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

Contents

1.1. OS Unix; deosebiri formale Unix - Windows:

Istorii 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 daca se intra sub Unix sau Windows).
- Folosirea unei masini virtuale **VmWare, VirtualBox**.
- Windows **assistant Linux**
- **Termux** – terminal Linux for Android

Deosebiri formale Unix – Windows:

		Unix	Windows
1	Specificare absoluta fisier	**/dir1/dir2/.../dirn/fisier**	**d:\dir1\dir2\...\dirn\fisier**
2	Separator directoare PATH	**dir1:dir2:...:dirn**	**dir1;dir2;...;dirn**
3	Specificare optiune	**com –opt**	**com /opt**
4	Separtor linii in fisier text (Mac OS linie\rlinie CR)	linie**n**linie (LF = 0A)	linie**r****n**linie (CR LF = 0D 0A)
5	Parametrii linie comanda: 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 opţiuni expresii fişiere

"ncomandă" este numele propriu-zis al comenzii;
"opţiuni" o opţiune Unix este specificată de obicei printr-o singură literă. In unele cazuri, litera este urmată de un argument şir de caractere sau număr întreg. Un grup de opţiuni este de obicei precedat de - (minus). Exista si formele lungi, prefixate de --
"expresii" sunt şiruri de caractere, utilizate ca argumente pentru comanda respectivă. Un caz particular sunt *expresiile regulare*, (care indică machete sintactice ale unor şiruri), de care ne ocupăm în secţiunea următoare.
"fişiere" reprezintă unul sau mai multe fişiere specificate relativ (doar numele acestora) sau absolut (cu cale completă) sau specificări generice ('*****' înlocuieşte orice şir de caractere, '**?**' înlocuieşte orice caracter, `[şir]` inlocuieste un caracter ce e in sir, `[!şir],` inlocuieste orice caracter ce nu e in `şir`).
Exemple:

- Forma scurta **`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`**

In mod implicit, comenzilor Unix le sunt asociate trei fişiere: *fişierul standard de intrare*, *fişierul standard de ieşire, fişierul standard de erori*. Uneori, dacă se doreşte referirea la ele în linia de comandă, aceste fişiere pot fi specificate prin `&0 &1 &2` sau `0, 1 2`.

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

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

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

`comanda >fout` sau **`comanda >>fout`** ieşirea standard va fi depusă în fişierul **`fout`**; dacă se foloseşte semnul "**>**" se va crea un fişier nou cu numele specificat în care se va scrie output-ul comenzii (în cazul în care fişierul există, conţinutul acestuia este suprascris); când se utilizează "**>>**" output-ul comenzii este adăugat la sfârşitul fişierului **`fout`** dacă fişierul există deja, în caz contrar creându-se fişierul respectiv. (Se poate si **1>fout**, **1>>fout**)

`2>&1` specifică faptul că pentru comandă fişierul de erori standard va fi acelaşi cu fişierul de ieşire standard. (Se poate si **2>ferrori**)

`comanda1 | comanda2` ieşirea 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 ieşire standard pentru **`comanda1`**, nici intrare standard pentru **`comanda2`**. Cele două comenzi aşteaptă una după cealaltă livrarea / primirea de octeţi prin acest pipe.

1.3. Expresii regulare: definire şi exemple

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

In cele ce urmează vom descrie, în cadrul cel mai general, notaţiile folosite **în expresiile regulare Unix.** Există multe tipuri de expresii regulare: Python, PERL, Java C++ etc. In esenţă există asemănări între acestea, dar din păcate sunt şi deosebiri. Mai mult, chiar expresiile regulare Unix sunt specifice unor anumite comenzi şi pot să difere, nesemnificativ, între ele.

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

Expresie regulară	Semnificaţie
.	orice caracter
\c	caracterul **c** îşi pierde eventualul statut de caracter special
[lista]	un singur caracter, oricare din **lista**
[c1-c2]	orice caracter cuprins lexicografic între caracterele **c1** şi **c2**
[^lista]	negaţia lui **[lista]**, deci un singur caracter, care nu este în listă
^	următorul şablon se aplică numai la început de linie
$	următorul şablon se aplică numai la sfârşit de linie
\<	semnifică început de cuvânt (un cuvânt este format din litere, cifre sau -, orice alt caracter este considerat separator)
\>	semnifică sfârşit de cuvânt
r*	şirul vid sau concatenarea repetată a expresiei regulare **r** cu ea insasi ori de câte ori
r+	concatenarea repetată a expresiei regulare **r** cu ea insasi cel puţin o dată
r?	şirul vid sau expresia regulară **r**
(r)	expresia **r** privită ca o singură entitate; în anumite situaţii este **\(r\)**
r1 r2	rezultatul concatenării expresiei regulare **r1** urmată de **r2**
r1 | r2	fie expresia regulară **r1**, fie expresia regulară **r2**
r\{n,m\}	repetă expresia regulară **r** de cel puţin **n** ori şi de cel mult **m** ori
n,m	partea de text dintre liniile **n** şi **m**
. (caracterul punct)	indică la editare linia curentă
$ (caracterul dolar)	indică la editare ultima linie
/şir/	prima dintre liniile următoare faţă de linia curentă care conţine **şir**
?şir?	prima.dintre liniile precedente faţă de linia curentă care conţine **şir**

Iată câteva exemple:
`[123]` - oricare dintre cifrele 1, 2, sau 3
`[123 ]` - 1, 2, 3, sau spatiu
`[a-z]` - orice litera mica
`[aeiou]` - orice vocala
`[A-Z]` - orice litera mare
`[0-9]` - orice cifra
`[^0-9]` - orice care nu este cifra
`[^,.:]` - orice cu exceptia virgula, punct, doua puncte
`.*` - orice secvenţă de caractere
`[a-zA-Z02468]` – orice literă mică, mare sau cifră pară

`[ \t]` – spaţiu sau tab
`^[^0-9]\+$` -orice linie nevidă care nu conţine cifre
`\([Nn][Oo] \)\+` - orice refuz, oricât de insistent! (No no no no no ...)

O construcţie 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 se poate defini, mai mult sau mai puţin exact:
`+\@.+\..+`
sau
`[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}`

Sintaxa unui url:
`^http:\/\/[a-zA-Z0-9_\-]+\.[a-zA-Z0-9_\-]+\.[a-zA-Z0-9_\-]+$`
sau
`^(https?:\/\/)?[0-9a-zA-Z]+\.[-_0-9a-zA-Z]+\.[0-9a-zA-Z]+$`

1.4. Clasificarea comenzilor; comenzi filtru

1.4.1. Clasificarea comenzilor Unix

Comenzile Unix pot fi clasificate în:

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

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

Dintre cele mai populare comenzi Unix amintim:

- Filtre Unix – intrarea este un fişier text (sau mai multe) sau în absenţă intrarea standard, o transformă şi dau transformarea la ieşirea standard: `grep, sort, uniq, cut, awk`

- Comenzi de lucru cu fisiere (+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 informatii despre useri: `finger, w, who, ps, last, id, users`

- Comenzi pentru informatii de retea: `netstat, ping, hostname, host, ftp, who`

- Alte comezi: `clear, date, mail, uptime, df, fg, bg`

Pentru aceste filtre prezentăm sumar sintexele şi un exemplu. Pentru detalii se pot consulta manualele acestor filtre.

1.4.2. Filtrul grep

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

`grep [`*`optiuni`*`] [-e `*`pattern`*` | -f fis_pattern] [fisier ...]`
Printre opţiuni amintim:

- `-E` - folosirea axpresiilor regulare extinse
- `-v` – afisează liniile ce NU se potrivesc cu pattern
- `-i` – comparări ignorecase
- -c numara liniile ce verifica"^$"
- `-q` – întoarce codul de retur 0 daca nu apar potriviri, 1 dacă apar.

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

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

Câteva exemple de căutare într-un fişier text **`a.txt`**: se vor afisa liniile nevide, liniile vide, liniile cu un număr par de caractere, liniile cu nume de ocean, liniile ce conţin 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 fişierul **`/etc/passwd`**, care are structura:

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

Se vor afişa liniile ce conţin stringul `dan`, ce conţin stringul `dan` la început de linie indiferent de litere mari sau mici, liniile ce au un `userName` format numai din cifre, `userName` cu cel putin 2 sau 5 vocale, `shell` sa 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 neconversaţional. El preia un fişier (sau mai multe, sau intrarea standard), aplică asupra lui comenzi de editare, după care rezultatul este tipărit la ieşirea standard (sau cu numele vechiului fişier daca se pune opţiunea `-n`).

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

De exemplu, să se elimine toate secvenţele **<cifră><literă><cifră>** dintr-un fişier:

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

Câteva transformări din **.a.txt**: înlocuirea vocalelor cu spaţii, convertirea vocalelor mici în vocale cu litere mari, ştergerea liniilor ce conţin cel puţin 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 constructii de expresii regulare mai deosebite in sed. Constructia `&` (sau `\&` depinde de context) semnifica continutul patternului care s-a potrivit. De exemplu, comanda de editare

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

pune în paranteză primele două cifre de la inceputul fiecarei linii: `4064123456` este înlocuit cu `(40)64123456`

Constructiile \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/'
```

înlocuieşte fişierul iniţial `phone.txt`:

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

cu noul conţinut:

```
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 şi eventual concatenează fişierele specificate în intrare şi le dă la ieşirea standard cu liniile ordonate alfabetic. In absenţa specificării fişîerelor de intrare, se ia intrarea standard.

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

Comanda are un mare număr de opţiuni şi acestea pot diferi de la un tip de Unix la altul. Oricum, prin opţiuni se pot specifica tipurile de comparaţii de sortare (text, numeric, date calendaristice etc. De

asemenea, se pot indica porţiunile din linii şi ordinea acestora care se compară în vederea stabilirii ordinii. In fine, se poate cere eliminarea duplicatelor. In absenţa opţiunilor, la ieşire se furnizează liniile ordonate alfabetic.

1.4.5. Filtrul uniq

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

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

Fără opţiuni, din liniile adiacente identice se dă la ieşire doar prima.

1.4.6. Filtrul cut

Preia linii de la fişierul (fişierele) de intrare sau de la intrarea standard şi la ieşire, din fiecare linie, dă la ieşire numai porţiunile indicate.

Indicarea porţiunilor se face prin liste cu construcţii de forma `n n-m n- -m`, unde `n` şi `m` sunt întregi, `n < m`, indicând numărul elementului (începând cu 1), de la `n` la `m` inclusiv, de la `n` pana la sfârşit, de la început până la `m`.

Numărarea se face la nivel de octet (`-b`), de caracter (`-c`) sau de câmp (`-f`). Diferenţa între caracter şi octet apare atunci când liniile conţin octeţi cu codul între 128 şi 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 fişier FIS ce conţine:

```
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 şi definirea programelor awk

Acest utilitar prelucrează fişiere text, selectând acele linii din text care satisfac anumite conditii (şabloane, expresii regulare), carora li se aplica o serie de actiuni. Numele utilitarului vine de la cei trei proiectanţi şi implementatori ai lui: A. Aho, P. Wieinberger şi B. Kerninghan. In prezent există mai multe variante îmbunătăţite: **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 începand din dreapta:

Intrarea în `awk` este constituită din lista de fişiere ale căror nume sunt date în linia de comandă: `fisier . . .` Aceasta lista poate lipsi, caz in care se prelucreaza intrarea standard.

Rezultatul filtrării prin `awk` este afişat la ieşirea standard.

Optiunea `-v` precede definirea unor variabile globale si a valorilor acestora: `var=val ....` Nu sunt necesare decăt specificările atribuirilor `var=val`, specificarea `-v` poate lipsi. Mai mult, prezenta lui `-v` in fisier de comenzi nu functioneaza! Este vorba de mecanismele de tratare a optiunilor atunci cand si awk si shell le trateaza.

Optiunea -F specifica caracterul `c` care va fi separator de cuvinte. In absenta optiunii, separatorul implicit este orice spatiu alb (`BLANK, TAB \n \r`).

program poate fi scris fie direct in comanda `awk`, fie pregatit in prealabil in **fisier_program** si indicat prin optiunea **-f.** Liniile din **program** sunt de forma:

```
conditie { instructiuni }
```

`awk` tratează pe rând câte o linie din fişierele de intrare şi pentru fiecare execută `instructiuni` atunci când `conditie` ia valoarea `true`. de Dacă `conditie` lipseşte atunci se considera implicit adevarata. .

Sintaxa conditiilor si a instructiunilor sunt similare cu cele din limbajul C. Variabilele nu trebuie sa fie declarate, ele se initializeaza automat. Tipul lor se deduce din context. Initial, valorile variabilelor sunt 0 pentru numere si "" (sirul vid) pentru siruri. Operanzii pot fi expresii aritmetice, expresii relationale, constante şi variabile. Pentru variabilele de tip şir operatorul de concatenare este spatiul. Exista câteva funcţii de lucru cu şiruri. Se pot folosi variabile de tip tablou ale căror indici pot să fie numerici (cu numerotarea incepand de la 1) sau şiruri de caractere - acestea din urmă sunt **tablouri asociative.**.

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

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

Condiţii predefinite:

- `BEGIN` este adevarată înainte de prima linie din primul fişier
- `END` este adevarată după ultima linie din ultimul fişier

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 fisierelor deja prelucrate plus cea curenta a fisierului curent.
- `FNR` - numărul de ordine al liniei curente din fisierul curent; liniile cu nr. 1 sunt primele linii din fiecare fisier; numaratoare începe de la 1 la începutul fiecărui fişier
- `FS` - separatorul de câmpuri (spatiul alb sau optiunea `-F`)
- `FILENAME` - numele fişierului curent care este tratat
- `OFS` - separator de câmpuri la ieşire (implicit este spaţiu)
- `ORS` - separator de înregistrări la ieşire (implicit este linie nouă)
- `ARGV` – şirul parametrilor din linia de comandă. **specificarea `program` sau `-f fisier_program` nu se ia in considerare ca argument.**
- `ARGC` – numărul parametrilor din linia de comandă. **Vezi mai sus.**
- variabilele globale definite prin optiunea -v.

Accesarea câmpurilor:

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

Funcţii predefinite:

- `length(sir)` - lungime şir; `length` <=> `length($0)`
- `substr(s,p,n)` - subşirul lui `s` care începe la poziţia `p` şi are lungimea `n`
- `index(s1,s2)` - întoarce poziţia la care `s2` apare în `s1` sau 0 la absenţă
- `sprintf(format, arg1,.. )` - întoarce ca rezultat şirul pe care `printf` l-ar tipări în C
- `split(s,a,c)` - unde `s` este şir, a este tablou şi `c` un caracter. Împarte şirul `s` în câmpuri considerând ca separator caracterul `c`; dacă `c` lipseşte atunci separatorul implicit este `FS`. Valorile împărţite sunt date ca valori elementelor tabloului `a`
- `system(cmd)` - executa comanda shell cmd si returneaza codul sau de retur

Instrucţiuni

- `variabilă = exprsie`
- instrucţiunile `if, for, while` ca şi în C
- `for (i in numetablou) instrucţiune`. `i` ia ca valori indicii lui `numetablou` şi se execută `instrucţiune` pentru fiecare valoare a lui `i`
- ; este separator de instrucţiuni
- } este separator de linie, continuarea unei linii se face cu caracterul \ pe ultima poziţie din linie
- `print listă-expresii [ > nume-fiş]` - afişează la ieşirea standard valoarea expresiilor separate prin OFS, iar la sfârşit de linie pune ORS. Dacă se specifica `>nume-fis` atunci scrierea se face în fişierul `nume-fis`.

1.5.2. Câteva exemple simple

In exemplele care urmeaza vom include `program` direct in linia de comanda `awk` şi preluăm intrarea din `/etc/passwd`. Acţiunile ce ni le propunem sunt: afişare `userName` (şi numai el), afişare `fullName` din liniile pare, afişare `homeDirectory` al carui `userName` începe cu vocală, afişare `fullName` al

cărui `UID` este par, afişeaza `userName` al cărui `shell` este `nologin`, afişeaza `fullName` al carui `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 fişiere program:

```
NR % 2 == 0 && $4 < 20 {print $5}
```
Afişeaza din liniile pare full name cu `GID` < 20

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

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

Face produsul digerenţelor dintre UID si GID

1.5.3. Ilustrare moduri de apel: numararea liniilor, cuvintelor si caracterelor

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

Mai intai solutia cu programul **`awk`** scris intr-un fisier separat. Vom pregati in fisierul **`fisp`** programul:

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

(la `length($0)` se adauga 1 pentru a numara terminatorul de linie). Comanda de numarare va fi:

```
awk -f fisp deprelucrat
```

Dacă fişierul **`deprelucrat`** are conţinutul:
```
ggg oooiioi 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 afiseze liniile din fisierul `f` pentru care ultimul cuvant este `CevaAiurea` si numarul curent al fiecarei astfel de linii. Prezentam 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 poihphp CevaAiurea
gggg
```

atunci rezultatul celor patru executii va fi:
```
SIR 1 CevaAiurea
SIR 3 hphph poihphp 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 afişeze liniile mai lungi de 5 caractere

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

```
# Solutia cu un singur fisier
awk 'length>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:

a. Pentru fiecare data, numarul total de useri conectati, durata totala a conectarilor.
b. Pentru fiecare user numarul total de conectari, durata totala a conectarilor, totalul siteurilor accesate, cea mai lunga conectare.
c. 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 cuvant 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 decit 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 aparea 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 fisierelor, 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 fisiere date ca parametri numele acelui fisier care are numar maxim de cuvinte si numarul cuvintelor.

14. Sa se afiseze numarul de fisiere, 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