Sisteme de Operare 1 - Curs 3

Curs tinut in 2012-2013 *de catre* lector dr. Sanda-Maria Dragos

Etape in tratarea liniei de comanda

slide 2

1. Despartirea in *comenzi elementare*
2. Impartirea pe *cuvinte*
3. Substituirea *variabilelor shell*

```
Constructiile de forma:
      ${...} sau "...${...}..."
      dar NU '...${...}...'
```

4. Substituirea *fisierelor generice*

```
      *       ?       [sir]   [!sir]
```

5. Substituirea *iesirilor unor comenzi*

```
      `comanda`       si daca constructia se afla intre "...".
      Daca se afla intre '...' atunci NU se realizeaza substitutia.
```

6. Se efectueaza *redirectarile* (<, <<, >, >>, >&, <&, &<, &>)
7. Se *definesc variabilele shell*

```
      Constructiile de forma          nume=valoare
```

8. Se fixeaza valorile variabilelor $0, $1, ..., $-, $#, $*, $@
9. Se executa *comanda*

```
Evaluarea comenzii:
 1. alias                (abrevieri ale unor comenzi shell): alias ll='ls -l'
 2. cuvant cheie         : if, elif, then, else, fi, case, esac, in, for, do, done,
 3. functii              (grupuri de comenzi organizate ca rutine diferite)
 4. comanda interna      : man, ls, cat, more, sort, ...
 5. program executabil/script shell - se cauta fisierul executabil precizat in
                 toate directoarele specificate in variabila de mediu PATH
```

10. Se *stabileste valoarea variabilei* $?

expr

Evaluarea expresiilor logice

slide 3

arg1 \& arg2

- operatie de tip SI: returneaza ARG1 daca ARG1 si ARG2 != 0 sau NULL, 0 altfel. ARG1 poate fi o alta expresie

```
$ VAR=ceva                # Set VAR to ceva
$ PRT=0                   # Set PRT to NUL
$ expr $VAR \& $PRT       # Return 0, arg1 and arg2
0                         # are NUL
$ PRT=hpij                # Set PRT to hpij printer
$ expr $VAR \& $PRT       # VAR is set, return its
ceva                      # value
```

arg1 \| arg2

- operatie de tip SAU: returneaza ARG1 daca ARG1 != 0 sau NULL, ARG2 altfel. ARG1 poate fi o alta expresie

```
$ VAR=0                   # Set variable VAR to NUL
$ expr $VAR \| undefined  # If VAR is NUL return
undefined                 # the undefined string

$ VAR=(aaa \& bbb)        # Set variable VAR to expr
$ expr $VAR \| ccc        # If VAR is not NUL return
aaa                       # the aaa string
```

Evaluarea expresiilor aritmetice cu intregi

arg1 * arg2

```
$ expr 80 \* 4
320
```

arg1 / arg2

```
$ expr 80 / 4
20
```

arg1 % arg2

```
$ expr 81 % 4
1
```

arg1 + arg2

arg1 - arg2

```
$ NUM=1                          $ NUM=1
$ expr $NUM + 1                  $ expr $NUM - 1
2                                0
```

Evaluarea expresiilor de comparare (numere sau siruri)

slide 4

arg1 = arg2

- returneaza 1 daca ARG1 = ARG2.

```
$ NUM=5
$ PREV=6
$ expr $NUM = $PREV      # NUM !- PREV so
0                        # 0 is displayed.
```

arg1 \> arg2

- returneaza 1 daca ARG1 > ARG2.

```
$ expr dog \> cat        # Since d has a higher ASCII
1                        # value than c a 1 is displayed.
```

arg1 \>= arg2 arg1 \< arg2 arg1 \<= arg2 arg1 != arg2

Evaluarea expresiilor cu siruri

STRING : REGEXP

match STRING REGEXP

- comparare de siruri: returneaza numarul caracterelor din ARG1 identice cu cele din ARG2. ARG1 e un sir, iar ARG2 e o expresie regulara

```
$ expr string : str      # First 3 characters match
3
$ expr string : ing      # Last 3 match but comparison must start at beginning of
0                        # arg1
$ expr string : strg     # Arg2 must match arg1 in its entirety
0
$ expr string : '.*'     # .* is a regular expression that matches any number of
6                        # any characters
$ expr string : '.*i'    # .*i matches any set of characters ending with i
4
```

- pentru compararea unei portiuni din ARG1 se foloseste \(...\) pentr ARG2

```
$ expr string : '\(.*\)'         # Returns all chars in arg1
string
$ expr string : '..\(..\)'       # Skips first 2 chars of arg1, returns next 2
ri
$ expr string : '\(...\)'        # Returns first 3 chars of arg1
str
$ expr string : '.*\(...\)'      # Returns last 3 chars of arg1
ing
$ expr string : '..\(.*\)..'     # Returns center of arg1 first and last 2 chars
ri                               # removed
$ expr string : '\(stx\)'        # Returns nothing
$ expr string : 'st\(.*\)'       # Return all of arg1 after skipping st
ring
```

length STRING

```
$ expr length "aaa"
3
```

substr STRING POS LENGTH

```
$ expr substr "abcdefg" 1 2
ab
```

index STRING CHARS

```
$ expr index "abcdefg" d
4

$ expr index "abcdefg" da
1

$ expr index "abcdefg" dac
1

$ expr index "abcdefg" dc
3
```

```
$ expr length  "abcdef"  "<"  5  "|"  15  -  4  ">"  8
1
```

Structuri de control alternative shell

slide 5

if

```
if [ $# -ge 2 ]
then
  echo "You need at least two paraters!"
  exit 1
fi

if [ $foo -ge 3 -a $foo -lt 10 ]; then


if [ -e file ]; then echo "This file exists!"; fi

```

[[expr]] - expr este o conditie care va fi evaluata

```
if [[ "$stringvar" == *string* ]]; then         # an asterisk ("*") will expand
                                                # to literally anything
if [[ "$stringvar" == *[sS]tring* ]]; then


if [[ $num -eq 3 && "$stringvar" == foo ]]; then # combining expressions using
                                                # the operators "&&" and "||"
```

((expr)) - expr este evaluata aritmetic

```
if (( $num <= 5 )); then
                # It features all the 'normal' operators, like "==", "<" and ">=".
                # It supports the "&&" and "||" combining expressions (but not -a
```

case

- macheta este o insiruire de specificari generice folosite pentru fisiere: *(? - orice caracter, * - orice sir de caractere, [...] - orice caracter din cele specificate), separate prin |*

slide 6

```
case $1 in
     [a-z]|[A-Z]) echo "letter";;
     [0-9]) echo "digit";;
     *) echo "no letter, nor digit";;
esac
```

Structuri de control repetitive

slide 7

for

```
for x in one two three four
do
     echo number $x
done

for fis in `ls .`; do cat $fis; done

s=0;
for i in `seq 1 10`
do
     s=`expr $s + $i`
done
```

```
factorial=1
N=15
for (( i=2; $i<=$N; i++ ))
do
        factorial=$(( factorial * $i ))
done
echo $factorial
```

```
factorial=1
for filename in $*
do
     case $filename in
        *.c )
                objname=`echo $filename | cut -d. -f1`.o  # ${filename%.c}.o
                gcc $filename -o $objname ;;
        *.s )
                objname=${filename%.s}.o
                as $filename $objname ;;
        *.o ) ;;
        *   )
                echo "error: $filename is not a source or object file."
                exit 1 ;;
     esac
done
```

while

slide 8

```
count=1
while [ -n "$*" ]
do
   echo "This is parameter number $count"
   shift
   count=`expr $count + 1`
done
```

```
factorial=1;
$N=15;
$i=2;
while [ $i -le $N ]
do
     factorial=$(( $factorial * $i ))
     i=`expr $i + 1`
done
```

until

slide 9

```
count=1
until [ -z "$*" ]
do
   echo "This is parameter number $count"
   shift
   count=`expr $count + 1`
done
```

```
factorial=1;
$N=15;
$i=2;
until [ $i -gt $N ]
do
   factorial=$(( $factorial * $i ))
   i=`expr $i + 1`
done
```

slide 10

Alte instructiuni folosite in contextul structurilor de control ciclice

break, continue

- se refera la parasirea, respectiv reiterarea ciclurilor: *for, while, until*

```
for ((;;))
do
     read var
     if [ "$var" = "." ]; then
        break
     fi
done
```

```
for f in "$@"
do
     # if .bak backup file exists, read next file
     if [ -f ${f}.bak ]
     then
          echo "Skiping $f file..."
          continue  # read next file and skip cp command
     fi
     # we are hear means no backup file exists, just use cp command to copy file
     /bin/cp $f $f.bak
done
```

true, false

```
while true
do
        ps
        sleep 100
done
```

```
until false
do
        ps
        sleep 100
done
```

Functions

slide 11

slide 12