, coada)**

Sintaxa:

tail +| -number [lbc] [f] [filename]

tail +| -number [l] [rf] [filename]

Obs - C-da se refera intotdeauna la un singur fisier (fisierul standard de intrare daca nu este specificat explicit).

- In absenta **op number** se folosete valoarea implicita 10, iar unitatea de afisare poate fi:

- linia (**optiunea l** sau implicit)

- blocul (**optiunea b**)

- caracterul (**optiunea c)**

Important!!!

- **+ number:** pozitia de inceput a afisarii este considerata relativa la inceputul fisierului ;

- **- number:** pozitia se considera relativ la sfirsitul fisierului;

- pt a II-a forma sintactica, unde se lucreaza numai cu linii, **optiunea r** are efectul afisarii in ordine inversa a liniilor;

- **optiunea f,** in cazul in care fisierul nu este pipe face ca operatia sa nu se termine dupa afisarea ultimei unitati solicitate, ci sa intre intr-un ciclu infinit:⟶ se poate urmari **cresterea dimensiunilor unui fisier scris de un alt proces.**

3. Determinarea numarului de linii , cuvinte si caractere dintr-un fisier

Comanda: **wc ("word count")**

Sintaxa:

wc [-lwc] [file name…]

Obs: - in absenta numelui de fisier se considera implicit fisierul **standard de intrare.**

- cuvint = un sir de caractere delimitat de **spatiu, tab** sau **"newline"**;

- daca in c-da apar mai **multe fisiere** se afiseaza si **numarul total** ca in expl:

```
$ ls –l *.txt
total 2
-rw-r--r-- 1 ada 42 nov 9 13:57 fis1.txt
-rw-r--r-- 1 ada 92 nov 9 13:38 fis2.txt
$ wc *txt
2   6   42  fis1.txt
4   16  92  fis2.txt
6   22  134 total
```

4. Sortarea continutului fisierelor text

Permite prezentarea continutului unui fisier text (sau a unui grup de aemenea fisiere) in ordinea data de continutul liniilor sau a unor cimpuri din liniile fisierului .

Comanda: **sort**

Sintaxa:

sort [-bdfiMnr] [-tc] [sort- field….] [-cmu] [-o output file] [-T directory] [-y kmen] [-z recsz] filename

Parametru obligatoriu: **fisierul**

- Forma sortata e afisata implicit in **fisierul standard de iesire**;
- Poate fi specificat un fisier care sa contina forma sortata prin optiunea **-o**
- Campurile din linie dupa care se face sortarea e indicata prin optiunea sort-field care are forma unor nr intregi (0, 1 2,3..) precedate de + sau -:
  - +nr: nr de ordine al campului de la care incepe sortarea
  - nr: nr de ordine al campului unde se opreste sortarea
- Separatorii de campuri:
  - "spatiile albe"
  - specificate de optiunea **–tc: c** = caracterul folosit
- E util in sortare ca valoarea unui anumit camp sa fie interpretata ca numar si nu ca sir de caractere: optiumea n atasata la numarul de ordine al campului

Exemplu de utilizare:

1) Sortarea fisierului de parole dupa valoarea numerica a identificatorului de utilizator (UID), care este al **treilea cimp al fiecarei linii**:

$ sort -t: +2n -3 / etc/passwd

Obs: separatorul de cimpuri este **:**

- al-3-lea cimp e un numar

- : separator

- n valoarea cimpului astfel marcata e interpretata ca numar

2) sortarea unei liste de teleffoane dupa numele de familie, iar in cazul aceluiasi nume de familie dupa numele de botez:

Adrian Ilie	**02132333681**
Vasile Pop	**0256486221**
Anca Marcu	**0254234614**

$ sort +1 -2 +0 -1 listatelefoane

5. Editoare pentru fisiere text

ed: editor orientat pe linie: terminalul utilizatorului folosea hirtia ca mediu de afisare

- editarea are loc in cadrul liniei curente
- trecerea de la o linie la alta se facea prin c-zi

vi: editorul standard in UNIX: terminalul utilizatorului e ecranul

Utilitare derivate din ed:

sed si **s**

- **Expresii regulate**

Def: un sablon pentru siruri de caractere. Sunt cateva caractee care au o semnificatie speciala si se numesc metacaractere

. = orice caracter

**** = caracterul care urmeaza e tratat ca si caracter normal

Ex: daca in loc de adunarea efectiva a+b se doreste scrierea textului se foloseste a\+b

^ = sablonul se aplica la sfarsitul liniei

Ex: **^…$** inseamna o linie formata din exact 3 caractere

^\ . inseamna linie ce incepe cu caracterul .

[lista] = se va folosi unul din caracterele din lista

Ex: **[a-d]** se folosesc caracterele a,b,c, sau d.

[^lista] = se va folosi orice caracter in afara de cele din lista

+ * ? | ( ) = se folosesc in expresii regulate

- Exemplu:
  - **e+** = sir de caractere format din unul sau mai multe caractere e
  - **e*** = sir de caractere format din unul sau mai multe caractere e sau sirul vid
  - **e?** = sir vid sau sir format dintr-un sir reprezentat de expresoia e
  - **e1 e2** = concatenare siruri reprezentate de expresiile e1si e2.
  - **e1|e2** = sir format din sirurile reprezentate de expresia e1sau e2

- Expresiile regulate se pot incadra intre ghilimele sau **apostrofuri (in acest caz matacaracterele nu sunt luate in considerare).**

Comanda: sed ("stream editor"- editor in flux)

Obs: in linia de c-da se specifica ce operatie se doreste a fi executata de catre sed si asupra carui fisier. Este posibil ca sed sa-si preia c-zile dintr-un fisier.

Sintaxa:

sed [-n] [-e] ***command file(s)***

sed [-n] -f ***scriptfile file(s)***

Obs:

- rezultatul e trimis in fisierul **standard de iesire**;
- optiunea **–n** suprima iesirea; se vor afisa numai liniile specificate cu c-da **p** a editorului
- optiunea **–e** arata ca urmatorul argument este o c-da si se utilizeaza cind se specifica mai multe c-zi in aceeasi linie;
- optiunea **–f** arata ca urmatorul argument este un fisier care contine comenzi de editare.

Sintaxa c-zilor de editare pt **sed**:

[addresss [, address]] [!] command [arguments]

Efect:

- Editorul **sed** ia pe rind liniile din fisierul specificat ca intrare si le aplica fiecareia c-da data, trimitind rezultatul in fisierul standard de iesire (daca iesirea nu e suprimata);
- Daca lipseste partea de adresa, c-da are efect asupra fiecarei linii din fisierul specificat;
- Daca se specifica numai o adresa, c-da se aplica liniilor care corespund acelei adrese (adresa poate fi exprimata printr-un fisier);
- Doua adrese separate prin virgula in cadrul c-zii fac ca aceasta sa se aplice liniilor unui fisier incepind cu prima care satisface prima adresa ….
- Daca o adresa e urmata de semnul exclamarii c-da se aplica liniilor care nu satisfac tiparul de adresa.

Tiparurile de adresa constau din numere sau simboluri puse intre caractere **"slash".**

Exemple:

1, 30 denota liniile 1 la 30;

/^$/ denota toate liniile goale;

1, / ^$/ denota liniile de la prima din fisier pina la prima linie goala;

/SAVE/! denota liniile care nu contin SAVE.

C-da cea mai frecvent utilizata este cea de substitutie **s**

Sintaxa:

s/pattern/replacement/[flags]

Obs Tiparul (pattern) comenzii consta din siruri de caractere care se reprezinta pe ele insele sau din expresii regulate construite dupa o sintaxa relativ complexa.

- Prin ***** se reda orice sir, dar un caracter si numai unul se indica prin **(.)** ceea ce face ca pt. specificarea punctului ca si caracter obisnuit sa fie necesar **\.**

Principalele fanioane:

g (global) inlocuieste toate aparitiile tiparului in linia prelucrata;

n inlocuieste a n-a aparitie a tiparului (implicit n are valoarea 1 si ia val pina la 512);

p tipareste linia daca s-a produs o inlocuire. Inlocuiri multiple in linie duc la tipariri multple;

Exemple:

sed s/Unix/ UNIX/g czi.tex
sed s/^$/"Linie goala"/ test.txt

(1) **inlocuieste** in fisierul czi.tex Unix → UNIX; Rezultatul e afisat in fisierul standard de iesire. Fisierul c-zi.tex ramine nemodificat. Pt a salva continutul modificat trebuie redirectata iesirea standard catre un fisier diferit de **czi.tex.**

(2) Liniile goale din test.tex se inlocuiesc cu linii care contin textul **"linie goala".**

Alte c-zi ale lui **sed**:

d – sterge liniile de adresa;

a - adaugarea de text dupa o linie;

i - inserarea de text inaintea unei linii.

Comanda: grep ("general regular expression processing")

Obs:

- frecvent utilizata pt cautarea de tipare in fisiere text.
- opereaza la nivel de linie.

Sintaxa generala a c-zii:

grep [options] regexp [file(s)]

Sintaxa expresiilor regulate aceeasi ca si la sed

Efect: In mod normal, afiseaza in fisierul standard de iesire liniile care contin un sir ce corespunde expresiei regulate

Optiuni:

-c tipareste numai liniile care corespund tiparului

-i ignora deosebirea: litere mari, litere mici

-v tipareste liniile care nu corespund expresiei regulate

-n tipareste liniile insotite de numarul lor de ordine in fisier

Exemple;

Presupunem ca fisierul telefon.txt contine liniile

Sorin Marian: 2201987
Marin Pop: 7765770
Sorin Amariucai: 1213453
Sorina Marineasa: 234532
Maria Popescu: 2233114
Ion Sorinica: 4560432

grep Sorin telefon.txt *# afiseaza liniile care contin sirul Sorin, deci 1 3 4 6*

grep –n Sorin telefon.txt *# afiseaza liniile care contin sirul Sorin precedate de numarul liniei*

grep –c Sorin telefon.txt *# sau*

grep Sorin telefon.txt | wc –l *# afiseaza numarul de linii care contin sirul Sorin, deci 4*

grep "^Sorin" telefon.txt *# afiseaza liniile in care Sorin e radacina prenumelui*

grep –vi mari telefon.txt *# afiseaza liniile care* ***nu contin*** *sirul* **mari** *nefacind distinctie intre literele mari si mici, deci linia 6*

ls -l | grep ^- # *afiseaza numai liniile care incep cu -, deci fisierele ordinare din fisierul curent*

O uitlizare tipica:

ps aux | grep netscape

Efect: se urmareste aflarea identificatorului de proces sub care ruleaza **browser-ul netscape,** cu scopul de a termina executia programului, care s-a blocat si nu mai raspunde.

6. Cautarea de tipare si editarea de rapoarte

Instrumente care sa poata extrage informatii din baze de date cu organizare relativ simpla.

awk : editor programabil care poate interpreta o secventa (script) de comenzi de editare - **limbaj de programare awk(Aho, Weinberger Kernighan)**

Azi: **Perl (Practical Extraction and Report Language)**

Comenzi pentru operatii asupra perifericelor

1. Conversia si copierea fisierelor cu diferite formate

Comanda: **dd**

Obs. Desi accepta ca parametrii orice fisiere ea este utilizata de regula pt scrierea directa in fisiere speciale, in speta ***discurile magnetice***.

Sintaxa:

dd [option=value]

Perechi (optiune, valoare)

if=name - specificarea fisierului de intrare(sursa). Implicit-fisierul standard de intrare;

of=name - specificarea fisierului de iesire(sursa). Implicit-fisierul standard de iesire;

ibs=n - dimensiunea blocurilor in fisierul de intrare(implicit 512)

obs=n - dimensiunea blocurilor in fisierul de iesire;

files=n - copie n fisiere de intrare(daca intrarea este o banda magnetica)

count=n - copie numai n inregistrari din fisierul de intrare.

Obs: In anumite situatii se poate folosi si c-da **cp pt copiere la nivel fizic.**

In Linux:

$ cp zImage /dev/fd0 #imagine executabila a nucleului sistemului

2. **Arhivarea si restaurarea fisierelor**

Comanda: **tar** ("**tape archiver**")

Utilizare: salvari/restaurari pe discuri magnetice.

3. Crearea unui sistem de fisiere

Comanda: **mkfs (make filesystem)**

Obs: Pt linux **mkfs** este doar un “front end” catre programe specifice tipurilor de sisteme de fisiere acceptate si are sintaxa:

mkfs [-V] [-t fstype] [fs-options] filesys [blocks]

Optiuni:

-V (verbose) determina afisarea detaliata a informatiilor despre actiunile executate (este implicita).

-t specifica tipul de sistem de fisiere dorit (*ext2* sau *minix).*

-**fs**-**options** desemneaza optiuni specifice fiecarui tip de sistem de fisiere

-filesys este numele partitiei sau unitatii de disc flexibil unde se creaza sistemul de fisiere;

-**blocks** desemneaza dimensiunea dorita pentru sistemul de fisiere(implicit se utilizeaza toata partitia)

EXEMPLU: Crearea pe un disc flexibil a unui sistem de tipul ***ext2, cu blocuri de 2048 octeti, cu numai 64 i-noduri si folosind doar 1200 kocteti(in loc de 1440 octeti):***

$ mkfs -b 2048 -N 64 /dev/fd0 600

mke2fs 1.15, 18-Jul-1999 for ext2 FS o.5b, 95/08/09

Obs- optiunile –b si –N sunt specifice programului mke2fs folosit implicit de mkfs.

4. Verificarea unui sistem de fisiere

Scop: refacerea consistentei unui sistem de fisiere dupa un incident care nu mai permite refacerea automata a consistentei.

Comanda: **fsck(filesystem check)**

Situatii de consistenta:

- La stergerea unei intrari dintr-un catalog, noul continut al catalogului este scris sincron pe disc(fara a utiliza tampoane si scrierea efectiva pe disc mai tirziu). Urmeaza decrementarea numarului de legaturi cu 1 si eventual eliberarea spatiului ocupat de fisier (daca nr. De legaturi a ajuns la 0)
- Daca apare o cadere a sistemului inainte de actualizarea nodului index, sistemul de fisiere ramine inconsistent: i-nodul va avea continutul cu 1 mai mare. Se poate corecta de catre programul de verificare care aduce nr. de legaturi la valoarea corecta egal cu nr total de intrari de catalog care contin acelasi nod index.

Comenzi diverse

1. Comprimarea /decomprimarea fisierelor

Sintaxa

compress [-cvf] [-b bits] [filename]

Efect: Inlocuieste fisierele date ca argumente cu fisiere cu acelasi nume la care se aduga sufixul **.z**.

Valoarea factorului de comprimare depinde de dimensiunea intrarii, de numarul de biti per cod si de distributia unor subsiruri comune.

Optiuni:

-c face ca fisierul sa fie scris in fisierul standard de iesire fara a afecta fisierul sursa(ca si la c-da **zcat**).

-v afiseaza procentajul de reducere pentru fiecare fisier.

-b se fixeaza limita superioara in biti pt codurile de subsiruri comune. Are valoare implicita 16. Subsirurile comune din fisierul supus comprimarii sunt inlocuite cu coduri de 9 biti (de la 257 in sus). Cind se ajunge la 512 se trece la coduri de 10 biti pina se ajunge la coduri de **bits** biti.

Compress verifica periodic factorul de comprimare si procedeaza:

- daca factorul creste, se folosete dictionarul de coduri existent;
- daca factorul scade, se renunta la tabela de subsiruri si se construiete una complet noua.

Noi c-zi:-**gzip** si **gunzip (** lucreaza cu fisiere cu sufixul **.gz**, dar prelucreaza la decomprimare si fisiere cu sufixul **.Z);**

- **bzip2 si bunzip2**(care lucreaza cu fisiere cu sufixul **.bz2);**

2. Cautarea fisierelor dupa nume

Sintaxa:

find pathname-list expression

Efect: pornind de la fiecare nume dat in pathname-list se cauta recursiv fisiere care corespund lui **expression** considerata ca expresie logica.

Operatorii care pot apare in expression:

a) **–name filename**: are “val true” daca argumentul corespunde cu fisierul curent in functionarea lui find. Se accepta si nume generice de fisiere:

- name “*.txt”- va fi “true“ pt toate fisierele .txt din ierarhia de fisiere in care se face cautarea.

b) **–perm onum**: “true” daca drepturile de acces la fisier corespund exact nr octal onum.

c) **-type c** - “true” daca tipul fisierului este **c**(f,d,b,c,p,s).

d) **–user uname** – “true” daca proprietarul fisierului este uname.

e) **–atime n** - “true” daca fisierul nu a fost accesat de n zile.

f) **–mtime n** - “true” daca fisierul nu a fost modificat de n zile.

g) **–catime n** - “true” daca fisierul nu a fost schimbat de n zile (modificarea atributelor sale).

h) **–print** – intotdeauna adevarat, afiseaza numele fisierului curent.

i) –ls - intotdeauna adevarata, afiseaza numele de cale al fisierului curent si info suplim despre acestea (analog c-zii **ls –li**).

j) **–prune** - intotdeauna adevarata are ca efect colateral oprirea cautarii la numele de cale curent, daca acesta e catalog (nu se cauta recursiv).

k) **–exec command** – “true” daca valoarea returnata de **command** este 0. Sfirsitul comenzii este marcat de caracterele **\ ;** . Argument al c-zii este si numele de cale curent indicat **{ }**.

Exemplu:

Stergerea tuturor fisierelor cu sufixul **.o** situate in catalogul curent sau in subcataloagele acestuia.

$ find . – name “*.o” –exec rm {} \ ;

3. Evaluarea expresiilor

Scop: ***Evalueaza o expresie si scrie rezultatul in fisierul standard de iesire. Fiecare atom din expresie apare ca un argument distinct(trebuie separat de alti atomi prin spatii).***

Operanzii sunt fie numere, fie siruri de caractere. Comanda forteaza transformarea tuturor operanzilor in valori intregi sau in siruri de caractere, functie de operatia care trebuie efectuata.

Operatorii apar fie sub forma de ***simboluri infix*** fie sub forma de ***cuvinte cheie prefix***. Este permisa utilizarea parantezelor pentru grupari, dar ele trebui citate.

Sintaxa:

expr *arg1 operator arg2 [operator arg3…..]*

Efect: **codul de retur** al c-zii este **0** daca valoarea expresiei este diferita de 0 sau de null, 1 daca expresia are valoarea 0 sau null, 2 daca expresia nu e valida

Operatori matem acceptati: **+, -, *, / si%.**

Operatori relationali: **=,!=,>,>=, < si <=.**

Daca operanzii sunt numerici se fac comparatii aritmetice, altfel comparatii lexicografice.

- Pot fi utilizati si operatori logici:
  - | SAU logic: daca primul operand {arg1} este nenul rezultatul este arg1, altfel este arg2
  - & SI logic: daca ambii operanzi sunt nenuli, rezultatul este arg1, altfel rezultatul e 0.
  - : arg2 se interpreteaza ca tipar care se cauta in arg1. Daca arg2 este incadrat de parantez rezultatul este portiunea din arg1 care corespunde tiparului, in caz contrar rezultatul consta din numarul de caractere care corespund

- Exemple
  - $ expr 5+10 / 2 da rezultatul 10
  - $ expr \(5+10\) / 2 da rezultatul 7
  - $ i = ’expr $i” +1’ # incrementeaza cu 1 variabila de mediu
  - Pentru exemplele urmatoare se considera variabila de mediu v are valoare program.c

- $ expr ”$v” : ‘.*’ # da rezultatul 9 numarul de caractere din
- $ expr ”$v” : ‘\(.*\)’ # da rezultatul program.c
- $ expr ”$v” : ‘\([a-z].*\)’ # da rezultatul program.

Interpretorul de comenzi (SHELL)

Stabileste ambianta sub care este lansat in executie programul utilizator.

Pentru shell-ul lansat in executie la intrarea in sesiune a unui uitilizator (**"login shell")** ambianta este stabilita implicit de sistemul de operare.

Program ciclic care executa iterativ urmatoarele actiuni:

- Citeste din fisierul de intrare;
- Stabileste ambianta de executie a liniei;
- Laseaza in executie fiecare comanda ca proces distinct (**fiu al procesului care corespunde interpretorului**);
- Asteapta terminarea executiei comenzilor lansate;
- Reia ciclul de la prima;

Obs Ciclul prezentat este **reluat pentru fiecare linie de comanda** si extins cu o serie de fcailitati de programare (**constructii pentru controlul fluxului**).

Facilitati SHELL fundamentale

Pt stabilirea ambiantei de executie a unei c-zi shell-ul executa in ordine urmatoarele:

- Imparte linia de c-da in "cuvinte"
- Interpreteaza **caracterele de citare** (**ghilimelele simple si duble**)

- Redirecteaza intrarile si iesirile;
- Substituie variabilele de mediu;
- Substituie comenzile;
- Expandeaza numele de fisiere;

Dupa terminarea acestei actiuni este lansata in executie comanda respectiva.

Exemplu:

Listarea numelor fisierelor sursa C din catalogul gazda care au fost create in 18 noiembrie a anului curent:

$ ls –l $HOME/*.c | grep "Nov18"

Efect:

1. Imparte linia in cuv: **ls**, **-l**, **$HOME/*.c**, **|** , **grep** si **"Nov18";**
2. Recunoaste ghilimelele
3. Recunoaste un caracter pt redirectarea intrarii si iesirii **|** . Face aranjamentele necesare pt ca iesirea comenzii **ls** sa devina intrare pentru comanda **grep** .
4. Recunoaste variabila de mediu **$HOME** care desemneaza **catalogul gazda al utilizatorului** si inlocuieste valoarea acestei variabile (de expl: **/home/tempus/profs/ada**) in linia de c-da .
5. Recunoaste numele generic ***.c** si expandeaza aceste nume producind o lista de fisiere care va fi introdusa in linia de c-da.

Shellul creaza 2 procese fii: unul pt a executa c-da **ls**, celalalt pt a executa c-da **grep** si lanseaza in executie cele doua procese

Mecanisme de citare in Shell

Caractere speciale:

**# & * ? [ ] ( ) = | ^ ; < > ` $ “ ‘ **

Separatori de cuvinte: spatiu, tab, newline

Mecanismele de citare au rol de a dezautoriza semnificatia caracterelor speciale(dar nu afecteaza separatorii)

**# & * ? [ ] ( ) = | ^ ; < > ‘ $ ’ “ **

‘text’ dezautorizeaza toate caracterele speciale din ***text***

“text” dezautorizeaza caracterele speciale din ***text*** cu exceptia **$ ‘ si \.**

\c dezautorizeaza caracterul special c. La sfirsit de linie elimina un *newline*.

Exemplu :

Linia de c-da $ echo 45 * 3 **Da rezultatul 45*3**

$ echo 45 *3 **echo: syntax error**

$ echo Hey! What’ s next? Mike’ s #1 friend has $$.

Produce:

Hey! Whats next? Mikes

Obs

Elimin nr mare de spatii

Disparitia celor 2 apostrofuri= caractere de citare

? e intre apostrofuri si isi pierde semnificata de caracter special

marcheaza pt shell inceputul unui comentariu- **e ignorat restul liniei**

$ echo “ Hey! What’ s next? Mike’ s #1 friend has $$.”

Produce:

Hey! What’s next? Mike’s #1 friend has 23812.

Obs

Spatiile din textul intre ghilimele sunt pastrate la fel si apostroful

$** isi pastreaza semnificatia speciala⟶ **$$ desemneaza valoarea **variabilei de mediu $, in care se pastreaza PID-ul shell-ului,** valoarea acestuia se inlocuieste in text

Redirectarea intrarilor si iesirilor

Exista valori implicite pentru argumentele cu semnificatia:

fisier de intrare(tastatura)- intrarea standard

fisier de iesire (terminalul)- iesirea standard

Shell-ul stabileste identitatea fisierelor standard de intrare si de iesire pe care le ataseaza fiecarei comenzi → **Acestea primesc descriptorii de fisier 0 (intrare) si 1 (iesire) in procesul care e utilizat pentru executia comenzii.**

Descriptorul 2 – pt erori (de regula e acelasi cu fisierul standard de iesire)

Shell-ul poate schimba identitatea fisierelor standard prin: caracterele **<**, respectiv **>**.

Exemplu:

$ cat main.c >program.c

Efect: listarea fisierului **main.c** din catalogul curent nu apare pe ecran ci se memoreza in fisierul **program.c** din catalogul curent.

Obs: shell-ul deschide in scriere fisierul **program.c** anterior lui **cat** si ii acorda **descriptorul 1** in procesul care va executa **cat.**

Comanda in care se redirecteaza atit intrarea cit si iesirea:

$ command <input >output

Bourne shell:

n> redirectarea pt scriere a fisierului cu **descriptorul n** (*pt fis std de eroare*).

Comanda in care se redirecteaza toate cele trei fisiere

$ command <input >output 2> errors

Obs pt a evita suprascrierea se foloseste **>>;** Expl. 2>>

$ command >>logfile

Efect: mesajele produse de command se adauga la sfirsitul lui **logfile(fisiere jurnal).**

Caz special prin redirectare: documentele here: <<c

urmatoarele caractere necesare unei c-zi se preiau:

- de la tastatura (in regim interactiv) sau
- din textul unei proceduri **shell**

pina la intilnirea lui **c**(caracter sau text).

Mecanismul de redirectare prin **pipe(canal)**

Sintaxa:

$ command1 | command2

Efect: comenzile command1 si command2 vor fi lansate in executie simultan:

command1→ command2

Shell-ul: specifica drept iesire, descriptor1 (pt command1), un "fisier"(o structura pipe) = descriptor 0 pt command2

Folosit in UNIX pentru combinarea unor c-zi

Exemplu: din iesirea c-zii ps sa se selecteze numai procesele care apartin unui utilzator si acestea sa se afiseze pe ecran:

$ ps -aux >temp

$ grep ada temp

Comanda cu pipe:

$ ps -aux | grep ada

Variabile de mediu

- Pastreaza o mare parte din info necesare pentru stabilirea ambiantei de executie a comenzilor.

- Caract: au ca si valori siruri de caractere si sunt initializate prin c-zi de forma;

name=value

- Referirile la variabilele de mediu(sau variabile shell) se fac precedind numele variabilei de $;

Exemple:

$ VAR=test

$ echo $VAR

test

$

Obs. Pt ca variabila de mediu sa ramina valabila pe intreaga durata a sesiunii unui utilizator si nu numai pe durata shell-ului, ea trebuie exportata. Atribuirea se face sub forma:

name=value; export name

Variabile de mediu

C-da Unix env produce listarea tuturor variabilelor de mediu existente in momentul emiterii ei si se poate utiliza pt a ilustra diferenta intre variabilele exportate si cele neexportate.

- Pt fiecare program respectiv in fiecare procedura Shell se pot defini variabile de mediu specifice.

- **Exista variabile de mediu predefinite care pot fi utilizate in toate** programele sau comenzile. Valorile acestor variabile sunt stabilite de sistem la intrarea in sesiune, dar pot fi modificate de utilizator pe parcurs.

Variabile de mediu

Lista variabilelor predefinite:

- **PATH** - lista cataloagelor in care se cauta c-zi. Atribuirea de valoare pt aceasta variabila:

  PATH=.:./bin:$HOME/bin:/bin:/usr/bin

Obs: Elementele listei sunt separate prin **:**

- Ordinea in lista este ordinea in care se face cautarea.

- Numele c-zii se cauta mai intii in catalogul curent . Apoi in subcatalogul bin al catalogului curent(./bin), apoi in subcatalogul bin al catalogului gazda al utilizatorului , in catalogul /bin si apoi in catalogul /usr/bin.

Adaugarea de noi cataloage in lista de cautare:

PATH=$PATH:/usr/local/bin

Variabile de mediu

- **HOME –** catalogul gazda al utilizatorului, argument implicit pt **cd.**
- **MANPATH-** lista cataloagelor in care se cauta paginile de manual on-line de catre comanda **man**

  MANPATH=/usr/man:/usr/local/man
- **TERM – indica tipul terminalului folosit in sesiunea curenta**
- **MAIL – numele de cale pentru cutia uitlizatorului**
- **PS1 – sirul folosit ca prompt: utilizatori- $; superutilizator #**
  - **se poate redefini de utilizator**
- **PS2 – sirul prompt suplimentar, utilizat cind o c-da se intinde pe mai multe linii(val traditionala >)**
- **$ - numarul procesului (PID) pt acest shell.**
- **# nr de argumente din linia de comanda**
- **0 (zero) – numele comenzii executate**
- **n – pt valori ale lui n intre 1 si 9 reprezinta al n-lea argument din linia de c-da.**

Substitutia comenzilor

Iesirea standard produsa de o c-da poate fi substituita in locul in care apare c-da respectiva daca c-da e incadrata de **‘….’**

Exemplu: un utilizator doreste sa trimita un mesaj prin posta electronica tuturor utilizatorilor aflati in sesiune pe acelasi calculator ca si el.

% mail ‘ who | cut –c1-8 | sort –u ’

Comentariu: ‘ argument ’

who - da lista sesiunilor deschise,

cut - se retin primele 8 coloane din linie adica numele utilizatorilor

-u – se face sortarea numelor cu eliminarea duplicatelor

Facilitati de programare in SHELL

Pe linga executia c-zilor citite de la terminal, shell-ul poate controla si executia unor comenzi citite dintr-un fisier(*proceduri shell* ,“shell scripts”)

Important!!!

Pot contine pe linga c-zi Unix propriu-zise si structuri pentru controlul fluxului si pot avea argumente preluate din linia de c-da prin care ele se lanseaza in executie.

Lansarea in executie a fisierelor cu proceduri shell

$ sh nume_fis argumente

Sau:

$ nume_fis argumente

Obs nume_fis trebuie sa fie fisier executabil, adica dupa crearea sa printr-un editor de text, trebuie emisa c-da

$ chmod a+x nume_fis

Structuri pentru controlul fluxului comenzilor

A. Controlul secvential

Sructuri pentru inlantuirea sau gruparea c-zilor

Operatori: **; – delimitator de comenzi**
( ) – gruparea c-zilor
&& – testeaza pt cod de retur true si executa
| | – testeaza pentru cod de retur false si executa

; - delimitator de comenzi –permite scrierea mai multor c-zi secventiale in aceeasi linie

cmd1; cmd2; cmd3

Efect: trecerea la executia c-zii urmatoare se face numai dupa terminarea precedentei din aceeasi linie.
Prin pipe (|) pot fi lansate simultan in executie, in procese separate mai multe c-zi.

Structuri pentru controlul fluxului comenzilor

( ) – gruparea c-zilor – permite combinarea fisierelor standard de iesire si de eroare ale mai multor comenzi intr-un singur flux pt simplificarea prelucrarii

Exemplu:

(cmd1; cmd2 | grep) | wc

EFECT: pt comanda wc fisierul de intrare e constituit din fisierul de iesire produs de cmd1 la care se adauga(>>) rezultatul grupului de c-zi cmd2 | grep

Detaliat:

cmd1 >/tmp/tmp$$; cmd2 | grep >> /tmp/tmp$$

wc /tmp/tmp$$

&& - testeaza pt cod de retur true si executa

cmd1 && cmd2

Efect: executia c-zii cmd2 are loc numai daca prima c-da cmd1 produce un cod de retur nul (se executa fara erori)

| | - testeaza pentru cod de retur false si executa

cmd1 | | cmd2

Efect: executia c-zii cmd2 are loc numai daca prima c-da cmd1 produce un cod de retur diferit de zero

Structuri pentru controlul fluxului comenzilor

B Structuri alternative

if-then-else

Sintaxa:

```
if  [ test-conditions ]
then cmd1
else cmd2
fi
```

Constructia [test conditions] ⇔ executia unei instructiuni sau secvente de instructiuni care poate returna:

o valoare nula (true) **=> se executa cmd1** sau

nenula (false) **=> se executa cmd2** (daca e prezenta, ea fiind optionala)

Obs: **Conditiile testate la if-then else sunt trecute in tabele**.

Conditii testate la *if-then-else*

Conditie	Descriere
-f *fisier*	*fisier* exista si e fisier obisnuit
-r f*isier*	*fisier* exista si are drept de citire
-w *fisier*	*fisier* exista si are drept de scriere
-x *fisier*	*fisier* exista si are drept de executie
-d *catalog*	*catalog* exista si are tipul catalog
-s *fisier*	*fisier* exista si are lungime nenula
-z *sir*	*sir* are lungime nula
-n *sir*	*sir* are lungime nenula
sir1 = sir2	cele doua siruri sunt egale
sir1 != sir2	cele doua siruri nu sunt egale
"sir"	*sirul* dintre ghilimele nu este vid
n1 –eq *n2*	numerele *n1* si *n2* sunt egale
n1 –ne *n2*	numerele *n1* si *n2* nu sunt egale
!*l*	*l* are valoarea logica false

Conditii testate la *if-then-else*

Conditie	**Descriere**
l1 –a *l2*	*l1* si *l2 sunt true*
l1 –o *l2*	*l1* sau *l2 sunt true*
(…)	grupare pentru asigurarea precedentei

Exemplu de utilizare:

Verificarea daca o variabila de mediu este definita si afisarea valorii acesteia. Daca variabila nu e definita (sirul care constituie definitia ei este vid) se va afisa un mesaj de eroare.

```
if [ "$JAVA_HOME" ]
then echo $JAVA_HOME
else echo "JAVA_HOME not defined."
fi
```


Structura alternativa:

```
case $variable in
vallist1) action1 ;;
vallist2) action2 ;;
…
*) defaultaction ;;
esac
```

Efect:

Valoarea variabilei care apare ca discriminant se compara pe rind cu valorile specificate in **vallist1**, **vallist2**.

La identificarea unei corespondente se executa actiunea asociata.

Dca valoarea variabilei nu corespunde nici unei valori din lista, se executa actiunea asociata tiparului generic *.

Exemplu: comanda de adaugare la sfirsitul unui fisier

```
case  $#  in
      1)  cat >>$1 ; ;
      2)  cat >>$2  <$1 ; ;
      *)   echo  'usage:  append  [from]  to '  ;;
esac
```

Obs: - variabila de mediu # indica numarul de parametrii in linia de comanda;

- daca e 1 parametru → continutul fisierului standard se adauga la sf fisierului al carui nume este dat prin acest parametru;
- daca sunt 2 parametrii → continutul fisierului indicat de primul parametru e adaugat la sfirsitul fisierului indicat de al-II-lea parametru;
- pt alte situatii se emite un mesaj de utilizare;

C. Structuri de ciclare

Structura **for** permite executia ciclica a unei liste de actiuni(comenzi) , modficind la fiecare parcurgere valoarea unei variabile specificate prin **for.**

```
for variable  [ in listvalue]
do
      action
done
```

Daca partea optionala lipseste, ***variabile*** ia pe rind ca valori argumentrele prezente in linia de c-da.

Exemplu: stuctura **for** pentru crearea unui nr de fisiere;

```
for  i  do  >$i;  done
```

Obs lipseste partea optionala, iar done e folosit pt ca e prezent separatorul ;

```
for  i  in  *
do
      echo $i
done
```

Efectul: afisarea numelor fisierelor din catalogul curent (cu exceptia celor a caror nume incepe cu '.')

Structuri pentru controlul fluxurilor comenzilor

Structura de ciclare **while**

Sintaxa:

```
while  command-list1
do command-list2
done
```

Obs: - valoarea testata de c-da while este codul returnat de ultima c-da simpla din command-list1. Daca aceasta e 0, se executa command-list2, dupa care se reia executia lui command-list1. Daca se returneaza o valoarea diferita de zero, ciclul de tremina.

O forma inrundita este c-da until, in care conditia de terminare a ciclului este inversa fata de while:

```
until  command-list1
do  command-list2
done
```

Comenzi interne interpretorului de comenzi shell

Controleaza modul de executie al procedurilor shell

A. C-da shift

Efect: eliminarea primului argument din linia de comanda si deplasarea spre stinga a argumentelor ramase: $2 devine $1 etc.

Exemplu de utilizare

```
while  [ $# -gt 0 ]
do echo
    shift
done
```

Obs: -se utilizeaza variabila de mediu # care are ca val. nr de param din linia de c-da

- efectul lui shift = reducerea cu 1 a nr de parametrii

Comenzi interne interpretorului de comenzi shell

B. Comanda interna read

Efect: citeste cite o linie din fisierul standard de intrare si atribuie cuvintele din linie unor variabile de mediu:

read [-r] [name…]

- Cuvintele din linia citita se atribuie pe rind variabilelor din lista **name**
- Daca nr de cuvinte din linie e mai mare decit nr de variabile, valoarea primita de ultima variabila cuprinde toate cuvintele ramase.
- Daca e prezenta op **–r ,** o pereche -newlinie nu e ignorata, iar backslash se considera parte din linie => se pot specifica valori pe mai multe linii.
- Daca lista de nume lipseste, cuvintele citite se atribuie variabilei REPLY
- Returneaza codul 0, cu exceptia cazului cind se ajunge la sfirsitul fisierului

Comenzi interne interpretorului de comenzi shell

Exemplu: Procedura shell prin care se citeste din fisierul standard de intrare (tastatura) continutul unui fisier care sa reprezinte o agenda telefonica

```
while read nume prenume telefon
do
        echo –n $nume “ “ $prenume ” “  $telefon “ “ >>agenda
        echo >>agenda
done
```

Comenzi interne interpretorului de comenzi shell

C. Comanda break

Efect: - determina parasirea unui ciclu **for**, **while** sau **until** si returneaza **0** cu exceptia cazului cind shell-ul nu executa un ciclu in momentul unui **break**.

break n

Obs. Nr de cicluri incuibate din care se iese.

D. Comanda **continue**

Efect: - **Determina trecerea la iteratia urmatoare a unui ciclu *for while* sau *until*.**

Continue n

→ ciclu exterior

Comenzi interne interpretorului de comenzi shell

E. Comanda:

eval [args…]

Efect: Determina concatenarea argumentelor si executia de catre shell a c-zii astfel obtinute. Se returneaza ca valoare a comenzii **eval** codul de retur al c-zii executate**.** Daca nu exista argumente, **eval** returneaza **true.**

F. Exista c-zi care functioneaza atit sub forma de c-zi interne cit si sub forma de c-zi externe**: cd, pwd, test, echo, type etc.**

Pt a contorla autorizarea sau dezautorizarea c-zilor interne este prevazuta **enable:**

enable [-n] [-all] [name…]

- Optiunea **–n** indica dezautorizarea comenzilor interne al caror nume este specificat prin ultimul argument.
- Daca se foloseste numai **–n** se obtine o lista a comenzilor momentan dezautorizate
- Daca se foloseste **–all** se obtine o lista a tuturor c-zilor interne.

Comenzi interne interpretorului de comenzi shell

G. Comanda:

set [--abefhkmnptuvxldCHP] [-o option] [arg…]

Efect: -permite specificarea unor conditii de functionare a shell-ului. Fanioanele c-zii sunt in mod implicit dezautorizate.

- daca se specifica un fanion precedat de – el este autorizat;
- utilizarea c-zii fara nici un argument produce listarea numelor si variabilelor de ambianta momentan definite.

Efect fanioane:

-a marcheaza automat pt export in ambianta urmatoarelor comenzi toate variabilele modificate sau create.

-b face ca stare joburilor de fundal sa fie raportata imediat si nu doar inainte de urmatorul prompt principal;

-f dezautorizeaza expandarea numelor de cale.