Laboratorul 2

1 Căutarea

Căutarea unui fragment de text într-un fişier se poate face folosind expresii regulate în cadrul comenzii `grep`(1). Fie următorul fişier text:

```
$ echo -e "words\nrat\nrata\nratat\nratata\nrata tata" > rat
```

unde `\n` reprezintă o linie nouă (ca în limbajul C). Pentru a găsi cuvintele ce conţin `at` în fişier folosim comanda:

```
$ grep at rat
rat
rata
ratat
ratata
rata tata
```

Dacă vrem să găsim cuvintele care se termină în litera `t` atunci vom folosi comanda:

```
$ grep t$ rat
rat
ratat
```

unde caracterul `$` simbolizează sfârşitul liniei. Pentru începutul liniei se foloseşte `^`:

```
$ grep ^w rat
words
```

Alte caractere speciale utile, numite *wildcards*, sunt:

- `*` – găseşte de 0 sau mai multe ori atomul precedent
- `+` – găseşte de 1 sau mai multe ori atomul precedent
- `?` – găseşte de 0 sau 1 ori atomul precedent

unde un *atom* este implicit caracterul precedent sau grupul precedent de caractere. Gruparea mai multor caractere se face cu ajutorul parantezelor rotunde `()`. În exemplul nostru putem căuta toate cuvintele care încep cu `rata` şi sunt urmate de `ta` de 0 sau mai multe ori astfel:

```
$ grep -E 'rata(ta)*' rat
rata
ratat
ratata
rata tata
$ grep -E 'rata(ta)+' rat
ratata
rata tata
```

```
$ grep -E 'rata(ta)?' rat
rata
ratat
ratata
rata tata
```

Observaţi că am folosit parantezele pentru grupare şi opţiunea `-E` pentru a folosi expresii regulate.

2 Liste de comenzi

Aşa cum s-a menţionat la curs, o listă de comenzi care se execută secvenţial poate fi tipărită pe o singură linie la promptul shell de maniera următoare:

- `cmd1; cmd2; cmd3; ...` – se execută mai întâi `cmd1`, după ce se termină `cmd1` se execută `cmd2`, şamd.

Iată un exemplu concret:

```
$ echo -n "prima lista de comenzi a utilizatorului "; whoami; ls -l
```

În comanda de mai sus (o listă de comenzi este până la urmă şi ea o comandă), `echo -n` suprimă afişarea caracterului newline.

2.1 Expresii logice

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

Comenzile efectuate în shell se execută cu succes sau nu şi întorc o valoare pentru a semnala acest lucru. Valoarea întoarsă şi semnificaţia ei este documentată în manual. În general o valoare nulă semnifică succes şi una nenulă eşec. Din această cauză modul de operare al expresiilor logice este pe dos:

- `cmd1 && cmd2` – execută `cmd2` doar dacă ieşirea lui `cmd1` este zero;
- `cmd1 || cmd2` – execută `cmd2` doar dacă ieşirea lui `cmd1` este nenulă.

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

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

În funcţie de ieşire se iau diferite decizii. De exemplu, fie fişierul `agenda.txt` în care se găsesc datele de contact ale 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
```

La finalul execuţiei 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 obţine 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` nu există:

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

3 Compilarea programelor C

3.1 Compilarea şi executarea unui program C

În sistemele de operare de tip UNIX, compilatorul de C este invocat prin co- manda `cc(1)` (scurt de la *C compiler*). Acesta aşteaptă ca argumente fişierele sursă şi, opţional, numele executabilului rezultat în urma compilării. În Linux, compilatorul uzual de C este varianta GNU, iar compilatorul se numeşte `gcc`.

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

În comanda de sus, compilatorul primeşte fişierul C `hello.c` şi numele executabilului `hello` transmis prin opţiunea `-o` (de la *output*). Atenţie, dacă omiteţi specificarea unui nume pentru executabil acesta va fi implicit denumit `a.out` din motive istorice.

Pentru a executa binarul rezultat se foloseşte comanda:

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

unde `hello` este numele executabilului, iar `./` spune shell-ului să nu caute exe- cutabilul în `$PATH` pentru că se află în directorul curent.

Formatul cel mai general al funcţiei `main` conţine în semnătura funcţiei nu- mărul argumentelor din linia de comandă (`argc`), un vector de stringuri care conţine argumentele

propriu-zise din linia de comandă (`argv`), şi respectiv me- diul de execuţie reprezentat ca un vector de stringuri (`envp`). Fiecare element al vectorului care reprezintă mediul de execuţie are forma discutată `nume=valoare`. Aceasta este modalitatea prin care shell-ul comunică mediul de execuţie comenzilor pe care le lansează şi care astfel pot folosi valorile variabilelor de mediu pentru a-şi adapta comportamentul la mediul de execuţie.

Compilaţi şi executaţi similar cu programul `hello.c` de mai sus programul `printenv.c` de mai jos, care tipăreşte prima variabilă de mediu din listă.

```
#include <stdio.h>

int main(int argc, char *argv[], char *envp[])
{
  printf("Prima variabila de mediu este %s\n", envp[0]);
  exit(0);
}
```

Obs: Aşa cum am menţionat anterior, fiecare comandă (program executabil) Unix întoarce un cod de retur folosit aşa cum am văzut în comenzi condiţionale, de pildă. Convenţia Unix este că orice program care întoarce cod de retur 0 s-a terminat cu succes, în vreme ce un cod nenul (uzual pozitiv) desemnează o condiţie de eroare. Din acest motiv, este o bună practică de programare să încheiaţi programele folosind apelul `exit` care este responsabil pentru a comunica acest cod de retur, care e returnat de către shell prin evaluarea variabilei `$?`.

4 Sarcini de laborator

1. Utilitarul `wc(1)`, cu opţiunea `-l` va afişa numărul de linii al fişierului (sau fişierelor) primit ca **argument**.

   ```
   $ wc -l curs1.tex
   300 Curs1/curs1.tex
   ```

   Să presupunem că am vrea sa efectuăm această operaţie pentru toate fişierele cu o anumită proprietate. Încercăm acest lucru în două moduri, dintre care doar al doilea funcţionează:

   ```
   $ find . -name "*.tex" | wc -l
   3
   $ find . -name "*.tex" | xargs wc -l
     300 ./Curs1/curs1.tex
     255 ./Curs2/curs2.tex
      84 ./Curs0/curs0.tex
     639 total
   ```

   Citiţi documentaţia pentru `xargs(1)` şi încercaţi să explicaţi comportamentul diferit al celor două comenzi de mai sus.

2. Creaţi un director `logs` şi câteva fişiere în acesta. În unele (nu toate) dintre acestea, adăugaţi textul `error`. Folosind `grep(1)` şi `rm`, ştergeţi toate fişierele care conţin textul `error`.

3. Creati două directoare `orig` şi `backup`. În directorul `orig` creaţi fişierele `foo`, `bar`, `baz`. Folosiţi comanda `find`(1) să copiaţi toate fişierele din `orig` în `backup` dar cu extensia `.orig` în plus (ex. `foo.orig`).

4. Modificaţi programul `hello` să citească un nume cu `scanf` (ex. `Alex`) pe care să-l salute pe urmă cu ajutorul funcţiei `printf` (ex. `"Hello, Alex!"`). Compilaţi şi executaţi. Încercaţi sa combinaţi programul vostru cu un pipe.

5. Modificaţi programul `printenv` de mai sus astfel încât să tipărească toate variabilele de mediu (N.B. vectorul `envp` se termină cu `NULL`). Compilaţi şi executaţi.

6. Creaţi un director `bin` în care copiaţi executabilul `hello`. Adăugaţi di- rectorul `bin` (folosind calea absolută) în variabila `$PATH` şi arătaţi că puteţi executa simplu cu comanda `hello` fără a avea nevoie de prefixul `./`.