Operating systems 1, Lecture 3

Sanda-Maria AVRAM, Ph. D.

Leave a feedback

The grammar of shell (cont.)

1

command:

basicCommand

(*commandList*)

{ *commandList* }

if *commandList* then *commandList* [elif *commandList* then *commandList*]*
[else *commandList*]? fi

case word in [word [| word]*) *commandList* ;;]+ esac

for name do *commandList* done

for name in [word]+ do *commandList* done

[ ]? - once at most
[ ]+ - at least once
[ ]* - 0 or more times
word - sequence of characters not including blanks (space or tab)
name - sequence of characters that starts with a letter and continues with letters, digits or _ (underscore)

Alternative control structures: *if*

2

```
if command_list_1
    then command_list_2
    elif command_list_3
        then command_list_4
    ...
    elif command_list_n
        then command_list_n+1
    else command_list_n+2
fi
```

Alternative control structures: *if*. Examples

3

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

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

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

Alternative control structures: *if*. Examples

4

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

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

# (( )) may contain arithmetic operators, such as "==", "<" and ">=".
 # can combine expressions with "&&" and "||" (but not with -a and -o )
if (( $num <= 5 )); then ...
```

Alternative control structures: *case*

5

```
case word in
  pattern_1) command_list_1;;
     ...
  pattern_n) command_list_n;;
esac
```

pattern is a set of generic specifications used for files:
* - any character,
* - any sequence of character,
[...] - any character of the specified ones, separated by |

Alternative control structures: *case*. Examples

6

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

Repetitive control structures: *for*

7

```
for name [ in list_of_words_separated_by_space ]
do
    command_list
done

for (( expr1 ; expr2 ; expr3 ))
do
    command_list
done
```

Repetitive control structures: *for*. Examples

8

```
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
```

Repetitive control structures: *for*. Examples

9

```
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
```

Repetitive control structures: *while*

10

```
while command_list_1
do
    command_list_2
done
```

Repetitive control structures: *while*

11

```
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
```

Repetitive control structures: *while*

12

```
while read e
do
    case $e in
        [a-z]|[A-Z]) echo "letter";;
        [0-9]) echo "digit";;
        *) echo "no letter, nor digit"
           exit;;
    esac
done
```

Repetitive control structures: *until*

13

```
until command_list_1
do
   command_list_2
done
```

Repetitive control structures: *until*

14

```
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
```

continue, break, true, false

15

```
continue [n]  # Resumes the next iteration of the enclosing: for, while, until
break [n]     # Exits from within a: for, while, until loop

true
false
```

continue, break. Examples

16

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

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

true, false. Examples

17

```
while true
do
ps
sleep 100
done

until false
do
ps
sleep 100
done
```

Processing the command line

18

1. Splitting it into **basic commands** (Separators: `;` `|` `&&` `||`)
2. Further splitting into **words** (Separators: `space` `TAB` `;`) (`"..."` și `'...'` - considered as a word)
3. Replacing **shell variables** (`${...}` or `"...${...}..."`, but NO `'...${...}...'`)
4. Replacing **generic files** (`*` `?` `[siz]` `[!siz]`)
5. Replacing **the outputs of some commands** (`` `comm` `` or `"...`comm`..."`, but NO `'...`comm`...'`)
6. **Redirects** are being made (`<`, `<<`, `>`, `>>`, `>&`, `<&`, `&<`, `&>`)
7. Define **shell variables** (`name=value`)
8. It is determined the values of the variables `$0`, `$1`, ..., `$-`, `$#`, `$*`, `$@`
9. It **executes the command**
10. **The value of the variable** `$?` is set

expr command

19

- evaluates a logical, arithmetic or string expression and displays the result at the standard output

expr *expression*

expr command with logical expressions

20

```
expr arg1 \&  arg2   # arg1 if arg1 and arg2 != 0 or NULL, 0 otherwise
expr arg1 \|  arg2   # arg1 if arg1 and arg2 != 0 or NULL, arg2 otherwise
```

expr command with logical expressions. Ex.

21

```
» VAR=ceva # assigns the value ceva to variable VAR
» PRT=0 # assigns the value NULL to variable PRT
» expr $VAR \& $PRT # evaluation of the AND expression results in:
0
» PRT=hpij # assigns the value hpij to variable PRT
» expr $VAR \& $PRT # evaluation of the AND expression results in:
ceva
» expr $PRT \| undefined # evaluation of the OR expression results in:
undefined
» VAR=(aaa \& bbb) # assigns the result of aaa \& bbb to variable VAR
» expr $VAR \| ccc # evaluation of the OR expression results in:
aaa
```

expr command with arithmetic expressions with integers

22

```
expr arg1 \* arg2    # = arg1 * arg2
expr arg1 /  arg2    # = arg1 / arg2
expr arg1 %  arg2    # = arg1 % arg2
expr arg1 +  arg2    # = arg1 + arg2
expr arg1 -  arg2    # = arg1 - arg2
```

expr command with arithmetic expressions with integers. Examples


```
» expr 80 \* 4
320
» expr 80 / 4
20
» expr 81 % 4
1
» NUM=1
» expr $NUM + 1
2
» expr $NUM - 1
0
```

expr command with comparison expressions


```
expr arg1  =  arg2    # 1 if arg1 = arg2, 0 otherwise
expr arg1 \>  arg2    # 1 if arg1 \> arg2, 0 otherwise
expr arg1 \>= arg2    # 1 if arg1 \>= arg2, 0 otherwise
expr arg1 \<  arg2    # 1 if arg1 \< arg2, 0 otherwise
expr arg1 \<= arg2    # 1 if arg1 \<= arg2, 0 otherwise
expr arg1 !=  arg2    # 1 if arg1 != arg2, 0 otherwise
```

expr command with comparison expressions. Examples


```
» NUM=5
» PREV=6
» expr $NUM = $PREV # NUM != PREV, therefore:
0
» expr dog \> cat # 'd' has ASCII code greater than 'c'
1
» expr 2 "<" 5 "|" 15 - 4 ">" 8
1
```

expr command with string expressions


```
expr string : expr    <=>    expr match string regexpr
# number of characters in string identical to those in expr
expr length string # number of characters in string
expr substr string pos len
# substring of string which starts at pos and has length len
expr index string chars
# the smallest position of a character in chars in string
```

expr command with string expressions. Examples


```
» expr string : str
3
» expr string : ing
0
» expr string : strg
0
» expr string : '.*'
6
» expr string : '.*i'
4
```

```
» expr string : '\(.*\)'
string
» expr string : '..\(..\)'
ri
» expr string : '\(...\)'
str
» expr string : '.*\(...\)'
ing
» expr string : '..\(.*\)..'
ri
» expr string : '\(str\)'

» expr string : 'st\(.*\)'
ring
```

expr command with string expressions. Examples (cont.)


```
» expr length "aaa"
3
» expr substr "abcdefg" 1 2
ab
» expr index "abcdefg" d
4
» expr index "abcdefg" dac
1
» expr index "abcdefg" dc
3
» expr length "abcdef" "<" 5 "|" 15 - 4 ">" 8
1
```

Shell Functions

DEFINING:

```
function_name()
{
    shell_commands
}
```

CALLING:

```
function_name
```

Example of shell function


```
#!/bin/bash
myvar="hello"
myfunc() {
    local x
    local myvar="one two three"
    for x in $myvar
    do
        echo $x
    done
}
myfunc
echo $myvar $x
```


The End