

Filtre

Utilizarea Sistemelor de Operare

Paul Irofti

Universitatea din București
Facultatea de Matematică și Informatică
Department de Informatică
Email: paul.irofti@fmi.unibuc.ro

Vocabular

- ▶ filtru – program, comandă, operație care procesează ieșirea în format text a unei comenzi astfel încât noua formă să poată fi procesată mai departe de utilizator sau de un alt program
- ▶ linie – șir de caractere care se termină cu `\n` (sau `\r\n` în Windows)
- ▶ separator (sau delimitator) – caracter sau șir folosit pentru a delimita datele într-o linie
- ▶ câmp (*field*) – un subșir dintr-o linie care reprezintă un anumit tip de date
- ▶ exemplu:
 - ▶ linie: nume prenume grupă serie
 - ▶ separator: " "
 - ▶ câmpuri: nume, prenume, grupă și serie

cut(1)

cut(1) extrage câmpuri din fiecare linie primită la intrare

- ▶ `list` – numere sau intervale separate de virgulă sau spații
- ▶ `-b list` – lista conține poziții în bytes
- ▶ `-c list` – lista conține pozițiile caracterelor
- ▶ `-f list` – lista specifică câmpuri
- ▶ `-d delim` – specifică delimitatorul pentru câmpuri (implicit este `\t`)
- ▶ `-n` – nu împarte caractere multi-byte în bytes
- ▶ `-s` – sare peste liniile care nu conțin delimitatoare

Tipuri de apel

- ▶ `cut -b [-n] list [file ...]`
- ▶ `cut -c list [file ...]`
- ▶ `cut -f list [-s] [-d delim] [file ...]`

Exemple: cut(1)

Afișează numele și shellurile folosite de utilizatorii din sistem:

```
$ cut -d : -f 1,7 /etc/passwd
nobody:/sbin/nologin
paul:/bin/ksh
build:/bin/ksh
joe:/bin/ksh
_mysql:/sbin/nologin
_postgresql:/bin/sh
souser:/bin/ksh
alex:/bin/ksh
```

Exemple: cut(1)

Arată numele și data la care s-au logat utilizatorii activi:

```
$ who | cut -c 1-8,18-30
deraadt  May  8 18:32
dlg      May  3 20:39
jsing    Apr 28 06:47
landry   Apr 19 14:22
deraadt  Apr 19 08:24
kettenis May  9 02:47
deraadt  May  3 22:18
pirofti  May  9 04:36
```

Lipește fișierele primite la intrare pe coloane (pe verticală)

- ▶ `-d list` – folosește delimitatorul pentru a înlocui caracterul linie nouă `\n` din fiecare fișier
- ▶ `-s` – serializare
- ▶ `-` – reprezintă intrarea standard (`stdin`)

Apel

- ▶ `paste [-s] [-d list] file ...`

Exemplu: paste(1)

Fie fişierele cu nume şi prenume:

```
$ echo "Paul\nAlex\nAna" > firstnames.txt
$ echo "Irofti\nAlexandrescu\nPopescu" \
  > lastnames.txt
```

Implicit, paste(1) lipeşte numele de prenume

```
$ paste firstnames.txt lastnames.txt
Paul      Irofti
Alex      Alexandrescu
Ana       Popescu
```

Acelaşi lucru dar cu serializare

```
$ paste -s firstnames.txt lastnames.txt
Paul      Alex      Ana
Irofti    Alexandrescu      Popescu
```

Exemplu: paste(1)

Afișează fișierele din directorul curent pe trei coloane:

```
$ ls | paste - - -  
1      10      11  
2       3       4  
5       6       7  
8       9      intro
```

Identic cu apelul `ls | paste -s -d '\t\t\n' -`

Crează o listă de directoare bin din sistem separate prin :

```
$ find / -name bin -type d | paste -s -d : -  
/usr/X11R6/bin:/usr/local/bin:  
/usr/local/lib/qt4/bin:  
...
```


split(1)

Împarte fișierul dat în mai multe fișiere de 1000 de linii fiecare.

- ▶ `-a suffix_length` – câte litere să conțină sufixul noilor fișiere
- ▶ `-b byte_count` – crează fișiere de lungimea dată în bytes
- ▶ `-l line_count` – crează fișiere cu numărul de linii dat
- ▶ `file` – fișierul de împărțit, implicit este `stdin`
- ▶ `name` – prefixul pentru noile fișiere

Apel

- ▶ `split [options] [file [name]]`

Implicit crează fișierele `xaa`, `xab`, `xac`, ...

Exemplu: split(1)

Împarte fișierul LaTeX în mai multe fișiere de 100 de linii:

```
$ wc -l uso-curs-11.tex
    362 uso-curs-11.tex
$ split -l 100 uso-curs-11.tex
$ ls x*
xaa xab xac xad
$ wc -l x*
    100 xaa
    100 xab
    100 xac
     62 xad
    362 total
```

Același lucru dar cu prefix uso-curs-11 și o singură literă sufix:

```
$ split -a 1 -l 100 uso-curs-11.tex uso-curs-11
```

join(1)

Alătorează linii care conțin chei identice din două fișiere diferite

- ▶ `file1`, `file2` – fișierele de intrare
- ▶ `-1 field` – câmpul cheie din fișierul 1
- ▶ `-2 field` – câmpul cheie din fișierul 2
- ▶ `-a file_number` – produce o linie pentru fiecare nepotrivire din fișierul dat (1 sau 2)

Apel

- ▶ `join [-1 field] [-2 field] [options] file1 file2`

Funcționează în aceeași paradigmă cu comanda SQL JOIN.

Exemplu: join(1)

Alătură persoanele pentru care am date legate de vârstă și venit:

```
$ cat age.txt
Paul 33
Alex 40
Ana 25
$ cat income.txt
Paul 3000
Ana 4500
$ join age.txt income.txt
Paul 33 3000
Ana 25 4500
```

Același lucru dar include și persoanele (ex. Alex) fără venit:

```
$ join -a1 age.txt income.txt
```

Basic Calculator – bc(1)

Calculator pentru operații aritmetice și logice

- ▶ `-l` – permite operații cu numere în virgulă mobilă
- ▶ `-e expr` – evaluează expresia, pot fi mai multe
- ▶ `file` – preia expresii din fișier
- ▶ operatorii binari sunt la fel ca cei din C
- ▶ operatorii logici `&&`, `||` și `!` sunt disponibili în unele implementări dar nu sunt specificați de standardul POSIX
- ▶ reprezintă un limbaj de sine stătător cu blocuri de control (ex. `while`)
- ▶ implicit pornește un shell specializat și așteaptă comenzi

Apel

- ▶ `bc [-l] [-e expr] [file]`

bc(1) – funcții

- ▶ `s(x)` – sinus
- ▶ `c(x)` – cosinus
- ▶ `e(x)` – exponent
- ▶ `l(x)` – logaritm
- ▶ `a(x)` – arctangent
- ▶ `sqrt(x)` – radical
- ▶ `scale=n` – precizie, n numere zecimale
- ▶ `quit` – terminare program

Exemplu: bc(1)

Operații de bază:

```
$ echo "1/3" | bc
0
$ echo "1/3" | bc -l
.333333333333333333333333
$ echo "1>3" | bc -l
0
$ echo "1<3" | bc -l
1
$ bc -l -e "sqrt(2)" -e quit
1.41421356237309504880
```

Exemplu: bc(1)

```
$ bc -l
scale=4
sqrt(3)
1.7320
4*a(1)
3.1412
c(4*a(1))
-1.0000
c(4*s(1))
-.9750
quit
```


Translate – tr(1)

tr(1) traduce caractere primite din stdin și afișează rezultatul la stdout

- ▶ string1, string2 – caracterele din primul șir sunt traduse ca cele din al doilea string
- ▶ -C, -c – aplică complementul setului de caractere din string1
- ▶ -d – șterge caracterele ce apar în string1
- ▶ -s – elimină duplicatele traduse conform ultimul operand (fie string1 fie string2)

Tipuri de apel:

- ▶ tr [-Ccs] string1 string2
- ▶ tr [-Cc] -d string1
- ▶ tr [-Cc] -s string1

Exemplu: tr(1)

Crează o listă de cuvinte primite la intrare:

```
$ echo "Ana are mere" | tr -cs "[A-Za-z]" "\n"  
Ana  
are  
mere
```

Scrie textul primit cu majuscule

```
$ echo "Ana are mere" | tr "[a-z]" "[A-Z]"  
ANA ARE MERE
```

Obține șirurile de caractere dintr-un fișier binar:

```
$ echo "int main(){return 0;}" | cc -xc -o test -  
$ cat test | tr -cd "[:print:]"  
ELF>@@@8@@@@>>>>>>>HH H XX X @@TTTPtd44eHH H  
...
```

sed(1) este un editor de text în toată regula, mai mult este chiar un limbaj de programare!

- ▶ folosit în general pentru substituții și eliminări de text
- ▶ diferit de tr(1) folosește expresii regulate
- ▶ mod de funcționare
 - ▶ parcurge linie cu linie intrarea căutând un tipar
 - ▶ dacă a găsit un șir de caractere care respectă tiparul aplică funcția dată de utilizator asupra acestuia
 - ▶ funcțiile pot fi predefinite (substituție, eliminare) sau scrise de utilizator

Apel:

- ▶ `sed [options] command [file ...]`

Exemple: sed(1)

Substituție s,tipar,text,

```
$ echo "Paul Alex Ana" | sed s,Alex,Paul,  
Paul Paul Ana
```

```
$ echo "Paul Alex Alex Ana" | sed s,Alex,Paul,  
Paul Paul Alex Ana
```

Substituție s,tipar,text,flag

```
$ echo "Paul Alex Alex Ana" | sed s,Alex,Paul,g  
Paul Paul Paul Ana
```

```
$ echo "Paul Alex Alex Ana" | sed s,Alex,Paul,2  
Paul Alex Paul Ana
```

```
$ echo "Paul Alex Ana" | sed s,Alex,Paul,wfile  
Paul Paul Ana
```

```
$ cat file
```

```
Paul Paul Ana
```

Exemplu: sed(1)

Eliminarea liniilor care conțin tiparul:

```
$ echo "Paul Alex\nAlex Ana" | sed /Paul/d  
Alex Ana
```

Dacă vrem să ștergem doar șirul de caractere:

```
$ echo "Paul Alex\nAlex Ana" | sed s/Paul//g  
Alex  
Alex Ana
```

Aplicare doar anumitor linii:

```
$ echo "Paul\nAlex\nAlex\nAna\nPaul\nGeorge" | \  
sed 1,3s/Paul//g
```

```
Alex  
Alex  
Ana  
Paul  
George
```

Exemplu: sed(1)

Inversarea a două cuvinte folosind expresii regulate

```
$ echo "Paul Irofti\nAlex Pop\nGeorge Stan" | sed 's
    ,^\([A-Z][A-Za-z]*\) \([A-Z][A-Za-z]*\),\2 \1,'
Irofti Paul
Pop Alex
Stan George
```

- ▶ `^` – caută un tipar la începutul liniei
- ▶ `[A-Z]` – trebuie să înceapă cu majusculă
- ▶ `[A-Za-z]*` – poate continua cu oricâte litere mici sau mari
- ▶ `()` – demarcă două expresii în tipar; folosim `\` ca să nu fie interpretate de shell (*escaping*)
- ▶ `\1 \2` – șirurile care au fost găsite cu cele două expresii

Exemple: sed(1)

Există și jocuri scrise în sed(1)



<http://aurelio.net/projects/sedarkanoid/>

Limbaj de programare specializat pentru procesarea datelor de tip text

- ▶ Alfred **A**ho, Peter **W**einberger și Brain **K**ernighan (1970)
- ▶ funcționează pe principiul identificare tipar - aplicare funcție ca `sed(1)`
- ▶ verifică mai mult de un tipar

```
tipar1 { comenzi }  
tipar2 { comenzi }
```

- ▶ verifică linia curentă cu fiecare tipar dat după care trece la următoarea linie
- ▶ Turing Complete

Apel:

- ▶ `awk [options] [program] file ...`

Exemple: awk(1)

Afișează liniile care conțin tiparul

```
$ echo "Paul Irofti\nAlex Pop\nGeorge Stan" | awk '/  
    Pop/ {print}'  
Alex Pop
```

Găsește de câte ori apare un șir în fișier:

```
$ echo "Dori și Nemo\nNemo la dentist\nDori s-a  
    pierdut" | \  
> awk 'BEGIN {print "Finding Nemo and Dori"}  
> /[Nn]emo/ {nemo++}  
> /[Dd]ori/ {dori++}  
> END {print "Found Nemo " nemo " times and Dori "  
    dori " times!"}'  
Finding Nemo and Dori  
Found Nemo 2 times and Dori 2 times!
```

Comenzile marcate cu BEGIN și END se execută doar o dată.

Exemple: awk(1)

Procesarea câmpurilor dintr-un fișier

- ▶ \$0 – se referă la linia întreagă
- ▶ \$1,\$2,...,\$(10) – se referă la fiecare câmp în parte
- ▶ FS(*field separator*) – este *separatorul*; implicit setat ca *spațiu*

```
$ cat cont.txt
```

```
OP      SUM
```

```
IN      10
```

```
OUT     5
```

```
IN      20
```

```
IN      3
```

```
OUT     25
```

```
$ awk '
```

```
> BEGIN {print "Fonduri disponibile"; fonduri = 0}
```

```
> /IN/ { fonduri += $2 }
```

```
> /OUT/ {fonduri -= $2 }
```

```
> END {print fonduri "RON"}' cont.txt
```

```
Fonduri disponibile
```

```
3RON
```

Variabile utile

- ▶ NF – numărul de câmpuri în linia curentă
- ▶ NR – numărul de linii citite până acum
- ▶ FILENAME – numele fișierului de intrare

Funcții utile

- ▶ toupper(), tolower() – litere mari, litere mici
- ▶ exp(), log(), sin() – funcții matematice
- ▶ length() – lungimea șirului
- ▶ int() – partea întreagă

Exemple: awk(1)

Afișează toate cuvintele:

```
$ echo "Ana are mere" | \
  awk '{for (i=1;i<=NF;i++) print $i }'
```

Ana
are
mere

Listă cu utilizatori și shell folosit:

```
$ awk 'BEGIN { FS = ":" }
> {print "User " $1 " uses " $7 " shell."}'
> /etc/passwd
```

User nobody uses /sbin/nologin shell.
User paul uses /bin/ksh shell.
User souuser uses /bin/ksh shell.
User _rsync uses /sbin/nologin shell.
User alex uses /usr/local/bin/bash shell.

Exemple: awk(1)

Afișează doar numele shellului pentru un utilizator anume:

```
$ awk 'BEGIN { FS = ":" }  
> /paul/ { cmd = "basename " $7;  
> cmd | getline shell;  
> print "User " $1 " uses " shell " shell."  
> close(cmd);  
> }' /etc/passwd  
User paul uses ksh shell.
```

Execută o comandă externă și obține rezultatul cu `getline`.