# 3. Expresii regulate. Filtre. Comanda sed.

## 1. Expresii regulate

Expresiile regulate sunt niste siruri de caractere ce reprezinta sabloane sau tipare (*pattern* in limba engleza). Ele se construiesc pe baza unei gramatici, la fel ca si un limbaj de programare. Aceste sabloane sunt folosite pentru "recunoasterea" si manipularea unor siruri de caractere.

Analog cu expresiile aritmetice, o expresie regulata este construita prin combinarea unor expresii mai mici cu ajutorul unor operatori.
Sa incepem cu un exemplu simplu:

```
a.z
```

Aceasta expresie regulata se potriveste cu orice sir de caractere ce contine literele 'a' si 'z' intre care se gaseste orice caracter -- dar unul singur (cu exceptia caracterului newline, deobicei), cum ar fi: "axz", "aaz", "barza", dar nu "abcz".

Cele mai simple expresii regulate sunt cele care "se potrivesc" cu un singur caracter: majoritatea caracterelor (toate literele si cifrele) se potrivesc cu ele insele. Alte caractere insa au semnificatie speciala , si daca dorim ca expresia regulata sa se potriveasca cu acel caracter, trebuie sa il citam (*quote* in limba engleza). Aceasta se poate realiza prin plasarea unui *backslash* ( '\' ) in fata caracterului respectiv. Expresiile regulate mai complexe se vor forma fie prin concatenare (scriere una dupa alta) fie cu ajutorul operatorilor ce vor fi descrisi mai jos.

Prin concatenarea a doua expresii regulate rezulta o expresie regulata ce se va potrivi cu siruri de caractere ce contin doua subsiruri adiacente ce se vor potrivi cu prima respectiv a doua expresie regulata.

O alta constructie care potriveste un singur caracter este o lista de caractere inchise intre paranteze drepte. Aceasta expresie se va potrivi cu oricare din caracterele din lista. Astfel, expresia regulata

```
compl[ei]ment
```

se va potrivi cu oricare din sirurile "complement", "compliment" sau "multumesc pentru complimentul dumneavoastra". Daca o constructie cu paranteze drepte incepe cu un ^ , atunci ea se va potrivi cu orice caracter ce **nu** este intre paranteze:

```
3[^6890]
```

reprezinta o expresie regulata ce se potriveste cu orice sir ce contine cifra 3 si nu contine pe pozitia urmatoare una din cifrele 6, 8, 9 sau 0 (atentie: daca in sirul caruia i se aplica expresia regulata nu contine dupa 3 nici un alt caracter, expresia nu se va potrivi! ). De asemenea se pot specifica intervale intregi ( considerind ordinea ASCII a caracterelor) cu ajutorul cratimei:

```
[A-Za-z]
```

reprezinta orice litera, mica sau mare.

Caracterele care nu se potrivesc cu ele insele si de care aminteam mai sus sunt metacaractere si operatori. O parte dintre ele, printre cele mai des utilizate si implementate in diversele variante de

expresii regulate le vom descrie mai jos, alaturi de alte constructii.

Punctul (**.**) se potriveste cu orice caracter, unul singur ( mai putin caracterul *newline* , de obicei ).

Expresiile regulate formate cu caracterele ^ si **$** sunt putin mai speciale, ele nu se potrivesc cu un sir de caractere ci cu sirul vid de la inceputul si respectiv sfirsitul unui rind (sir). De exemplu

```
^turing$
```

se va potrivi doar cu sirul "turing" (nu si cu "featuring" sau "turing ").

Sunt definiti o serie de operatori pentru a specifica repetitiile. Ei se aplica in dreapta unei expresii regulate , facand-o sa se potriveasca repetitiv:

| **Operator** | **Modificare adusa** |
|---|---|
| ***** | Potriveste de 0 sau mai multe ori |
| + | Potriveste de 1 sau mai multe ori |
| **?** | Potriveste de 0 sau 1 ori |

Parantezele rotunde se folosesc pentru a grupa expresiile regulate. Astfel, daca scriem

```
ab*
```

aceasta semnifica un 'a' urmat de oricate 'b'-uri (inclusiv nici unul) ; dar daca scriem

```
(ab)*
```

operatorul * se aplica grupului, semnificatia fiind 0 sau mai multe repetitii ale sirului de caractere "ab".

Un alt operator util este |, operatorul de alternare. Rezultatul lui este ca se va potrivi fie expresia regulata din stanga, fie cea din dreapta:

```
ion (pozitiv)|(negativ)
```

se va potrivi fie cu "ion pozitiv" fie cu "ion negativ".

Pentru mai multe detalii, consultati paginile de manual **grep(1)** si **perlre(1)**. Ultima prezinta expresiile regulate implementate in limbajul perl, si nu vor fi intotdeauna compatibile cu comenzi precum grep si sed. In schimb gasiti acolo o descriere mai amanuntita. Expresiile regulate pot fi folosite si din limbaje de programare precum C, vedeti **regex(3)** si **regex(7)**.

## 2. Filtre

Filtrele sunt comenzi care citesc rand cu rand fisierul standard de intrare si afiseaza la iesirea standard parte sau randurile citite, modificate sau nu, functie de semantica lor. Operatia continua pina la intalnirea marcajului de sfarsit de fisier. Aceste comenzi se inlanuie des cu ajutorul pipe-urilor, pentru a le conjuga efectul. In sistemele UNIX exista o serie de comenzi care se comporta ca filtre, multe dintre ele facand parte din standardul POSIX. Majoritatea filtrelor pot insa citi date si din fisiere specificate in linia de comanda. In continuare vom da o scurta descriere pentru cateva filtre mai des folosite:

- ***cat***
  scrie fiecare linie de la intrarea standard (sau din fisiere ale caror nume sunt date ca argumente) la iesirea standard, fara modificari.
- ***head***

scrie primele maxim 10 linii (10 este implicit; se poate specifica numarul de linii cu argumentul -n, sau se poate preciza un numar de octeti cu argumentul -c) de la intrare catre iesirea standard.

- ***tail***
  similara cu comanda ***head***, scrie ultimele 10 linii.
  **Observatie:** Aceasta comanda este obligata sa citeasca intreg fisierul inainte de a scrie ceva la iesire, astfel fiind un impediment in pipeline-izare. Se recomanda sa fie folosita ultima, daca este posibil, intr-un lant de comenzi. De ce?
- ***sort***
  afieaza liniile citite in mod ordonat lexicografic, implicit ordinea fiind crescatoare. Comanda are o serie de argumente utile. Cititi pagina de manual **sort(1)**!
- ***uniq***
  elimina liniile succesive identice dintre cele de la intrarea standard. Atentie: doua linii identice, dar care nu sunt citite una dupa alta nu vor fi depistate!
- ***cut***
  tipareste portiuni din liniile citite. Aceste sectiuni pot fi intervale de octeti, caractere sau campuri, functie de unul din argumentele -b, -c respectiv -f. In ultimul caz se vor tipari acele campuri delimitate de un caracter (implicit TAB, se poate preciza cu argumentul -d) care sunt specificate argumentul -f in forma: *lista[,lista]...*. O *lista* poate fi un simplu numar reprezentand campul dorit, sau poate fi de forma N-M, unde N si M sunt numere reprezentand primul respectiv ultimul camp ce trebuie afisat. Sau N sau M poate lipsi, in locul lor subintelegandu-se primul respectiv ultimul camp din linie. Aceeasi notatie folosita cu -b sau -c seminifica intervalul de octeti sau caractere ce se vor afisa. Exemplu:

  ```
  ls -l | cut -f 1 -d ' '
  ```

  va tipari doar lista de permisiuni a fisierelor din directorul curent.
- ***tr***
  translateaza sau sterge caractere. ***tr*** implicit translateaza, caz in care trebuie date ca argumente doua siruri de caractere reprezentand doua seturi. Caracterele din primul set vor fi translatate in caracterele din al doilea. Daca numarul de caractere din seturile nu este acelasi, caracterele excedentare dintr-al doilea se ignora daca acesta e mai lung, sau se repeta ultimul caracter din al doilea set (daca acesta e mai scurt) pana la lungimea primului set. In cazul argumentului *-d* se da un singur set de caractere, care vor fi eliminate la scrierea la iesirea standard. Argumentul *-s* (squeeze) realizeaza "contractia" caracterelor din setul dat ca parametru: in cazul in care la intrare filtrul citeste doua sau mai multe caractere identice, din set, va fi tiparit la iesire doar unul singur. Exemplu:

  ```
  ps -x | tr -s ' ' | cut -f 2,6 -d ' '
  ```

  Comanda *ps -x* afiseaza lista proceselor utilizatorului care o invoca. Cu ajutorul lui *tr -s ' '* se sterg spatiile dintre coloanele afisate, iar *cut -f 2,6 -d ' '* face sa apara la iesire doar coloana corespunzatoare identificatorilor de proces (coloana 2) si cea a numelui comenzii corespunzatoare procesului (coloana 6).
- ***grep***
  este un filtru mai complex. Are cel putin un parametru, care va fi interpretat ca expresie

  regulata. Liniile citite de la intrarea standard (sau din fisierele date ca parametri) care se potrivesc cu expresia regulata data vor fi afisate. Exista o serie de optiuni care modifica modul de afisare. O parte le gasiti in lista de mai jos:
  - *-i* face comanda insensibila la diferentele dintre literele mari si cele mici
  - *-v* tipareste liniile cu care tiparul dat *nu* se potriveste
  - *-n* tipareste numarul liniei urmat de caracterul **:** urmat de linia in sine

- *-E* forteaza folosirea expresiilor regulate extinse (in mod normal se folosesc niste expresii regulate cu sintaxa foarte simpla); exista comanda **egrep** care e echivalenta cu **grep -E**
- *-x* tipareste doar liniile pentru care cu tiparul dat se potriveste cu intreaga linie, nu cu un subsir din aceasta
- *-c* tipareste doar numarul de linii ce s-ar fi afisat in mod normal

Exemplu:

```
ps -x | grep lab9
```

Comenzile afiseaza liniile generate de **ps** corespunzatoare proceselor ce contin in numele lor (sau in parametri) sirul "lab9".

- ***tee***
  scrie la iesirea standard ceea ce citeste de la intrarea standard, in plus scrie si in fisierele ale caror nume sunt date ca parametri.
- ***wc***
  afiseaza numarul de caractere (parametrul *-m*), de octeti (*-c*), de linii (*-l*) sau se cuvinte (*-w*), sau lungimea celei mai lungi linii intalnite (parametrul *-L*). De exemplu:

  ```
  cat /etc/passwd | wc -l
  ```

  va afisa numarul de intrari (linii) in fisierul de parole.
- ***xargs***
  citeste argumente de la intrarea standard, separate prin spatii sau caractere *newline* si executa comanda data ca parametru (sau **echo** implicit) cu o lista de argumente specificate ca parametri urmate de argumentele citite de la intrare. Liniile goale vor fi ignorate. Parametrul *-isir* va modifica comporarea lui xargs astfel incat comanda data se va executa cu argumentele precizate ca parametri in care se inlocuieste sirul *sir* cu argumentele citite de la intrarea standard. Exemplu:

  ```
  echo "1 2 3" | xargs mkdir -v
  ```

  va lansa de trei ori comanda mkdir, cu parametrii -v si cate una din cifrele 1, 2 si 3.

## 3. Comanda sed

Comanda **sed** deriva din editorul **ed** care este orientat pe linie (spre deosebire de chiar cele mai simple editoare actuale care sunt orientate pe ecran). Spre deosebire de acesta din urma, sed nu este interactiv ci aplica tuturor sau unui grup de linii din fisierul prelucrat (care poate fi intrarea standard daca nu se specifica un nume de fisier ca parametru) o anumita comanda. Sintaxa (simplificata) este:

```
sed [-n] [-e script] [-f script-file] [script-if-no-other-script] [file...]
```

Optiunile *-e* (dupa care urmeaza o comanda) si *-f* (dupa care urmeaza un nume de fisier care contine cate o comanda pe fiecare rand) adauga comenzi in lista celor ce vor fi aplicate asupra

liniilor de text. Daca nici -e si nici -f nu sunt gasite, primul parametru care nu este optiune (nu incepe cu cratima) este considerat comanda. Daca dupa ce s-au prelucrat parametrii asa cum am descris mai raman si alti parametri in linia de comanda, acestia se vor considera ca nume de fisiere care trebuie procesate (in care caz sed nu se mai comporta ca filtru).

Sintaxa comenzilor **sed** este destul de complexa. Descrierea completa o gasiti in pagina texinfo a comenzii sed (in pagina de manual descrierea nu este foarte clara) pe care o puteti citi cu comanda **info sed**. In general o comanda **sed** este compusa dintr-o adresa sau interval de adrese de linii

urmat de un caracter ce reprezinta o actiune de efectuat, urmat eventual de un sir de caractere a carui semnificatie depinde de actiune.

Intervalele de adrese se specifica sub forma a doua adrese despartite prin virgula. O adresa poate fi un numar si atunci reprezinta numarul liniei, sau o expresie regulata incadrata de caractere '/', caz in care reprezinta prima linie (incepand cu linia curenta) ce corespunde (se potriveste) expresiei regulate. O adresa mai poate fi si caracterul $, care semnifica "ultima linie a ultimului fisier de intrare".

Actiunile corespunzatoare comenzilor nu se vor aplica, in cazul in care avem adrese, decat liniilor corespunzatoare adresei respectiv liniilor cuprinse in intervalul dat. Cea mai des folosita actiune (sau comanda) este cea de substitutie. Ea are forma:

```
s/regexp/replacement/flags
```

unde *regexp* este o expresie regulata, *replacement* este un sir de caractere cu care se va inlocui acea parte a liniei prelucrate care corespunde expresiei regulate, iar *flags* este o lista de zero sau mai multe caractere dintre urmatoarele:

- *g* Inlocuieste toate portiunile care se potrivesc cu *regexp* nu doar prima astfel de portiune.
- *p* Daca substitutia a fost facuta, tipareste linia corespunzator modificata (utila cu argumentul *-n*).
- *numarul N* Inlocuieste doar a N-a potrivire a expresiei regulate.

Portiunea *replacement* poate sa contina constructii de genul *\N* ce reprezinta portiunea din text care s-a potrivit cu a N-a subexpresie regulata din *regexp*, subexpresie incadrata de paranteze precedate de backslash. De asemenea se mai poate folosi caracterul & (ne-citat) ce reprezinta tot textul ce va fi inlocuit (care s-a potrivit pe tiparul *regexp* dat). Exista o serie de alte comenzi, dar descrierea lor depaseste scopul acestei lucrari. Pentru mai multe detalii consultati documentatia. Exemplu:

```
sed -e 's/^\([^:]*:[^:]*:\)10\([0-9][0-9]\)/\120\2/' /etc/passwd
```

Inlocuieste in fisierul de parole id-urile de utilizator cu valori intre 1000 si 1099 cu id-uri cu valori cu 1000 mai mari (intre 2000 si 2099). Daca se foloseste fisierul:

```
gigi:x:500:500::/home/gigi:/bin/bash
mimi:x:1000:500::/home/mimi:/bin/bash
nini:x:1011:1011::/home/nini:/bin/bash
fifi:x:500:1012::/home/fifi:/bin/bash
bibi:x:1013:500::/home/bibi:/bin/bash
```

va fi tiparit urmatorul rezultat:

```
gigi:x:500:500::/home/gigi:/bin/bash
mimi:x:2000:500::/home/mimi:/bin/bash
nini:x:2011:1011::/home/nini:/bin/bash
fifi:x:500:1012::/home/fifi:/bin/bash
bibi:x:2013:500::/home/bibi:/bin/bash
```

Pentru intelegerea expresiei regulate aveti in vedere:

- fisierul de parole */etc/passwd* are urmatorul format: fiecare rand reprezinta o intrare corespunzatoare unui utilizator si contine sapte campuri despartite prin caracterul doua puncte. Aceste campuri in ordine de la stanga la dreapta sunt: numele de login, parola criptata (sau x pentru sistemele ce ascund parolele in /etc/shadow), user id, group id, numele real al utilizatorului, calea absoluta spre directorul home al utilizatorului, shell-ul ce va fi lansat cand

utilizatorul "intra" cu login.
- ^ semnifica inceput de rand
- constructia [^:]* se potriveste cu oricate caractere diferite de **:** si a fost folosita in locul uneia de tipul **.*** pentru a se evita situatia in care punctul (.) s-ar fi potrivit si cu delimitatorul de campuri din fisierul de parole care este caracterul **:**, caz in care nu s-ar fi selectat intotdeuna primele doua campuri in prima subexpresie dintre paranteze. Incercati varianta data mai jos pe exemplul de mai sus si observati diferentele:

```
sed -e 's/^\(.*:.*:\)10\([0-9][0-9]:\)/\120\2/' /etc/passwd
```

- operatorii repetitivi sunt "lacomi" (greedy), ei "inghit" din sirul pe care il potrivesc cit mai mult posibil, astfel incit restul expresiei regulate sa se mai potriveasca totusi.

## 4. Exemple

- Verificam daca nu exista nume de login dublate in fisierul de parole:

```
#!/bin/sh
len1=`cat /etc/passwd | wc -l`
len2=`cat /etc/passwd | cut -f 1 -d ':' | sort | uniq | wc -l`
if test $len1 -eq $len2 ; then
    echo Nici o dublura de nume \!
else
    echo Dublura de nume \!
fi
```

- Un script shell care numara aparitiile vocalelor dintr-un fisier text dat ca parametru:

```
#!/bin/sh
lung1=`cat $1 | wc -m`
lung2=`cat $1 | tr -d aeiou | wc -m`
echo Numarul de vocale este `expr $lung1 - $lung2`
```

- "Ucidem" toate procesele unui utilizator (se poate realiza cu comanda skill):

```
ps -aux | grep nume login | grep -v grep | tr -s ' ' | \
    cut -f 2 -d ' ' | xargs -ixyz kill -9 xyz
```

Observati iesirea obtinuta adaugand pe rind comenzile din sirul de mai sus.
- Urmatorul script shell invoca comada cdrecord transmitindu-i ca argumente parametrii liniei de comanda, suprimand dintre acestia argumentul dev care este periculos si inlocuindu-l cu unul corespunzator sistemului pe care ruleaza:

```
#!/bin/bash
cmdline="dev=1,5,0"

for i; do
cmdline="$cmdline `echo $i | sed -e 's/.*dev.*//'`"
done
cdrecord $cmdline
```

- Paginile in format HTML corespunzatoare paginilor de manual date aici au fost generate astfel:

```
cd ~/tmp
cp /usr/share/man/man1/grep.1.gz .
cp /usr/share/man/man1/tr.1.gz .
[...]
gzip -d *.gz
ls *.1 | sed -e 's/.1$//' | xargs -i xyz \
    sh -c ``man2html xyz.1 > xyz.html''
```

```
sh -c 'man2html xyz.1 > xyz.html'
```

## 5. Exercitii

Scrieti un shell script care tipareste numarul liniilor dintr-un fisier dat ca parametru care nu sunt comentarii, cat si numarul de caractere punct si virgula (**;**) care apar in aceste linii. O linie se considera ca fiind comentariu daca incepe cu oricite spatii urmate de semnul diez (#) sau doua slash-uri (//). Daca este invocat fara parametri, script-ul va citi linii de la intrarea standard.

### Problema:

Scrieti un shell script care citeste de la intrarea standard propozitii, cite una pe linie. Scriptul va verifica un set de reguli dat mai jos si de asemenea va afisa propozitia citita, aducindu-i anumite modificari. In cazul in care linia citita nu corespunde setului de reguli enuntat, script-ul va afisa, in locul liniei modificate, mesajul "eroare: <textul citit>".

Se vor verifica urmatoarele:

- Propozitia incepe cu litera mare.
- Linia se termina cu "." (punct) si contine un singur punct.
- Caractere din cadrul propozitiei pot fi litere mici sau mari, cifre, spatii, si semnele de punctuatie:
  , -
  (virgula, cratima)

Propozitiile care corespund regulilor de mai sus vor fi afisate astfel incit caracterul "n" ce apare inaintea unui "p" sau "b" va fi transformat in "m".

In continuare dam citeva exemple de propozitii `citite de la intrarea standard` iar dedesubt *rezultatul asteptat* de la script:

```
In continuare dam citeva exenple de propozitii.
```
*In continuare dam citeva exemple de propozitii.*
```
nu stiu altii cum sint, dar eu sint cam lenes
```
*eroare: Nu stiu altii cum sint, dar eu sint cam lenes*

---

*Autor:* ***Petre Mierlutiu***

---