# Laboratorul 2

# Shell

## 1 Variabile

Variabilele shell nu au tip (ex. întreg, caracter, string, vector). Ele sunt inițializate direct umând convenția matematică `variabilă=valoare`.

```
$ a=2009
$ b=alex
$ c="fata merge pe jos"
```

**Atenție**, nu trebuie să existe spații în jurul semnului `=`.

```
$ a = 2009
ksh: a: not found
```

Pentru a folosi o variabilă, numele acesteia trebuie prefixat cu `$`:

```
$ echo $a, $b, $c
2009, alex, fata merge pe jos
```

La inițializarea unei variabile pot participa alte variabile:

```
$ d="$b: $c ($a)"
$ echo $d
alex: fata merge pe jos (2009)
```

O variabilă este extinsă reinițializând-o cu valoarea veche împreună cu elementele noi:

```
$ name=bond
$ echo $name
bond
$ name="james $name"
$ echo $name
james bond
```

Variabilele de mediu sunt folosite atât de shell cât și de unele comenzi de sistem. Prin convenție sunt scrise cu litere mari (vezi `PATH` din laboratorul trecut) și sunt manipulate în același mod ca variabilele simple.

## 2 Expresii logice

Toate shell-urile POSIX oferă acces la expresii logice pentru a procesa și lega diferite comenzi. Expresiile logice primare de tip `și` și `sau` au aceiași sintaxă ca în limbajul C: `&&`, respectiv `||`.

Comenzile efectuate în shell se execută cu succes sau nu și întorc o valoare pentru a semnala acest lucuru. Valoarea întoarsă și semnificația ei este documentată în manual. În general o valoare nulă semnifică succes și una nenulă eșec. Din această cauză **modul de operare al expresilor logice este pe dos**:

- `cmd1 && cmd2` – execută cmd2 doar dacă ieșirea lui cmd1 este zero
- `cmd1 || cmd2` – execută cmd2 doar dacă ieșirea lui cmd1 este nenulă

Valorea întoarsă de ultima comandă executată este salvată în variabila `?`. În următorul exemplu, întâi creăm un fișier nou `animals.txt` în care punem cuvântul `cat`, după care căutăm pe rând cuvintele `cat` și `dog` în acest fișier. Prima dată căutarea se întoarce cu succes (zero), iar a doua oară fără nici un rezultat deci cu valoare nenulă (unu).

```
$ echo cat > animals.txt
$ grep cat animals.txt
cat
$ echo $?
0
$ grep dog animals.txt
$ echo $?
1
```

În funcție de ieșire se iau diferite decizi. De exemplu, fie fișierul `agenda.txt` în care se găsesc datele de contact a diferitor persoane. Pentru a căuta dacă o anumită persoană există în agendă se poate folosi comanda `grep(1)`

```
$ echo Ana > agenda.txt
$ grep Ana agenda.txt
ana
$ grep Mara agenda.txt
$
```

Mai sus se constată că `ana` există în agendă, dar `mara` nu. La finalul execuției comenzii `grep(1)` aceasta iese fie cu valoarea 0, dacă s-au găsit una sau mai multe intrări, sau 1 dacă nu s-a găsit nimic. Pentru a obține un verdict mai prietenos putem folosi expresia logică `sau`

```
$ echo Ana > agenda.txt
$ grep Ana agenda.txt || echo "Persoana nu exista!"
Ana
$ grep Mara agenda.txt || echo "Persoana nu exista!"
Persoana nu exista!
```

Dacă nu ne amintim dacă am trecut-o pe Ana-Maria drept Ana sau Maria în agendă, putem de asemenea folosi expresii logice pentru a căuta după Maria în caz că Ana un există:

```
$ echo Maria > agenda.txt
$ grep Ana agenda.txt || grep Maria agenda.txt
Maria
```

# 3 Compilare

Deocamdată nu am învățat cum să folosim un editor în linia de comandă. Din acest motiv vom folosi comanda `cat(1)` (vezi Laboratorul 1 Secțiunea 4) pentru a scrie programe scurte C pe care să le compilăm și executăm. Când comanda `cat(1)` primește ca argument caracterul `-`, atunci comanda așteaptă date de la tastatură.

```
$ cat -
$ cat -
echo
echo
stop doing that
stop doing that
press ctrl-d
press ctrl-d
```

`cat(1)` continuă să citească date până când utilizatorul apasă simultan tastele `Ctrl` și `d`, pe scurt `Ctrl-d`.

Implicit, comanda `cat(1)` repetă datele de intrare pe ecran. Pentru a scrie într-un fișier datele de la intrare putem să folosim operatorul de redirecționare `>` (vezi Laboratorul 1). Astfel avem toate ingredientele pentru a scrie primul nostru program C:

```
$ cat - > hello.c
#include <stdio.h>

int main()
{
printf("Hello, World!\n");
return 0;
}
```

după ultima acoladă { apăsați `Ctrl-d` pentru a încheia șirul de date de intrare. Verificăm dacă noul fișier conține într-adevăr ce am introdus de la tastatură tot cu comanda `cat(1)`:

```
$ cat hello.c
#include <stdio.h>

int main()
{
printf("Hello, World!\n");
return 0;
}
```

În sistemele de operare de tip UNIX, compilatarul de C este invocat prin comanda `cc(1)` (scurt de la *C compiler*). Acesta așteaptă ca argumente fișierele sursă și, opțional, numele executabilului rezultat în urma compilării.

```
$ cc hello.c -o hello
```

În comanda de sus compilatorul primește fișierul C `hello.c` și numele executabilului `hello` transmis prin opțiunea `-o` cu *o* de la *ouptut*. **Atenție**, dacă omiteți specificarea unui nume pentru executabil acesta va fi implicit denumit `a.out` din motive istorice.

Pentru a executa binarul rezultat se folosește comanda:

```
$ ./hello
Hello, World!
```

unde `hello` este numele executabilului, iar `./` spune shell-ului să nu caute executabilul în `$PATH` pentru că se află în directorul curent. În Laboratorul 1 am învățat că `.` simbolizează directorul curent.

# 4 Sarcini de laborator

1. Executați toate comenzile prezentate în acest laborator.
2. Compilați și executați programul `hello` doar dacă compilarea a decurs cu succes. Faceți asta într-o singură comandă folosind expresii logice.
3. Modificați programul `hello` să citească un nume cu `scanf` (ex. Alex) pe care să-l salute pe urmă cu ajutorul funcției `printf` (ex. "Hello, Alex!"). Compilați și executați.
4. Creați un director `bin` în care copiați executabilul `hello`. Adăugați directorul `bin` (**folosind calea absolută**) în variabila `$PATH` și arătați că puteți executa simplu cu comanda `hello` fără a avea nevoie de prefixul `./`.