Anexa 3 - Sed; awk

Contents

Filtrarea pe baza de expresii regulate

Cele mai puternice utilitare pentru filtrarea textului in lumea UNIX sunt **sed** si **awk**. Acestea permit utilizatorilor sa editeze fisierele text si sa filtreze iesirea altor comenzi folosind expresii regulate.

Similaritati

Exista multe similaritati intre **sed** si **awk**:

- au o sintaxa de invocare asemanatoare
- permit specificarea unor instructiuni care sa fie executate pentru fiecare linie a unui fisier
- folosesc expresii regulate pentru detectarea potrivirilor (pattern matching)

Sintaxa

Sintaxa folosita pentru invocarea **awk** sau **sed** este:

```
command 'script' filenames
```

Aici **command** este **awk** sau **sed**, **script** este o insiruire de comenzi intelese de **awk** sau **sed**, iar **filenames** este lista de fisiere asupra carora se actioneaza. Daca nu se specifica nici un fisier, se foloseste intrarea standard. In acest fel putem utiliza **awk** si **sed** ca si filtre pentru iesirea altor comenzi.

Mod de rulare

Cand **awk** sau **sed** ruleaza, vor realiza urmatoarele operatii:

1. se citeste o linie de la intrare
2. se realizeaza o copie a acestei linii
3. se executa **script** pe aceasta linie
4. se repeta pasul 1 pentru urmatoare linie

Structura unui script

Structura unui script este de forma

```
/pattern/   action
```

Aici **pattern** este o expresie regulata, iar **action** este actiunea pe care fie **awk**, fie **sed** va trebui sa o execute cand se intalneste sablonul. Caracterele **/** din jurul expresiei regulate sunt folosite ca delimitatori.

Cand **awk** sau **sed** executa un script, foloseste urmatoarea procedura pentru fiecare inrregistrare:

1. cauta secvential fiecare **pattern** din script pana cand se gaseste o potrivire
2. cand se gaseste o potrivire, se executa actiunea **action** asupra liniei de intrare
3. cand actiunea este completa, trece la urmatorul **pattern** si repeta pasul 1
4. cand toate sabloanele au fost epuizate, se citeste urmatoarea linie

Inaintea ultimului pas (pasul 4) **sed** afiseaza inregistrarea modificata; in **awk** va trebui specificata functia de afisare.

Expresii regulate

Elementele de baza ale unei expresii regulate sunt:

- caracterele obisnuite
- metacaracterele

Caractere obisnuite sunt litere, numere, sau caractere precum spatiu sau underscore. Metacaracterele sunt caractere care au un inteles special in cadrul unei expresii regulate. Ele sunt expandate in niste caractere obisnuite. Prin utilizarea metacaracterelor nu trebuie specificate toate combinatiile distincte de caractere pentru care vrei sa existe potrivire.

Metacaractere sunt:

- **.** - potrivire cu orice caracter mai putin newline
- ***** - potrivire cu 0 sau mai multe aparitii ale caracterului anterior
- **[chars]** - potrivire cu unul din caracterele din secventa **chars**; se poate utiliza **-** pentru utilizarea unei plaje de caractere (**a-z**, **0-9**); daca primul caracter este **^** atunci se potriveste cu orice caracter care nu este specificat in **chars**
- **^** - potrivire cu inceputul liniei
- **$** - potrivire cu sfarsitul liniei
- **** - trateaza caracterul ce urmeaza literal; este folosit pentru escaparea altor metacaractere (inclusiv \)

Exemple

Fie pattern-ul

```
/peach/
```

Daca expresia este utilizata in **sed** sau **awk** atunci orice linie care contine acest sir este selectata. Dintre liniile care pot fi selectate, putem avea

```
we have a peach tree in the backyard
i prefer peaches to plums
```

Pattern-ul

```
/a.c/
```

se potriveste cu orice linie care contine siruri precum **a+c, a-c, abc, match, a3c**, in vreme ce pattern-ul

```
/a*c/
```

se potriveste cu sirurile de mai sus si, in plus, cu **ace, yacc, arctic**. De asemea se potriveste si cu linia

```
close the window
```

Se observa ca nu exista litera **a** in propozitie; aceasta din cauza ca ***** se potriveste cu 0 sau mai multe aparitii ale lui **a**.

Se observa ca expresiile regulate folosite in cadrul **sed** si **awk** sunt similare cu cele folosite de utilitarul **lex** (**flex**).

sed

sed inseamna **stream editor**. Citeste fiecare linie de la intrare si realizeaza un set de actiuni. Sintaxa de baza pentru **sed** este:

```
sed 'script' files
```

script este o secventa de comenzi de forma

```
/pattern/ action
```

pattern este o expresie regulata iar **action** poate avea una din formele de mai jos:

- **p** - afiseaza linia
- **d** - sterge linia
- **s/pattern1/pattern2/** - substituie aparitia sablonului **pattern1** cu **pattern2**

Vom exemplifica afisarea liniilor. Avem un fisier **fruit_prices.txt**. Dorim sa afisam informatii numai despre acele fructe care au pretul sub 1. Comenzile de executat sunt urmatoarele:

```
razvan@ragnarok:~/cfiles/solab/labs/lab8$ sed '/0\.[0-9][0-9]$/p' fruit_prices.txt
fruit           price
banana          0.89
banana          0.89
peach           0.79
peach           0.79
kiwi            1.50
pineapple       1.29
apple           0.99
apple           0.99
mango           2.20
razvan@ragnarok:~/cfiles/solab/labs/lab8$ sed -n '/0\.[0-9][0-9]$/p' fruit_prices.txt
banana          0.89
peach           0.79
apple           0.99
```

Se observa necesitatea folosirii optiunii **-n**. In mod obisnuit, **sed** afiseaza toate liniile de la intrare la iesire.

Daca dorim sa eliminam intrarea despre un fruct vom folosi comanda:

```
razvan@ragnarok:~/cfiles/solab/labs/lab8$ sed '/^[Mm]ango/d' fruit_prices.txt
fruit           price
banana          0.89
peach           0.79
kiwi            1.50
pineapple       1.29
apple           0.99
```

Se observa ca in cazul folosirii actiunii **d** nu mai este nevoie sa precizam optiunea **-n**.

Pentru substitutie de expresii regulate, sintaxa este

```
/pattern/s/pattern1/pattern2/
```

In acest caz, **pattern1** este inlocuit cu **pattern2** pentru toate liniile care se potrivesc cu **pattern**. Foarte frecvent **pattern** este omis.

Sa consideram urmatorul exemplu in care dorim sa inlocuim sirul **eqal** cu **equal** din cauza unor greseli de editare. Setul de comenzi de rulat este:

```
razvan@ragnarok:~/cfiles/solab/labs/lab8$ cat nash.txt
things that are eqal to the same thing are eqal to each other
razvan@ragnarok:~/cfiles/solab/labs/lab8$ sed 's/eqal/equal/' nash.txt
things that are equal to the same thing are eqal to each other
razvan@ragnarok:~/cfiles/solab/labs/lab8$ sed 's/eqal/equal/g' nash.txt
things that are equal to the same thing are equal to each other
```

Se observa ca in prima faza se producea inlocuirea numai a primei aparitii a lui **pattern1**. Pentru a se realiza inlocuirea tuturor aparitiilor s-a folosit optiune **/g** (global).

De multe ori va trebui sa folosim comenzi **sed** multiple (de forma **/patttern/ action**). Pentru aceasta folosim optiunea **-e** in forma

```
sed –e 'command1' –e 'command2' ... –e 'commandN' files
```

Din cauza ca **sed** citeste de la intrarea standard in lipsa unui fisier, poate fi folosit ca si filtru.

awk

awk este un limbaj de programare care permite cautarea in fisiere si modificarea inregistrarilor din aceste fisiere prin intermediul sabloanelor. Numele **awk** vine de la numele creatorilor sai: Alfred Aho, Peter Weinberger si Brian Kernighan.

Ca si la **sed**, sintaxa de baza este de forma:

```
awk 'script' files
```

script este o succesiune de comenzi de forma:

```
/pattern/ { actions }
```

pattern este o expresie regulata, iar **actions** reprezinta una sau mai multe comenzi care sunt acoperite in acest capitol. Daca se omite **pattern**, se aplica actiunile asupra fiecarei linii.

Spre exemplu, o implementare a lui cat folosind **awk** ar fi:

```
$ awk '{ print; }' file
```

Editarea campurilor

Cand se citeste o linie, **awk** plaseaza campurile care le-a parsat in variabila **1** pentru primul camp, variabila **2** pentru al doilea camp si asa mai departe. Pentru a accesa un camp, vom folosi operatorul **$**. Astfel primul camp este **$1**.

Un exemplu de utilizare este:

```
razvan@ragnarok:~/cfiles/solab/labs/lab8$ awk '{ print $1 $3 ; }' fruit_prices.txt
fruitquantity
banana100
peach65
kiwi22
pineapple35
apple78
mango34
razvan@ragnarok:~/cfiles/solab/labs/lab8$ awk '{ print $1 , $3 ; }' fruit_prices.txt
fruit quantity
banana 100
```

```
peach 65
kiwi 22
pineapple 35
apple 78
mango 34
```

Putem formata iesirea folosind comanda **printf**:

```
$ awk '{ printf "%-15s %s\n", $1 , $3 ; }' fruit_prices.txt
fruit           quantity
banana          100
peach           65
kiwi            22
pineapple       35
apple           78
mango           34
```

Actiuni specifice unor pattern-uri

Dorim sa indicam acele fructe care costa mai mult de 1 prin asezarea unei stelute la sfarsitul liniei. Avem urmatorul script:

```
razvan@ragnarok:~/cfiles/solab/labs/lab8$ cat highlight.sh
#!/bin/sh

awk '
        / *[1-9][0-9]*\.[0-9][0-9] */   { print $1, $2, $3, "*"; }
        / *0\.[0-9][0-9] */             { print ; }
' fruit_prices.txt
razvan@ragnarok:~/cfiles/solab/labs/lab8$ ./highlight.sh
banana          0.89            100
peach           0.79            65
kiwi 1.50 22 *
pineapple 1.29 35 *
apple           0.99            78
mango 2.20 34 *
```

Se observa ca avem o afisare necorespunzatoare. Pentru a evita aceasta situatie vom folosi variabila 0 care este folosita pentru a stoca intreaga linie asa cum fusese ea citita. Scriptul va arata in felul urmator:

```
razvan@ragnarok:~/cfiles/solab/labs/lab8$ cat highlight.sh
#!/bin/sh

awk '
        / *[1-9][0-9]*\.[0-9][0-9] */   { print $0, "*"; }
        / *0\.[0-9][0-9] */             { print ; }
' fruit_prices.txt
razvan@ragnarok:~/cfiles/solab/labs/lab8$ ./highlight.sh
banana          0.89            100
peach           0.79            65
kiwi            1.50            22 *
pineapple       1.29            35 *
apple           0.99            78
mango           2.20            34 *
```

Filtrarea intrarii standard

In lipsa unui fisier, **awk** foloseste intrarea standard si poate fi folosit ca si filtru. Daca dorim sa afisam numai numele si dimensiunea unui fisier vom folosi urmatoarea comanda:

```
$ /bin/ls -l | awk '$1 !~ /total/ {printf "%-32s %s\n", $9, $5;}'
all_lab8.tgz                    81418
comenzi.txt                     7705
fruit_prices.txt                123
highlight.sh                    120
lab8                            4096
lab8-code.tgz                   504
lab8-split.tgz                  13321
lab8-teme.html                  5304
lab8-teme.lyx                   3831
lab8.html                       28686
lab8.lyx                        40986
lab8.ps                         113461
lab8.tex                        25969
lab8.tgz                        8803
nash.txt                        62
passwd.awk                      129
```

Operatorul **!~** este un operator boolean si intoarce **true** daca valoarea precizata (in acest caz primul camp) nu se potriveste cu pattern-ul (in acest caz **total**).

awk - limbaj de programare

Variabile

Definirea si utilizarea variabilelor in **awk** este similara cu cea din shell. Totusi, aici se face distinctia intre variabile numerice si siruri de caractere. Astfel, secventa de instructiuni:

```
a=1
b=a+1
```

se comporta exact ca si cum ar fi scris in C: a primeste valoarea 1 si b primeste valoarea 2.

Operatorii aritmetici sunt **+, -, *, /, %, ^**. La fel ca si in C, este permisa folosirea operatorului compus de forma **+=, -=, *=,** etc.

Sabloanele speciale: BEGIN si END

Daca dorim sa realizam anumite actiuni numai la inceputul sau la sfarsitul programului putem utiliza pattern-urile speciale BEGIN si END. In acest caz, forma generala pentru o comanda **awk** devine:

```
awk '
     BEGIN { actions }
     /pattern/ { actions }
     /pattern/ { actions }
     END { actions }
```

```
' files
```

Cand se precizeaza BEGIN, actiunile asociate acestuia vor fi executate inaintea citirii oricarei linii. Cand se precizeaza END, actiunile asociate vor fi executate inainte de iesire.

Variabile built-in

awk predefineste un set de variabile care pot fi folosite de utilizator. Acestea sunt:

- FILENAME - numele fisierului de intrare curent
- NR - numarul liniei curente
- NF - numarul de campuri din linia curenta
- OFS - output field separator (implicit spatiul)
- FS - input field separator (implicit spatiu si TAB)
- ORS - output record separator (implicit newline)
- RS - input record separator (implicit newline)

Pentru a stabili un nou separator, putem utiliza optiunea **-F** sau putem specifica folosind o actiune la BEGIN. Cele doua exemple sunt prezentate mai jos:

```
$ awk 'BEGIN { FS=: ; } { print $1, $6 ; }' /etc/passwd
$ awk -F':'
```

Controlul fluxului

Exista trei forme principale de controlul fluxului **if, for, while**.

Sintaxa de baza pentru **if**:

```
if (expression) {
     action1
else {
     action2
}
```

Pentru **while** este:

```
while (expression) {
     actions
}
```

Pentru **for**:

```
for (initialize_counter; test_counter; increment_counter) {
      action
}
```

Sintaxa pentru toate aceste comenzi este identica cu cea din C si permite o foarte mare flexibilitate utilizatorului.

De multe ori este util sa realizam un fisier separat pentru utilizarea unor comenzi awk. Pentru aceasta in loc de **script** in sintaxa de baza pentru awk vom folosi acel nume de fisier. Spre exemplu in loc de:

```
razvan@ragnarok:~/cfiles/solab/labs/lab8$ awk -F':' '
BEGIN { num = 0; }
/home/ { num += 1; }
END { print "Numarul total de utilizatori care au directorul de baza in /home este:", num; }
' /etc/passwd
Numarul total de utilizatori care au directorul de baza in /home este: 9
```

putem utiliza

```
razvan@ragnarok:~/cfiles/solab/labs/lab8$ cat passwd.awk
BEGIN   { num = 0; }
/home/  { num += 1; }
END     { print "Numarul de utilizatori care au directorul de baza in /home este ", num; }
razvan@ragnarok:~/cfiles/solab/labs/lab8$ awk -F':' -f passwd.awk /etc/passwd
Numarul de utilizatori care au directorul de baza in /home este  9
```

Exerciții

Pentru acest laborator folosiți resursele laboratorului. Dezarhivați fişierul resursă şi intrați în directorul `lab12-util/`

Punctare

- Fiecare exercițiu valorează 0.5 puncte, indiferent de dificultate
- Fiecare BONUS se punctează cu 0.5 puncte
- Pentru a facilita corectarea exerciţiilor, asistentul va puncta un student la fiecare 5 exerciții rezolvate (să nu piardă prea mult spațiu pe hârtie :-D).
- Din acelaşi motiv de facilitare a corectării, recomandăm rezolvarea exerciţiilor în ordine; nu există motive să săriți peste exerciții decât dacă nu găsiți soluția într-un interval acceptabil.

sed

Intrați în subdirectorul sed/

ex1: utilizare sed - acomodare

Rulați următoarele comenzi. Ce efect are fiecare? Încercați să intuiți efectul comenzii înainte de rularea acesteia.

```
sed "" lab11-enunturi.txt
sed "p" lab11-enunturi.txt
sed "d" lab11-enunturi.txt
sed -n "p" lab11-enunturi.txt
sed "p; p" lab11-enunturi.txt
sed -e "p" -e "p" lab11-enunturi.txt
sed "p" lab11-enunturi.txt | sed "p"
```

De ce comanda

```
sed p lab11-enunturi.txt
```

rulează cu succes, dar comanda

```
sed p; p lab11-enunturi.txt
```

nu? (nu folosim ghilimele sau apostroafe)

ex2: comenzi sed - acomodare

Rulați următoarele comezi. Ce efect are fiecare?

```
sed "/^$/d" lab11-enunturi.txt
sed -n "/shell/p" lab11-enunturi.txt
sed -n "s/shell/Bash/g" lab11-enunturi.txt
```

ex3: scripturi sed - acomodare

Rulați următoarele comenzi. Ce efect are fiecare?

```
sed -n -e "s/[sS]hell/Bash/g" -e "/Bash/p" lab11-enunturi.txt
sed -n '<ENTER>
    s/[sS]hell/Bash/g<ENTER>
    /Bash/p' lab11-enunturi.txt
sed -nf ex03_script.sed lab11-enunturi.txt
chmod +x ex03_script.sed && ./ex03_script.sed lab11-enunturi.txt
```

ex4: folosire de mai multe comenzi sed în acelaşi script

Utilizați o comandă sed mai simplă pentru obținerea aceluiaşi efect ca la exercițiul anterior.

ex5: emulare comenzi folosind sed

- Ce comandă emulează următoarea comandă sed?

```
sed -n "$=" lab11-enunturi.txt
```

- Emulați comanda cat
- Emulați comanda grep
- Emulați comanda grep -v

ex6: atenție la ordinea substituțiilor

- Folosiți fişierul ex06_test.txt pentru a substitui aparițiile lui a cu b şi aparițiile lui b cu c.
- Rezultatul aşteptat se află în fişierul ex06_result.txt

ex7: fiecare comandă sed operează asupra întregului pattern-space

- Fişierul ex07_test.txt conține un număr de cifre binare.
- Transformați conținutul fişierului într-o codificare Manchester (0 -> 10, 1 -> 01).
- Rezultatul aşteptat se află în fişierul ex07_result.txt

ex8+: folosire comandă s pentru eliminare, separatori de cuvinte (\< ... \>)

- Fişierul ex08_test.txt conține un set de cuvinte. Eliminați _cuvintele_ "asa" şi "alfa".
- Rezultatul aşteptat se află în fişierul ex08_result.txt

ex9: folosire comandă s şi expresii (\1, \2, etc.)

- Fişierul ex09_test.txt conține o listă de nume în forma "Nume, Prenume".
- Schimbați formatul în "Prenume Nume".
- Rezultatul aşteptat se află în fişierul ex09_result.txt

ex10: folosire comandă s şi expresii

- În cadrul fişierului mon_test_util.h substituiți apelurile şi declarațiile de funcții ce încep cu mon_... în monitor_...
- Se consideră apel/declarație de funcție dacă după numele ei apare o paranteză rotundă deschisă.

ex11: folosire spații de acțiune şi hold-space (h, G)

- Afişați doar prima şi ultima linie din fişierul lab11-enunturi.txt
- Afişați doar ultima şi prima linie (în această ordine)

ex12: folosire comandă N

Folosiți ca intrare fişierul ex12_test.txt şi:

- afişați numai liniile impare
- afişați numai liniile pare

BONUS:

- afişați liniile în ordinea 213465... liniile fişierului

ex13: emulare comenzi

Folosiți ca intrare fişierul ex12_test.txt şi:

- Emulați comanda cat -n: afişați numărul liniei urmat de conținutul liniei.
- Veți avea nevoie de două comenzi sed în pipeline.
- Comanda = este folosită pentru a afişa numărul liniei.

ex14: emulare comenzi

Folosiți ca intrare fişierul ex12_test.txt şi:

- afişați primele 10 linii dintr-un fişier
- afişați toate liniile mai puțin primele 10
- afişați fişierul în ordine inversă

ex15: folosire comenzi i şi a

Folosiți fişierul de test mon_test_util.h.

- adăugați mesajul "/* simple file */" la începutul fişierului
- adăugați mesajul "/* eof */" la sfârşitul fişierului

Atenție: caracterele / (slash) şi * (star) sunt speciale.

ex16:

Stergeți comentariile C/C++ dintr-un fişier

ex17+: folosire hold-space şi comenzi asociate (h, G)

- Realizați un script sed care eliminele liniile goale duplicate (din mai multe linii goale rămâne una singură).
- Folosiți fişierul ex17_test.txt pe post de fişier de test.
- Rezultatul aşteptat se găseşte în fişierul ex17_result.txt

BONUS:

- Extindeți exercițiul la linii albe
- O linie albă conține doar caractere albe (spațiu şi TAB)
- Se tolerează o primă linie albă la începutul fişierului de ieşire

ex18: folosire cumul de comenzi s şi expresii regulate

Folosiți fişierul ex18_test.txt şi aplicați următoarele reguli:

- eliminați liniile care încep cu unul din caracterele []{}|;:
- eliminați liniile care încep cu 4 caractere - (minus)
- eliminați tag-urile HTML
- înlocuiți expresiile \verb|mesaj cu *mesaj* (mesaj este prefixat şi sufixat de două caractere apostrof)
- eliminați caracterele carriage return (\r)
- Hint: dacă un set conține caracterele [sau] atunci] trebuie să apară primul iar

awk

Intrați în subdirectorul awk/

ex19: acomodare awk; asemănare awk/cut

Rulați comenzile de mai jos. Ce efect au?

```
cut -d ':' -f 1 /etc/passwd
awk -F ':' '{print $1}' /etc/passwd
awk 'BEGIN { FS=":"} {print $1}' /etc/passwd
awk -f ex19_script.awk /etc/passwd
chmod +x ex19_script.awk && ./ex19_script.awk
```

ex20: acomodare awk: awk mai util decât cut/tr

Rulați următoarele comenzi. Ce efect au?

```
cut -d ':' -f 1,4 /etc/passwd | tr ':' '\t'
awk -F ":" '{ printf "%-12s %6s\n", $1, $4}' /etc/passwd
awk -f ex20_script.awk /etc/passwd
```

ex21: acomodare awk: awk mai simplu/util/corect decât cut/grep

Ce efect au următarele comenzi?

```
grep '/home' /etc/passwd
awk '/\/home/ { print $0 }' /etc/passwd
grep '/home' /etc/passwd | cut -d ':' -f 1
awk -F ':' '/\/home/ { print $1}' /etc/passwd
awk -F ':' '$6 ~ /^\/home/ { print $1}' /etc/passwd
```

ex22: awk echivalent cu cut

```
afişaţi, pentru fiecare intrare din directorul curent, numele, data ultimului acces şi dimensiune
folosiţi cut şi apoi awk
puteţi afişa cele 3 componente în ordinea de mai sus folosind cut?
```

awk echivalent cu grep/cut

ex23:

- Afişaţi_doar_ adresa hardware a interfeţei eth0.
- Folosiţi grep|cut şi apoi awk

ex24:

- Afişaţi numele subdirectoarelor directorului curent (încep cu d la "ls -l")
- Afişaţi fişierele din directorul curent care au drept de scriere de others.
- Folosiţi grep/cut şi apoi awk

BONUS:

- Folosiţi find

ex25:

- Instalaţi apache2
- Din fişierul de configurare Apache2 care este valoarea directivei PidFile? Dar a directivei User?
- Folosiţi grep/cut şi apoi awk
- Fişierul de configurare Apache2 este /etc/apache2/apache2.conf

ex26:

- Dezinstalaţi pachetele instalate care conţin "apache2"
- Folosiţi dpkg cu opţiunea -l pentru a afla pachetele care conţin şirul "apache2"
- Folosiţi awk pentru a afla numele pachetelor instalate (cele care încep cu "ii")

ex27:

- Afişaţi partiţiile montate în sistemul local de fişiere şi tipul lor
- Folosiţi /etc/fstab

BONUS:

- Afişaţi partiţia ocupată de sistemul de fişiere rădăcină (/) şi tipul ei

ex28:

- Afişaţi doar numele modulelor din sistem (lsmod).
- Folosiţi cut, tail şi apoi awk.
- Hint: "tail -n +count" afişează toate liniile mai puţin primele count (de ex: tail -n +3 afişează toate liniile mai puţin primele 3)

Folosire if

ex29:

- Afişaţi modulele de care nu depinde nici un alt modul.
- Hint: numărul de înregistrări ale liniei trebuie să fie 3

ex30:

- Afişaţi lista de module de care depinde modul
- Hint: un modul m1 depinde de un altul m2 dacă m1 apare în lista Used by a lui m2

ex31:

- Afişaţi utilizatorii din sistem care
  - ♦ se pot autentifica (au x în dreptul parolei)
  - ♦ numele de utilizator conţine subşirul "an"
  - ♦ uid este mai mare sau egal decât 1000

Combinare awk cu alte comenzi shell

ex32:

- Trimiteţi un e-mail utilizatorilor din sistem al căror director home ocupă mai mult spaţiu de 10 MB.
- Folosiţi comanda mail

ex33:

- Afişaţi pentru fiecare utilizator numărul de adrese IP de la care s-a conectat.
- Consideraţi ultimele 100 de conectări din sistem (folosiţi last)

ex34: folosire getline

- Afişaţi tipurile de sisteme de operare care pot fi încărcate la pornirea sistemului
- Afişaţi titlul sistemului de operare (title) şi partiţia unde se emontează /boot (root)

ex35: folosire expresii regulate complexe; folosire split

- Afişaţi toate ţintele (target) dintr-un fişier Makefile

ex36: script complex awk: vectori, if, BEGIN, END

- Fişierul ex36_test.txt conţine o listă de studenţi în formatul Nume, Prenume, Grupa, Medie
- Analizând acest fişier afişaţi pentru fiecare grupă media şi cel mai bun student (şi media acestuia)