

Computer Systems

Operating system tasks



Review

- Computers, signals, informations, implementation of numbers, code writing
- Architecture, operating system, client-server role
- Graphical, character connection, file systems
- Base commands, foreground and background processes
- I/O redirection
- Filters
- Regular expressions

What comes today?

- Variables and their usage
- Variable substitution
- Command substitution
- Elementary operations (arithmetical, string)
- Logical expressions
- ...

Variables in UNIX shell I.

- You are able to create a string variable.
 - There are environment or shell variables.
- The first character of a variable name is a letter, the followings can be letters, numbers or underlines.
- The environment variables are visible in the environment and in each command started from the environment. (global)
 - env – program is running in the defined environment
 - Without a parameter it writes out the variables!
 - In different systems the written variables may differ!
 - PATH (.profile), separator character :, PS1, LOGNAME, etc.

Variables in UNIX shell II.

- To define a variable: = symbol
 - Example: team=RealMadrid; number=5;
 - sentence="The apple is under the tree."
 - special_sentence='This is 5\$ and 3%\10% !'
- The content of a variable is always a string! Numbers are not „numbers” – they are also characters!
- set command: it writes out all of the variables (environments too)
- unset command: to delete a variable
 - unset team # set result: _=apple

The content and the scope of variable

- Example: `tallyho="Tally-ho Real Madrid!"`
- Content: `$tallyho`
 - `echo $tallyho # Tally-ho Real Madrid!`
 - `echo Say: $tallyho #substitution`
- `export tallyho`: The `tallyho` variable becomes an environment variable!
 - `unset` deletes it too!
- Environment variable can be seen by everybody!
 - A „normal“ variable can be used only by the actual shell!

Variable substitution

- `team=RealMadrid; echo $teams` # What will be the result?
- `echo $team is the best!`; # Naturally! 😊
- `echo ${team}is the best!`
- `unset team; x=${team-FCBarcelona}` # if team variable is not existing, x value will be FCBarcelona
 - If it is not existing then it is an initialization
 - `x=${team=FCBarcelona}` # team changes as well!
 - `echo $x is the best!` # ???
- `y=${team?Hello}` # if team is not defined, Hello is written out, the shell logs out, y does not get a new value!
- `y=${team+FCBarcelona}` # if team is defined, y=FCBarcelona

Command substitution

- ``command`` the command will be executed and it will be replaced with its output.
 - In Bash shell: `$(command)`
 - `who_am_i=`whoami` # $(whoami)`
 - `a=`date` ; b='date' #`
 - `echo $a # ???`
 - `echo $b # ???`
 - The result will be the word: date.
 - `eval $b #man eval`
 - The same as the command should be `$b!`

My first shell script

- The name of the file: first

```
illes@panda:~$ vi first
illes@panda:~$ chmod +x first
illes@panda:~$ cat first
echo This is my first shell script!
```

- A small modification...

```
illes@panda:~$ ./first
This is my first shell script!
illes@panda:~$ first
This is my first shell script!
illes@panda:~$
illes@panda:~$ cat first
#!/bin/sh
#
echo This is my first shell script!
```

Operations in shell

- There is only one operation: string concatenation
 - In shell each variable is a string!
- `x=red; echo summer $x apple! # ?`
 - Arithmetical expressions are not able to be evaluated directly.
 - `x=apple; y=$x+tree; echo $y # apple+tree`
 - `a=5; b=$a+1; echo $b #5+1`
- String concatenation is the only operation which is supported!

Arithmetical operations

- let instruction, is included in bash, it is not included in old sh
 - `a=2; let b=a+1; echo $b` # result 3
- expr instruction
 - `expr $a op $b` e.g.: `expr 3 * 4` !!!
 - op: `<, <=, >, >=, !=, =, +, -, *, /, %` (mod)
 - Suggestion: use expr due to compatibility!
- bc command (filter)
 - It waits for a C like input string
 - Cycle, trigonometric functions, function definitions are existing
 - E.g.: `echo 2*16 | bc` #32

BC example

- Complex example:
 - Function definition
 - Usage of square root and trigonometric function!
- Filename: bcexample
- Execution: `chmod +x bcexample`
 - `./bcexample`

```
#!/usr/bin/bc -l
# -l option means the usage of math library (for using s)
define fact (x) {
if (x <= 1) return (1);
return (fact(x-1) * x);
}
print " 5 factorial =: ";
print fact(5);
print " !\n";
print „The square root of 25 is: ";
print sqrt(25);
print "\n";
print „The value of sinus PI/2 :";
print s(3.1415/2);
print "\n";
quit
```

Command parameters

- Example: Best team is Real Madrid! #if this is a command... 😊
 - What is the name of the command?
- \$1, \$2, \$3, ... # parameter variables
 - echo \$1 # team
- \$0 # name of command (Best)
- \$# # number of parameters
- \$* # each of the parameters, no effect of double quote mark!
- „\$@" # parameters separately, example: param

Parameter example

- Call it: param peach tree 'apple tree'

```
echo usage of parameters
echo „we are giving a compound
parameter too ,apple tree””
#each of them separately
echo '$* usage'
for i in $*
do
echo $i # peach, tree, apple, tree
      # each in a new line
done
```

```
# everything together
echo "$*" usage in for cycle'
for i in "$*"
do
echo $i # result in one line!
      # parameters are between """
done
echo "$@" usage'
# the under mentioned line is
# the short version of: for i in $@
for i
do
echo $i
done
```

Execution of param example

```
illes@panda:~$ param grass tree 'apple tree'
```

Usage of parameters

We are using a compound parameter too ('apple tree')

```
$* usage
```

```
grass
```

```
tree
```

```
apple
```

```
tree
```

```
"$*" usage in for cycle
```

```
grass tree apple tree
```

```
"$@" usage
```

```
grass
```

```
tree
```

```
apple tree
```

```
illes@panda:~$
```

More command parameters

- Review:
 - \$0 – command name
 - \$1,...\$9 parameters
- Only 9 parameters can be?
- No, we can use the {} symbols!
- It is OK, but great number of parameters are rather rarely!

Usage of great number of parameters

- `${10}`
 - The tenth parameter, without parantheses it is `$10`, and it would be a concatenation (0 character is concatenated to `$1`)!
 - `${100}` # 100. parameter...
- Shift instruction
 - It shifts the parameters to the left with one.
 - `$1` will be thrown away and the other parameters steps to the left with one: `$2->$1,...,$10->$9` (if there is `$11`, then `$10` gets its value)
 - Typical processing with the help of cycles.

Other useful variables

- \$\$ - The PID number of the running program!
- \$! – The PID number of the last executed background process!
- \$- - The shell options . ??????

The termination and result of the program

- A shell script after finishing the execution of the last instruction ends its working and goes back to the calling one, into the shell!
- Is there a result of it?
 - Yes, there is!
- Each instructions have got a result!
 - This result shows the succes or failure of the program!
- A program runs succesfully if it can execute its task regularly!
 - It differs at each program!

The result of commands

- Such result of command comes into being in each case!
- The result of the command is in \$? variable!
- echo hello! ; echo \$? # 0 will be the result of the echo command!
- If the result is 0, then the command ran down successfully!
- If the result is a positive number (mostly 1), the command was unsuccessful

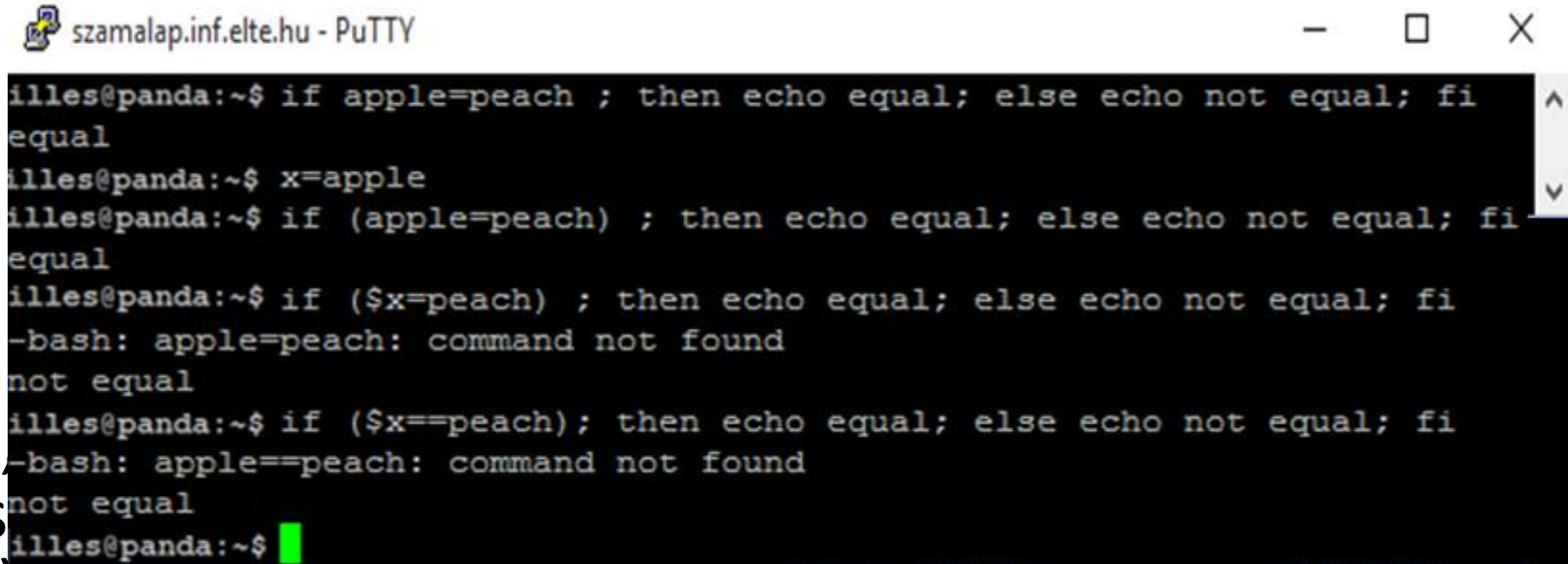
```
illes@panda:~$ cp frog peter
cp: cannot stat 'frog': No such file or directory
illes@panda:~$ echo $?
1
illes@panda:~$
```

exit command

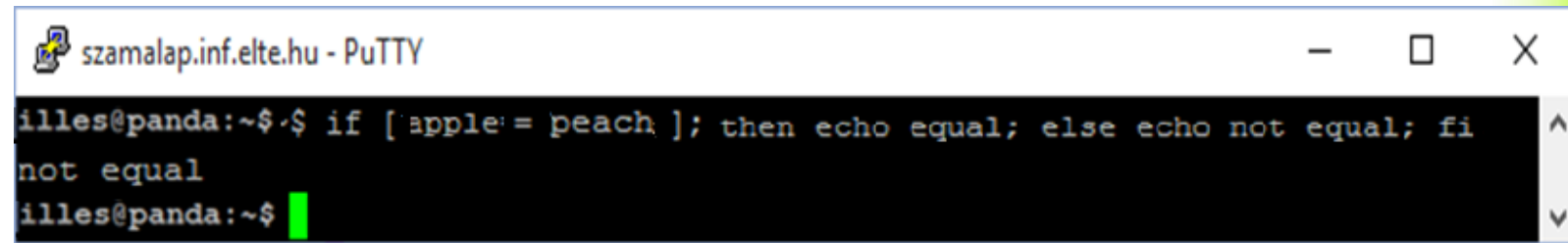
- Using exit command we can step out from a shell program!
- exit value # where value is an integer between 0-255
 - If the parameter of exit is 0, then it means a succesful program execution!
 - This success can be interpreted as logical true.
 - If the parameter is a positive number, the execution failures!
 - It can be interpreted as logical false!
- It is similar to C programming language, but the representation is the opposit!

Logical expressions

- Let us see the followings:
- There is no logical expression in itself!
- There is an assignment, it can be interpreted as a logical one too! (true)
- Only [] logical expression is good!



```
szamalap.inf.elte.hu - PuTTY
illes@panda:~$ if apple=peach ; then echo equal; else echo not equal; fi
equal
illes@panda:~$ x=apple
illes@panda:~$ if (apple=peach) ; then echo equal; else echo not equal; fi
equal
illes@panda:~$ if ($x=peach) ; then echo equal; else echo not equal; fi
-bash: apple=peach: command not found
not equal
illes@panda:~$ if ($x==peach); then echo equal; else echo not equal; fi
-bash: apple==peach: command not found
not equal
illes@panda:~$
```



```
szamalap.inf.elte.hu - PuTTY
illes@panda:~$ if [apple=peach]; then echo equal; else echo not equal; fi
not equal
illes@panda:~$
```

Logical value as the result of an instruction

- There is no logical operation, as we have seen!
 - The result of each command can be interpreted as a logical value too!!!!
- **test** instruction helps! This instruction can be replaced by [] symbols!
- test operand1 operator operand2 # the space is important!
 - or: [operand1 operator operand2] # the space is important here too, before] and after [!

Test – arithmetic operations

- test, or [...] logical examination
- 0 – true, 1- false, echo \$?
- true – true command, exit 0
- False – false command, exit 1
 - -lt,-gt,-le,-ge,-eq,-ne numerical examination
 - [\$x -lt 5]
 - -f file, -d dir file or directory existing
 - -o, or, -a and operator, ! negation
 - -r,-w,-x testing file permissions read, write, execution

Test – string, file examination

- Comparing strings:
 - =, != equal, not equal string comparison (exclamation point and equal sign as in C or C++)
 - test \$a = \$b # true, if \$a and \$b are equal strings
 - test \$a != \$b # true, if \$a and \$b are not equal strings
 - test -z \$X # true, if the length of \$X string = 0, it is an empty string
 - test -n \$X # true, if the length of \$X string is not 0, not an empty
- Full reference: man test

Test sample – I.

- This sample shows the most frequently used features!
- Simple logical expressions.

```
$ x=4
$ [ $x -lt 6 ] # test $x -lt 6
$ echo $?
0 (true)
$ test "alma" = "körte" # [ „alma” = „körte” ]
$ # text equation result
$ echo $?
$ 1 (false)
$ test -z „alma” # empty string check
$ echo $?
1
```

Test példa – II.

- Compound logical check
- Fájl, könyvtár vizsgálat

```
$ x=4
```

```
$ y=fradi
```

```
$ [ $x -eq 4 -a $y != "vasas" ]
```

```
$ # logical AND relation (-a)
```

```
$ echo $?
```

```
0 (true)
```

```
$ [ ! $x -eq 4 ]
```

```
$ echo $?
```

```
1 (false)
```

```
$ test -f fradi # true, if file fradi is exist
```

```
$ [ -d alma ] # true, if directory alma is exist
```



Thank you!