

1. Comenzi, expresii regulate, filtrele grep, sed, awk

Contents

1.	COMENZI, EXPRESII REGULATE, FILTRELE GREP, SED, AWK.....	1
1.1.	OS UNIX; DEOSEBIRI FORMALE UNIX - WINDOWS:	1
1.2.	COMENZI.....	2
1.3.	EXPRESII REGULATE: DEFINIRE ȘI EXEMPLE	3
1.4.	CLASIFICAREA COMENZILOR; COMENZI FILTRU	4
1.4.1.	Clasificarea comenzilor Unix.....	4
1.4.2.	Filtrul grep	5
1.4.3.	Filtrul sed (Stream Editor)	6
1.4.4.	Filtrul sort	7
1.4.5.	Filtrul uniq.....	7
1.4.6.	Filtrul cut.....	7
1.5.	AWK; PROGRAMAREA ÎN AWK	8
1.5.1.	Apelul și definirea programelor awk	8
1.5.2.	Câteva exemple simple	10
1.5.3.	Ilustrare moduri de apel: numărarea liniilor, cuvintelor și caracterelor	10
1.5.4.	Afișarea primului cuvânt din fiecare linie	11
1.5.5.	Afișarea liniilor care au un anumit ultim cuvânt.	11
1.5.6.	Să se afișeze liniile mai lungi de 5 caractere.....	12
1.5.7.	Prelucrări asupra unui fișier cu câmpuri fixe.	12
1.5.8.	Rearanjarea cuvintelor din liniile unui fișier	13
1.6.	PROBLEME PROPUSE	14

1.1. OS Unix; deosebiri formale Unix - Windows:

Istории insolite:

<https://www.levenez.com/unix/>

<https://www.levenez.com/windows/>

<https://www.levenez.com/lang/>

Cum folosim Unix?

- Un sistem numai cu **OS Linux** (Ubuntu ...) sau **macOS**
- Dual operating system (**grab** – decide dacă se intră sub Unix sau Windows).
- Folosirea unei mașini virtuale **VmWare, VirtualBox**.
- Windows (Subsystem for Linux) **assistant Linux**
- **Termux** – terminal Linux for Android

Deosebiri formale Unix – Windows:

		Unix	Windows
1	Specificare absolută fișier	/dir1/dir2/.../dirn/fișier	d:\dir1\dir2\...\dirn\fișier
2	Separator directoare PATH	dir1:dir2:...:dirn	dir1;dir2;...;dirn
3	Specificare opțiune	com -opt	com /opt
4	Separator linii în fișier text (Mac OS linie\rlinie CR)	linie\nlinie (LF = 0A)	linie\r\nlinie (CR LF = 0D 0A)
5	Parametrii linie comandă: com arg1 arg2 ... argn	\$0 \$1 ... \$9	%0 %1 ... %9
6	Valoarea unei variabile shell	\${nume}	%nume%

1.2. Comenzi

O comandă (Unix sau Windows) este de forma:

ncomandă opțiuni expresii fișiere

"**ncomandă**" este numele propriu-zis al comenzii;

"**opțiuni**" o opțiune Unix este specificată de obicei printr-o singură literă. În unele cazuri, litera este urmată de un argument șir de caractere sau număr întreg. Un grup de opțiuni este de obicei precedat de - (minus). Există și formele lungi, prefixate de --

"**expresii**" sunt șiruri de caractere, utilizate ca argumente pentru comanda respectivă. Un caz particular sunt *expresiile regulate*, (care indică machete sintactice ale unor șiruri), de care ne ocupăm în secțiunea următoare.

"**fișiere**" reprezintă unul sau mai multe fișiere specificate relativ (doar numele acestora) sau absolut (cu cale completă) sau specificări generice ('*' înlocuiește orice șir de caractere, '?' înlocuiește orice caracter, [șir] înlocuiește un caracter ce e în șir, [!șir], înlocuiește orice caracter ce nu e în șir).

Exemple:

- Forma scurtă **ls -l** , forma lungă **ls --all**
- **cut -d : -f 1,2,3 /etc/passwd** sau
- **cut -delimiter=: -fields=1,2,3 /etc/passwd**
- **gcc -Wall -g -o hello hello.c**
- **rm alf[aeiou]bet**

În mod implicit, comenzilor Unix le sunt asociate trei fișiere: *fișierul standard de intrare*, *fișierul standard de ieșire*, *fișierul standard de erori*. Uneori, dacă se dorește referirea la ele în linia de comandă, aceste fișiere pot fi specificate prin &0 &1 &2 sau 0, 1, 2.

La terminarea execuției oricărei comenzi în sistem se returnează un număr întreg, numit *cod de retur* sau *exit status*. În general, codul de retur '0' denotă faptul că execuția comenzii s-a încheiat cu succes.

Fișierul standard de intrare este asociat implicit tastaturii, iar celelalte două sunt asociate monitorului. Aceste asocieri pot fi modificate (*redirectate*, *redirecționate*) astfel:

comanda <fin datele de intrare (input-ul) pentru comandă se vor prelua din fișierul text **fin**, pregătit în prealabil.

comanda >fout sau **comanda >>fout** ieșirea standard va fi depusă în fișierul **fout**; dacă se folosește semnul ">" se va crea un fișier nou cu numele specificat în care se va scrie output-ul comenzii (în cazul în care fișierul există, conținutul acestuia este suprascris); când se utilizează ">>" output-ul comenzii este adăugat la sfârșitul fișierului **fout** dacă fișierul există deja, în caz contrar creându-se fișierul respectiv. (Se poate și **1>fout**, **1>>fout**)

2>&1 specifică faptul că pentru comandă fișierul de erori standard va fi același cu fișierul de ieșire standard. (Se poate și **2>ferrori**)

comanda1 | comanda2 ieșirea standard pentru **comanda1** se constituie automat în intrare standard pentru **comanda2** (conectare în *pipe*). Cele două comenzi se execută în paralel, fără a se crea o ieșire standard pentru **comanda1**, nici intrare standard pentru **comanda2**. Cele două comenzi așteaptă una după cealaltă livrarea / primirea de octeți prin acest pipe.

1.3. Expresii regulate: definire și exemple

Expresiile regulate sunt șabloane care indică anumite forme sintactice care trebuie să le aibă stringurile care **satisfac** aceste șabloane.

În cele ce urmează vom descrie, în cadrul cel mai general, notațiile folosite **în expresiile regulate Unix**. Există multe tipuri de expresii regulate: Python, PERL, Java C++ etc. În esență există asemănări între acestea, dar din păcate sunt și deosebiri. Mai mult, chiar expresiile regulate Unix sunt specifice unor anumite comenzi și pot să difere, ne semnificativ, între ele.

În continuare, prin **c**, **c1** și **c2** vom nota caractere, iar prin **r**, **r1** și **r2** vom nota expresii regulate deja construite. Tabelul următor descrie machetele expresiilor regulate Unix:

Expresie regular	Semnificație
caracter ASCII	Orice caracter ASCII, cu excepția caracterelor <code>.</code> <code>[</code> <code>{</code> <code>(</code> <code>)</code> <code>\</code> <code>*</code> <code>+</code> <code>?</code> <code> </code> <code>^</code> <code>\$</code> este o expresie regulată.
<code>.</code> <code>[</code> <code>{</code> <code>(</code> <code>)</code> <code>\</code> <code>*</code> <code>+</code> <code>?</code> <code> </code> <code>^</code> <code>\$</code>	Acestea sunt caractere speciale în expresie regulată.
<code>.</code>	înlocuiește orice caracter orice caracter într-o expresie regulată
<code>\c</code>	unde <code>c</code> este unul dintre caracterele speciale, își pierde statutul de caracter special și apare ca un caracter obișnuit în expresie regulată.
<code>[lista]</code>	un singur caracter, oricare din lista
<code>[c1-c2]</code>	orice caracter cuprins lexicografic între caracterele c1 și c2
<code>[^lista]</code>	negația lui <code>[lista]</code> , deci un singur caracter, care nu este în listă
<code>^</code>	următorul șablon se aplică numai la început de linie
<code>\$</code>	următorul șablon se aplică numai la sfârșit de linie
<code>\<</code>	semnifică început de cuvânt (un cuvânt este format din litere, cifre sau <code>-</code> , orice alt caracter este considerat separator)
<code>\></code>	semnifică sfârșit de cuvânt
r*	șirul vid sau concatenarea repetată a expresiei regulate r cu ea însăși ori de câte ori
r+	concatenarea repetată a expresiei regulate r cu ea însăși cel puțin o dată
r?	șirul vid sau expresia regulată r
(r)	expresia r privită ca o singură entitate
r1r2	rezultatul concatenării expresiei regulate r1 urmată de r2
r1 r2	fie expresia regulată r1 , fie expresia regulată r2
r{n,m}	repetă expresia regulată r de cel puțin n ori și de cel mult m ori
n,m	partea de text dintre liniile n și m
<code>.</code> (punct)	indică la editare linia curentă
<code>\$</code> (dolar)	indică la editare ultima linie
<code>/șir/</code>	prima dintre liniile următoare față de linia curentă care conține șir
<code>?șir?</code>	prima dintre liniile precedente față de linia curentă care conține șir

Unele comenzi pot folosi construcții escape ce definesc clase de caractere, cele din tabelul următor:

<code>\d</code>	cifră
<code>\l</code>	literă mică
<code>\s</code>	spațiu
<code>\u</code>	literă mare
<code>\w</code>	cuvânt

<code>\D</code>	Nu e cifră
<code>\L</code>	Nu e literă mică
<code>\S</code>	Nu e spațiu
<code>\U</code>	Nu e literă mare
<code>\W</code>	Nu e cuvânt

Iată câteva exemple:

`[123]` - oricare dintre cifrele 1, 2, sau 3
`[123]` - 1, 2, 3, sau spațiu
`[a-z]` - orice litera mică
`[aeiou]` - orice vocală
`[A-Z]` - orice literă mare
`[0-9]` - orice cifră
`[^0-9]` - orice care nu este cifră
`[^, . :]` - orice cu excepția virgula, punct, două puncte
`.*` - orice secvență de caractere
`[a-zA-Z02468]` - orice literă mică, mare sau cifră pară
`[\t]` - spațiu sau tab
`^[^0-9]\+$` - orice linie nevidă care nu conține cifre
`\ ([Nn] [Oo] \) \+` - orice refuz, oricât de insistent! (No no no no no ...)

O construcție de forma:

`[0-3] [0-9] [-/] [0-1] [0-9] [-/] [0-9] [0-9]`

indică scrierea unei date calendaristice. Printre formele admise amintim:

`ZZ-LL-AA`, `ZZ/LL-AA`, `ZZ-LL/AA`.

Numar real in diverse limbaje de programare:

`[-+]? ([0-9]+\.?|\.[0-9]) [0-9]* ([eE] [-+]? [0-9]+)?`

De exemplu, numarul lui Avogadro se scrie: `6.023E+23`

Sintaxa unei adrese email:

`[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,6}`

Sintaxa unui url:

`(http|https|ftp) : [\/]{2} ([a-zA-Z0-9\-\.\.]+\.[a-zA-Z]{2,4})`

1.4. Clasificarea comenzilor; comenzi filtru

1.4.1. Clasificarea comenzilor Unix

Comenzile Unix pot fi clasificate în:

- Comenzi interne incluse în sh
- Comenzi externe, fie din setul standard Unix, fie elaborate de utilizatori:
 - Fișiere rezultate din compilări ale unor limbaje de programare (majoritar C)
 - Scripturi (fișiere de comenzi).

Nu avem în intenție prezentarea comenzilor Unix, ci recomandăm consultarea manualelor acestora: `$ man [secțiune] numecomanda`

Dintre cele mai populare comenzi Unix amintim:

- Filtre Unix – intrarea este un fișier text (sau mai multe) sau în absență intrarea standard, o transformă și dau transformarea la ieșirea standard: `grep`, `sed`, `sort`, `uniq`, `cut`, `awk`
- Comenzi de lucru cu fișiere (+directoare): `ls`, `pwd`, `cat`, `find`, `locate`, `file`, `more`, `less`, `rm`, `mkdir`, `rmdir`, `cp`, `mv`, `cd`, `chmod`, `chown`, `ln`, `touch`, `du`, `cmp`, `diff`, `head`, `tail`, `split`, `wc`
- Comenzi pentru aflarea de informații despre utilizatori: `finger`, `w`, `who`, `ps`, `last`, `id`, `users`
- Comenzi pentru informații de rețea: `netstat`, `ping`, `hostname`, `host`, `ftp`, `who`
- Alte comenzi: `clear`, `date`, `mail`, `uptime`, `df`, `fg`, `bg`

Pentru aceste filtre prezentăm sumar sintaxele și un exemplu. Pentru detalii se pot consulta manualele acestor filtre.

1.4.2. Filturul `grep`

caută în unul sau mai multe fișiere (sau în intrarea standard) linii care satisfac o anumită expresie regulată. Distribuțiile Unix oferă mai multe variante: `grep`, `egrep`, `fgrep`, `rgrep`. În sintaxă fie se specifică direct expresia regulată prin `pattern`, fie acest `pattern` se depune într-un fișier:

```
grep [opțiuni] [-e pattern | -f fis_pattern] [fișier ...]
```

Printre opțiuni amintim:

- `-E` - folosirea expresiilor regulate extinse
- `-v` - afișează liniile ce NU se potrivesc cu `pattern`
- `-i` - comparații ignorecase
- `-c` - numără liniile ce verifică
- `-q` - întoarce codul de retur 0 dacă nu apar potriviri, 1 dacă apar.

De exemplu, să se tipărească liniile din fișierul `linii.c` care conțin vocale scrise cu majuscule:

```
grep -e [AEIOU] linii.c
```

Câteva exemple de căutare într-un fișier text **a.txt**: se vor afișa liniile nevide, liniile vide, liniile cu un număr par de caractere, liniile cu nume de ocean, liniile ce conțin adrese email.

```
grep -E "." a.txt
```

```
grep -E "^$" a.txt
```

```
grep -E "^(..)*$" a.txt
```

```
grep -E -i
```

```
"\<atlantic\>|\<pacific\>|\<indian\>|\<arctic\>|\<antarctic\>" -E a.txt
```

```
grep -E -i "\<[^@*\\!?!?]+@[a-z0-9_-]+(\.[a-z0-9_-]+)+\>" a.txt
```

Căutarea în fișierul **/etc/passwd**, care are structura:

```
userName:password:UID:GID:fullName:homeDirectory:shell
```

Se vor afișa liniile ce conțin stringul `dan`, ce conțin stringul `dan` la început de linie indiferent de litere mari sau mici, liniile ce au un `userName` format numai din cifre, `userName` cu cel puțin 2 sau 5 vocale, `shell` să fie `bash`:

```
grep -E "dan" /etc/passwd
grep -E -i "^dan:" /etc/passwd
grep -E "^[^0-9]+:" /etc/passwd
grep -E -i "^[^:]*[aeiou][^:]*[aeiou][^:]*:" /etc/passwd
grep -E -i "^[^:]*([aeiou][^:]*){2,}:" /etc/passwd
grep -E -i "^[^:]*([aeiou][^:]*){5,}:" /etc/passwd
grep -E -v "/bash$" /etc/passwd
```

1.4.3. Filtrul sed (Stream Editor)

Este un editor de texte neconversațional. El preia un fișier (sau mai multe, sau intrarea standard), aplică asupra lui comenzi de editare, după care rezultatul este tipărit la ieșirea standard (sau cu numele vechiului fișier dacă se pune opțiunea -n).

```
sed [optiuni] [ -e comenziEdit | -f fis_comenziEdit] [ fișier ... ]
```

De exemplu, să se elimine toate secvențele <cifră><literă><cifră> dintr-un fișier:

```
$ sed "s/[0-9][a-z,A-Z][0-9]//g" fis1
```

Câteva transformări din **.a.txt**: înlocuirea vocalelor cu spații, convertirea vocalelor mici în vocale cu litere mari, ștergerea liniilor ce conțin cel puțin 5 cifre, eliminarea perechilor de litere identice, duplicarea vocalelor:

```
sed -E "s/[aeiou]/ /gi" a.txt
sed -E "y/aeiou/AEIOU/" a.txt
sed -E "/[0-9]{5,}/d" a.txt
sed -E "s/([a-z])([a-z])/\2\1/gi" a.txt
sed -E "s/([aeiou])/\1\1/gi" a.txt
```

Doua construcții de expresii regulate mai deosebite în sed. Construcția & (sau \& depinde de context) semnifică conținutul patternului care s-a potrivit. De exemplu, comanda de editare

```
s/^[0-9][0-9]/(&)/g
```

pune în paranteză primele două cifre de la începutul fiecărei linii: 4064123456 este înlocuit cu (40) 64123456

Construcțiile \1, \2, ..., \9 semnifică prima, a doua, respectiv a 9-a potrivire din "expresie_regulară")

```
$ cat phone.txt | sed 's/\(.*\)\\\(.*-\\\)\\(.*$\\)/Area code: \1 Second: \2 Third: \3/'
```

înlocuiește fișierul inițial phone.txt:

```
(555) 555-1212
(555) 555-1213
(555) 555-1214
(666) 555-1215
(666) 555-1216
(777) 555-1217
```

cu noul conținut:

```
Area code: (555) Second: 555- Third: 1212
Area code: (555) Second: 555- Third: 1213
Area code: (555) Second: 555- Third: 1214
Area code: (666) Second: 555- Third: 1215
```

Area code: (666) Second: 555- Third: 1216

Area code: (777) Second: 555- Third: 1217

1.4.4. Filtrul sort

Preia și eventual concatenează fișierele specificate în intrare și le dă la ieșirea standard cu liniile ordonate alfabetic. În absența specificării fișierelor de intrare, se ia intrarea standard.

```
sort [optiuni] [ fisier . . . ]
```

Comanda are un mare număr de opțiuni și acestea pot diferi de la un tip de Unix la altul. Oricum, prin opțiuni se pot specifica tipurile de comparații de sortare (text, numeric, date calendaristice etc. De asemenea, se pot indica porțiunile din linii și ordinea acestora care se compară în vederea stabilirii ordinii. În fine, se poate cere eliminarea duplicatelor. În absența opțiunilor, la ieșire se furnizează liniile ordonate alfabetic.

1.4.5. Filtrul uniq

tratează liniile adiacente care sunt identice (NU face sortare!).

```
uniq [optiuni] [ fisier ]
```

Fără opțiuni, din liniile adiacente identice se dă la ieșire doar prima.

1.4.6. Filtrul cut

Preia linii de la fișierul (fișierele) de intrare sau de la intrarea standard și la ieșire, din fiecare linie, dă la ieșire numai porțiunile indicate.

Indicarea porțiunilor se face prin liste cu construcții de forma $n \ n-m \ n- \ -m$, unde n și m sunt întregi, $n < m$, indicând numărul elementului (începând cu 1), de la n la m inclusiv, de la n pana la sfârșit, de la început până la m .

Numărarea se face la nivel de octet ($-b$), de caracter ($-c$) sau de câmp ($-f$). Diferența între caracter și octet apare atunci când liniile conțin octeți cu codul între 128 și 255 (caractere din ASCII extins). Apelurile posibile sunt:

```
cut -b lista [ fisier . . . ]
cut -c lista [ fisier . . . ]
cut -f lista [-d delimitator] [ fisier . . . ]
```

Caracterul delimitator implicit este TAB, dar în locul lui se poate specifica altul, prin $-d$.

Ca exemplu, să considerăm un fișier FIS ce conține:

```
foo:bar:baz:qux:quux
one:two:three:four:five:six:seven
alpha:beta:gamma:delta:epsilon:zeta:eta:theta:iota:kappa:lambda:mu
the quick brown fox jumps over the lazy dog
```

Efectul următoarelor trei rulări sunt:

```
$ cut -d ":" -f 5- FILE
quux
five:six:seven
epsilon:zeta:eta:theta:iota:kappa:lambda:mu
the quick brown fox jumps over the lazy dog

$ cut -c 4-10 FILE
:bar:ba
:two:th
ha:beta
quick

$ cut -c 4-10,14,16- FILE
:bar:bau:quux
:two:th:our:five:six:seven
ha:betama:delta:epsilon:zeta:eta:theta:iota:kappa:lambda:mu
quick w fox jumps over the lazy dog
```

1.5. awk; programarea în awk

1.5.1. Apelul și definirea programelor awk

Acest utilitar prelucrează fișiere text, selectând acele linii din text care satisfac anumite condiții (șabloane, expresii regulate), carora li se aplica o serie de acțiuni. Numele utilitarului vine de la cei trei proiectanți și implementatori ai lui: A. Aho, P. Weinberger și B. Kernighan. În prezent există mai multe variante îmbunătățite: **gawk**, **mawk**, **nawk** etc. Noi vom trata doar varianta standard. Sintaxa comenzii este:

```
awk [ -f fisier_program | 'program' ] [ -Fc ] [ [ -v ] var=val ... ]
[fisier ... ]
```

Explicăm parametrii începând din dreapta:

Intrarea în awk este constituită din lista de fișiere ale căror nume sunt date în linia de comandă: `fișier . . .`. Aceasta lista poate lipsi, caz în care se prelucrează intrarea standard.

Rezultatul filtrării prin awk este afișat la ieșirea standard.

Opțiunea `-v` precede definirea unor variabile globale și a valorilor acestora: `var=val . . .`. Nu sunt necesare decât specificările atribuirilor `var=val`, specificarea `-v` poate lipsi. Mai mult, prezența lui `-v` în fișier de comenzi nu funcționează! Este vorba de mecanismele de tratare a opțiunilor atunci când și awk și shell le tratează.

Opțiunea `-F` specifică caracterul `c` care va fi separator de cuvinte. În absența opțiunii, separatorul implicit este orice spațiu alb (BLANK, TAB \n \r).

program poate fi scris fie direct în comanda awk, fie pregătit în prealabil în **fișier_program** și indicat prin opțiunea **-f**. Liniile din **program** sunt de forma:

```
conditie { instructiuni }
```

awk tratează pe rând câte o linie din fișierele de intrare și pentru fiecare execută `instructiuni` atunci când `conditie` ia valoarea `true`. Dacă `conditie` lipsește atunci se consideră implicit adevărată. .

Sintaxa condițiilor și a instrucțiunilor sunt similare cu cele din limbajul C. Variabilele nu trebuie să fie declarate, ele se initializează automat. Tipul lor se deduce din context. Inițial, valorile variabilelor sunt 0 pentru numere și "" (șirul vid) pentru șiruri. Operanzii pot fi expresii aritmetice, expresii relationale, constante și variabile. Pentru variabilele de tip șir operatorul de concatenare este spațiul. Există câteva funcții de lucru cu șiruri. Se pot folosi variabile de tip tablou ale căror indici pot să fie numerici (cu numerotarea începând de la 1) sau șiruri de caractere - acestea din urmă sunt **tablouri asociative**.

Expresiile sunt cele din C. Operatorii relaționali se extind asupra stringurilor. Pentru stringuri există *operatorii de potrivire* cu expresiile regulate **expr ~ /expreg/** pentru potrivire și **expr !~ /expreg/** pentru nepotrivire.

condiție - este o expresie logică construită cu operatorii din C: ||, &&, !, () .

Condiții predefinite:

- BEGIN este adevărată înainte de prima linie din primul fișier
- END este adevărată după ultima linie din ultimul fișier

Variabile predefinite:

- NF - numărul de cuvinte (câmpuri) din linia curentă, cuvintele notate \$1, \$2, . . . \$NF.
- NR - numărul de ordine al liniei curente (numărătoarea începe de la 1) ce include lungimile fișierelor deja prelucrate plus cea curentă a fișierului curent.
- FNR - numărul de ordine al liniei curente din fișierul curent; liniile cu nr. 1 sunt primele linii din fiecare fișier; număratoarea începe de la 1 la începutul fiecărui fișier
- FS - separatorul de câmpuri (spațiul alb sau opțiunea -F)
- FILENAME - numele fișierului curent care este tratat
- OFS - separator de câmpuri la ieșire (implicit este spațiu)
- ORS - separator de înregistrări la ieșire (implicit este linie nouă)
- ARGV - șirul parametrilor din linia de comandă. **specificarea program sau -f fișier_program nu se ia în considerare ca argument.**
- ARGV - numărul parametrilor din linia de comandă. **Vezi mai sus.**
- variabilele globale definite prin opțiunea -v.

Accesarea câmpurilor:

- se face cu \$1, \$2 ... \$i, \$(i+1), \$NF, iar întreaga linie se referă cu \$0
- sir1 sir2 este operația de concatenare a sirurilor; se face scriind unul după altul sirurile de concatenat

Funcții predefinite:

- length(sir) - lungime șir; length <=> length(\$0)
- substr(s,p,n) - subșirul lui s care începe la poziția p și are lungimea n
- index(s1,s2) - întoarce poziția la care s2 apare în s1 sau 0 la absență
- sprintf(format, arg1, ..) - întoarce ca rezultat șirul pe care printf l-ar tipări în C
- split(s,a,c) - unde s este șir, a este tablou și c un caracter. Împarte șirul s în câmpuri considerând ca separator caracterul c; dacă c lipsește atunci separatorul implicit este FS. Valorile împărțite sunt date ca valori elementelor tabloului a
- system(cmd) - execută comanda shell cmd și returnează codul sau de retur

Instrucțiuni

- variabilă = expresie
- instrucțiunile if, for, while ca și în C
- for (i in numetablou) instrucțiune. i ia ca valori indicii lui numetablou și se execută instrucțiune pentru fiecare valoare a lui i

- ; este separator de instrucțiuni
- } este separator de linie, continuarea unei linii se face cu caracterul \ pe ultima poziție din linie
- print listă-expresii [> nume-fiș] - afișează la ieșirea standard valoarea expresiilor separate prin OFS, iar la sfârșit de linie pune ORS. Dacă se specifica >nume-fis atunci scrierea se face în fișierul nume-fis.

1.5.2. Câteva exemple simple

În exemplele care urmează vom include program direct în linia de comandă `awk` și preluăm intrarea din `/etc/passwd`. Acțiunile ce ni le propunem sunt: afișare `userName` (și numai el), afișare `fullName` din liniile pare, afișare `homeDirectory` al cărui `userName` începe cu vocală, afișare `fullName` al cărui `UID` este par, afișează `userName` al cărui `shell` este `nologin`, afișează `fullName` al cărui `userName` este mai lung de 10 caractere.

```
awk -F: '{print $1}' /etc/passwd
awk -F: 'NR % 2 == 1{print $6}' /etc/passwd
awk -F: '/^[aeiouAEIOU]/ {print $6}' /etc/passwd
awk -F: '$3 % 2 == 0 {print $6}' /etc/passwd
awk -F: '$NF ~ /nologin$/ {print $1}' /etc/passwd
awk -F: 'length($1) > 10{print $1}' /etc/passwd
```

Câteva exemple de fișiere program:

```
NR % 2 == 0 && $4 < 20 {print $5}
```

Afișează din liniile pare full name cu `GID < 20`

```
BEGIN {sum=0}
{sum += $3}
END {print sum}
```

Însumează toate `gid`

```
BEGIN {prod=1}
$3-$4 != 0 {prod *= $3-$4}
END {print prod}
```

Face produsul digerențelor dintre `UID` și `GID`

1.5.3. Ilustrare moduri de apel: numărarea liniilor, cuvintelor și caracterelor

Pentru a ilustra modurile de apel `awk`, am ales un program de numărare linii, cuvinte și caractere din fișierul numit `deprelucrat`.

Mai întâi soluția cu programul **awk** scris într-un fișier separat. Vom pregăti în fișierul **fisp** programul:

```
{car += length($0)+1; cuv += NF;}
END {print "Fișier:" FILENAME, "Linii:" NR, "Cuvinte:" cuv, "Caractere:" car;}
```

(la `length($0)` se adaugă 1 pentru a număra terminatorul de linie). Comanda de numărare va fi:

```
awk -f fisp deprelucrat
```

Dacă fișierul **deprelucrat** are conținutul:

```
ggg ooioi jxj
jjj
```

rezultatul executiei va fi:

```
Fisier:deprelucrat Linii:2 Cuvinte:4 Caractere:20
```

Si acum, varianta cu program scris direct in linia de comanda:

```
awk '{car += length($0)+1; cuv += NF;}\
END {print "Fisier:" FILENAME, "Linii:" NR, "Cuvinte:" cuv, "Caractere:" car;}'\
deprelucrat
```

Intr-o constructie de forma:

```
comanda | awk -f fisp
```

awk va prelucra iesirea standard data de comanda.

Prelucrarea a trei fisiere se face:

```
awk -f fisp fisier1 fisier2 fisier3
```

In acest caz se va tipari numele ultimului fisier (*fisier3*), iar numarul de linii, cuvinte si caractere sunt valori cumulate din cele trei fisiere!

1.5.4. Afişarea primului cuvânt din fiecare linie

Vom lua ca fişier de intrare fişierul deprelucrat:

```
awk '{ print $1}' deprelucrat
```

Rezultatul execuţiei va fi:

```
ggg
jjj
```

1.5.5. Afişarea liniilor care au un anumit ultim cuvânt.

Să se scrie un program care să afişeze liniile din fişierul *f* pentru care ultimul cuvânt este CevaAiurea si numarul curent al fiecărei astfel de linii. Prezintam rezolvarea in trei variante: direct, cu variabila globala *ultim* si cu ARGV.

```
awk '$NF == "CevaAiurea" {print "direct:", NF, $0;}' f
awk '$NF == ultim {print "cu var:", NF, $0;}' \-v ultim=CevaAiurea f
awk '$NF == ARGV[2] {print "cu ARGV:", NF, $0;}' f CevaAiurea
# Dupa prelucrarea lui f va spune ca nu gaseste fisierul CevaAiurea.
awk '$NF == ultim {print "ca parametru in script shell:", NF, $0;}' ultim="$1" f
# CevaAiurea se va da la linia de comanda a fisierului shell
```

Daca fişierul *f* are conţinutul:

```
CevaAiurea
uguui iuuhih
hphph poi hphp CevaAiurea
gggg
```

atunci rezultatul celor patru executii va fi:

```
SIR 1 CevaAiurea
SIR 3 hphph poihiphp CevaAiurea
```

In loc de SIR apare direct: cu var: cu ARGV: ca parametru in script shell: cu mesaj de eroare (CevaAiurea no such file). Executia a patra trebuie inclusa intr-un script shell si CevaAiurea primul parametru al acestuia.

1.5.6. Să se afișeze liniile mai lungi de 5 caractere

Am ales ca intrare fișierul `nume_fis`. Afișarea acestor linii să se facă în ordinea inversă apariției lor, **pentru fiecare fișier in parte**. Mai intai prezentam varianta simpla, cand se da la intrare un singur fișier, apoi varianta generală, care trateaza mai multe fisiere de intrare:

```
# Solutia cu un singur fisier
awk 'length($0)> 5 {x[++n]=$0;}\
END {for ( ; n>=0; n--) print x[n];}' nume_fis

# Solutia cu mai multe fisiere
awk 'END {print "Fisierul:"fisier; for ( ; n>=0; n--) print x[n];}\
NR>1 && FNR==1 {print "Fisierul:"fisier; for ( ; n>=0; n--) print x[n];\
n=0;}\
FNR==1 {fisier=FILENAME;}\
length>5 {x[++n]=$0;}' nume_fis nume_fis f awk3
```

La varianta cu mai multe fisiere trebuie remarcată succesiunea celor 4 conditii pentru a "prinde" numele vechiului fisier atunci cand a aparut deja fisierul cel nou. Se vor tipări liniile mai lungi de 5 caractere din cele 4 fisiere (primele doua sunt de fapt acelasi fisier).

1.5.7. Prelucrari asupra unui fisier cu campuri fixe.

Sa presupunem ca avem fisierul text log cu linii avand fiecare dintre ele cinci campuri care inventariaza activitatile unor useri conectati la niste servere:

```
popescu www.scs.ubbcluj.ro azi 60 130
ionescu www.scs.ubbcluj.ro maine 3 20
dan linux.scs.ubbcluj.ro ieri 7 400
popescu www.scs.ubbcluj.ro azi 20 130
dan www.scs.ubbcluj.ro ieri 35 20
dan linux.scs.ubbcluj.ro alaltaieri 400 10
```

Cele 5 campuri inseamna:

User AdresaServer DataConectarii DurataConectarii SiteuriAccesate

Se cer:

- Pentru fiecare data, numarul total de useri conectati, durata totala a conectarilor.
- Pentru fiecare user numarul total de conectari, durata totala a conectarilor, totalul siteurilor accesate, cea mai lunga conectare.
- Pentru fiecare server, numarul total de conectari, durata totala a acestora, cea mai scurta conectare.

Programele celor trei cerinte sunt:

```
#awk5 cerinta a.
```

```

awk 'NF >= 4 {tuc[$3]++; dtc[$3]+=$4;}\
END {print "Solutie a:";\
print "Total useri conectati:"; for (u in tuc) print "\t", u, tuc[u];\
print "Durate totale conectari:"; for (u in dtc) print "\t", u, dtc[u];}' log

#awk5 cerinta b.
awk 'NF >= 5 {tc[$1]++; dtc[$1]+=$4; tsa[$1]+=$5; if ($4>clc) clc = $4;}\
END {print "Solutie b:";\
print "Total conectari:"; for (u in tc) print "\t", u, tc[u];\
print "Durata totala conectari:"; for (u in dtc) print "\t", u, dtc[u];\
print "Total site-uri accesate:"; for (u in tsa) print "\t", u, tsa[u];\
print "Cea mai lunga conectare:", clc;}' log

#awk5 cerinta c.
awk 'BEGIN {csc = 999999999;}\
END {print "Solutie c:";\
print "Total conectari:"; for (u in tc) print "\t", u, tc[u];\
print "Durate conectari:"; for (u in dtc) print "\t", u, dtc[u];\
print "Cea mai scurta conectare:" csc;}\
NF >= 4 {tc[$2]++; dtc[$2]+=$4; if ($4 < csc) csc = $4;}' log

```

Solutiile celor trei rulari sunt:

```

Solutie a:
Total useri conectati:
    alaltaieri 1
    ieri 2
    azi 2
    maine 1
Durate totale conectari:
    alaltaieri 400
    ieri 42
    azi 80
    maine 3
Solutie b:
Total conectari:
    popescu 2
    ionescu 1
    dan 3
Durata totala conectari:
    popescu 80
    ionescu 3
    dan 442
Total site-uri accesate:
    popescu 260
    ionescu 20
    dan 430
Cea mai lunga conectare: 400
Solutie c:
Total conectari:
    www.scs.ubbcluj.ro 4
    linux.scs.ubbcluj.ro 2
Durate conectari:
    www.scs.ubbcluj.ro 118
    linux.scs.ubbcluj.ro 407
Cea mai scurta conectare:3

```

1.5.8. Rearanjarea cuvintelor din liniile unui fisier

Se dă un string numit **dictionar de prefixe**, fiecare prefix urmat de ':', si un fisier text. Se cere crearea unui alt fisier, ale caror linii au acelasi continut ca si cele din fisierul initial, dar puse intr-o alta ordine. Noua ordine este urmatoarea: mai intai cuvintele care incep cu unul din prefixele din dictionar, in ordinea acestor prefixe; apoi cuvintele ramase, in ordinea din linia initiala. In noul fisier, cuvintele vor avea cate un sufix de forma: ";n", unde n este numarul de pozitie in linia initiala.

```
awk 'NR==1{split(dict, d, ":"); print dict;}\
$0 != "" {print $0;\
for (i=1; i<=NF; i++) if ($i != "") $i = $i ":" i;\
i=0;\
for (k=1; d[k]!=""; k++)\
  for (j=i+1; j<=NF; j++)\
    if (index($j,d[k])==1) {\
      i++; t = $j;\
      for (l=j; l>=i; l--) $l = $(l-1);\
      $i = t;\
    }\
print $0;}' dict={:c:FI: awk1
```

Ca fisier de prelucrat am folosit programul de rezolvare a exemplului 1 de mai sus. Ca dictionar am folosit trei cuvinte (intamplator primele doua au cate un caracter).

Ca rezultat se tipareste dictionarul si fiecare linie inainte si dupa prelucrare:

```
{:c:FI:
awk '{car += length($0)+1; cuv += NF;}\
cuv:5 awk:1 '{car:2 +=:3 length($0)+1;:4 +=:6 NF;}\:7
END {print "Fisier:" FILENAME, "Linii:" NR, "Cuvinte:" cuv, "Caractere:" car;}'\
{print:2 cuv,:8 car;}'\:10 FILENAME,:4 END:1 "Fisier"::3 "Linii"::5 NR,:6
"Cuvinte"::7 "Caractere"::9
deprelucrat
deprelucrat:1
```

1.6. Probleme propuse

1. Sa se afiseze pentru fiecare fisier dat ca parametru numarul de cuvinte si numarul de caractere.
2. Sa se afiseze numarul maxim de linii consecutive care coincid dintr-un acelasi fisier dat ca parametru si continutul liniei respective, precum si numele fisierului care o contine.
3. Sa se afiseze pentru fisierele date ca parametri, pentru liniile din acestea care sunt mai lungi de 30 de caractere, numarul liniei (din cadrul fisierului), primul cuvint si ultimul.
4. Sa se afiseze pentru fiecare fisier dat ca parametru primele trei caractere din fiecare cuvint. Daca lungimea unui cuvint este mai mica decat 3, acesta va fi completat cu caracterul blank.
5. Sa se afiseze din fiecare fisier dat ca parametru numerele liniilor care au lungimea cel putin 10. De asemenea sa se afiseze continutul liniilor respective mai putin primele 10 caractere. La terminarea analizei unui anumit fisier se va afisa numele fisierului si numarul de linii care au fost afisate.
6. Sa se afiseze liniile din fisierele date ca parametru care contin un acelasi cuvint aflat in pozitii consecutive. Pentru liniile respective sa se afiseze si numarul liniei (in cadrul fisierului din care face parte).
7. Sa se afiseze continutul fisierelor date ca parametru dupa cum urmeaza: primul fisier afisat asa cum este iar fisierul urmator cu cuvintele din linii (cuvintele fiind separate de :) scrise in ordine inversa. (Modul de afisare se reia pentru fisierele urmatoare).
8. Sa se afiseze continutul fisierelor date ca parametru, fiecare fisier fiind afisat incepind cu ultima linie, continuind cu cea anterioara acesteia s.a.m.d.
9. Exista un fisier 'file1', care are 2 coloane de numere. Se cere crearea unui nou fisier intitulat 'file2' care contine coloanele 1 si 2 ca in primul fisier, dar mai contine o a treia coloana reprezentand raportul

numerelor din prima si a doua coloana. In cel de-al doilea fisier vor apare doar liniile pentru care coloana 1 este mai mica decat coloana 2.

10. Sa se afiseze numarul total de bytes din toate fisierele din directorul curent care au fost modificate ultima data in luna noiembrie (a oricarui an).

11. Dandu-se un fisier de configurare de forma de mai jos, sa se copieze fisierele din stanga in fisierele din dreapta. Fisierul de configurare are forma:

/home/x/awks/temp/file1	/home/x/final
/home/x/awks/temp/file2	/home/x/final
/home/x/awks/temp/file3	/home/x/final
/home/x/awks/temp/file4	/home/X/final

Indicatie: pentru a executa comanda de copiere a fisieleror, folositi functia system(comanda), care executa comanda data ca parametru.

12. Sa se afiseze lungimea si continutul celei mai lungi linii din fisierele date ca parametru.

13. Sa se afiseze dintr-o lista de fisier date ca parametri numele celui fisier care are numar maxim de cuvinte si numarul cuvintelor.

14. Sa se afiseze numarul de fisier, numarul total de cuvinte si numarul mediu de cuvinte din fisierele date ca parametri.

15. Sa se afiseze ora curenta sub forma ora xx, xx minute, xx secunde