

OPERATING SYSTEMS

– Laboratory 3 –

SHELL PROGRAMMING

1. INTRODUCTION

- shell = a special program/language that provides an interface between user and the operating system kernel
- shell types: *sh* (*Bourne shell*), *csh* (*C shell*), *ksh* (*Korn shell*), *bash* (*GNU Bourne-again shell*)
- script = a text file that contains commands (internal or external)
- bash script example:

```
#!/bin/bash

pwd
ls
```

Important:

– the characters `#!` On the first line of the script is NOT a simple comment (it's called *shebang*) and after it must follow the absolute path to the program that must be run for the next lines of the script, in this case `/bin/bash`

- to run a bash script we must add execution permissions first, then run with `./scriptname`:

```
chmod +x script_1.sh
./script_1.sh
```

- comments start with `#` (*hash*)
- variables:
 - variable names can contain letters, digits and „`_`” (*underscore*), first character must be a letter, reserved words can not be used as var names

– all is case sensitive (distinction between uppercase and lowercase letters)

– examples:

```
n=45
name=Ana
msg="Enter a number:"
```

- reserved words (*keywords*):

```
if then else elif fi
for while until do done
case in esac
```

- internal commands (*built-in commands*):

– print the list of internal commands: `help`

– print information about a command: `help command`

– examples: `echo` `read` `printf` `test`

- bash scripts:

- read and print a number: script_2.sh

```
#!/bin/bash

echo "Enter a number:"
read n

echo "The number was: $n"
```

- read and print a string: script_3.sh

```
#!/bin/bash

echo -n "Enter your name: "
read name

echo "Hello" $name
```

- special variables:

\$0	Name of the script file
\$1, ..., \$9	Command line arguments given for execution
\$#	Number of command line arguments given
\$*	Array of command line arguments
@	List of individual command line arguments
\$?	Exit code (exit status) of the last executed command
\$\$	PID-ul of the current process
\$_	PID-ul of the last command launched in background

- example: script_4.sh

```
#!/bin/bash

echo "Filename: $0"
echo "First argument: $1"
echo "Second argument: $2"
echo "Argument count: $#"
```

Use `chmod +x script_4.sh` to add permissions, then run with various arguments:
`./script_4.sh 1 2 3 string word "hello everyone"`

2. ARITHMETICAL EXPRESSIONS WITH INTEGER NUMBERS

- a shell variable is implicitly considered a string
- example: script_5.sh

```
#!/bin/bash

A=1234
B=5678

echo "$A + $B"
```

2.a. Command `expr`

`expr` *expression*

- evaluates and prints at standard output the value of an arithmetical integer expression
- operators:

<code>+ - * / // %</code>	Sum, diff, mult, div , mod
<code>= !=</code>	Numerical comparisons:
<code>\> \>=</code>	– return 1 if the relation between s and d is true
<code>\< \<=</code>	– else returns 0
<code>\(\)</code>	For subexpressions (parenthesis)
<code>S \ D</code>	return S if S is not NULL and not 0, return D otherwise
<code>S \& D</code>	return S if both S and D are not NULL and not 0, 0 otherwise
<code>length S</code>	Length of S
<code>index S CHARS</code>	Position of the first occurrence in S or 0 (index starts at 1)
<code>substr S P L</code>	Substring starting with S on position P and length L

2.b. Command `let`

- evaluate and print at standard output the value of an integer expression
- operators: `++ -- ! ~ ** * / % + - << >> <= >= < > == != & ^ | && ||`

2.c. Double parenthesis

- example: `script_6.sh`

```
#!/bin/bash

# Doing simple math in Bash
# Shell script variables are by default treated as strings, not numbers.

# read the first number
echo -n "First number: "
read num1

# read the second number
echo -n "Second number: "
read num2

# 1. Using compound command (( expression ))
sum=$((num1+num2))

# 2. Using let built-in command
#let sum=num1+num2

# 3. Using expr extern command
sum=`expr $num1 + $num2`      # must put spaces around + sign

# 4. Using declare built-in command
#declare -i sum
#sum=num1+num2

echo "The result is:" $sum
```

3. COMMAND `test`

- syntax:

```
test condition          or          [ condition ]
```

- evaluate *condition* and return 0 if true, otherwise a nonzero value
- allow string/integer comparisons, and file options checking

3.a. Compare integers

- relational operators: `-lt -le -eq -ne -ge -gt`
- AND /OR / NOT: `-a, -o , ! ,`
- **see** `man test`

3.b. String comparison

```
-n STRING
    the length of STRING is nonzero

STRING equivalent to -n STRING

-z STRING
    the length of STRING is zero

STRING1 = STRING2
    the strings are equal

STRING1 != STRING2
    the strings are not equal
```

3.c. File checking options

```
-b FILE
    FILE exists and is block special

-c FILE
    FILE exists and is character special

-d FILE
    FILE exists and is a directory

-e FILE
    FILE exists

-f FILE
    FILE exists and is a regular file

-g FILE
    FILE exists and is set-group-ID

-G FILE
    FILE exists and is owned by the effective group ID

-h FILE
    FILE exists and is a symbolic link (same as -L)

-k FILE
    FILE exists and has its sticky bit set

-L FILE
    FILE exists and is a symbolic link (same as -h)

-O FILE
    FILE exists and is owned by the effective user ID

-p FILE
    FILE exists and is a named pipe

-r FILE
    FILE exists and read permission is granted

-s FILE
    FILE exists and has a size greater than zero
```

```
-w FILE
    FILE exists and write permission is granted

-x FILE
    FILE exists and execute (or search) permission is granted
```

4. IF/THEN/ELIF/ELSE/FI

- syntax:

```
if condition
then
    statement(s) to be executed
elif condition
then
    statement(s) to be executed
elif condition; then
    statement(s) to be executed
else
    statement(s) to be executed
fi
```

- examples: if_1.sh, if_2.sh

```
#!/bin/bash
for A in $@
do
    if test -f $A
    then
        echo $A is a file
    elif test -d $A
    then
        echo $A is a dir
    elif echo $A | grep -q "^[0-9]\+$"; then
        echo $A is a number
    else
        echo We do not know what $A is
    fi
done
```

```
#!/bin/bash
for A in $@
do
    if [ -f $A ]
    then
        echo $A is a file
    elif [ -d $A ]
    then
        echo $A is a dir
    elif echo $A | grep -q "^[0-9]\+$"
    then
        echo $A is a number
    else
        echo We do not know what $A is
    fi
done
```

5. FOR/DO/DONE

- syntax:

```
for var in list
do
    statement(s) to be executed
done
```

- examples: for_1.sh, for_2.sh, for_3.sh, for_4.sh, for_5.sh

```
#!/bin/bash

for N in 1 2 3 4 5
do
    echo $N
done
```

```
#!/bin/bash

for N in {0..5}
do
    echo $N
done
```

```
#!/bin/bash

NAMES='Ana Iulia Maria Tudor'

for N in $NAMES
do
    echo $N
done
```

```
#!/bin/sh

if [ $# -eq 0 ]
then
    echo "Error: Not enough arguments."
    exit 1
fi

for arg in $*
do
    echo $arg
done
```

```
#!/bin/bash

# Count all the lines of code of the C files
# in the directory given as command line argument
# and its subdirectories, excluding lines that are
# empty or contain only blank spaces

S=0
for F in `find $1 -type f -name "*.c"`
do
    N=`grep "[^ \t]" $F | wc -L`
    S=`expr $S + $N`
done

echo $S
```

- specify a pattern for a file name (*filename wildcards*)
 - * Any sequence of characters, even empty (except first dot - beginning of file)
 - ? One character (except first dot at the beginning)
 - [abc] Any character in the list between []
 - [!abc] Any character not in the list between [!]
- example:
 - print files with a name that starts with a letter and have extension of two letters:

```
ls [a-zA-Z]*.??
```

6. WHILE/DO/DONE, UNTIL/DO/DONE

- syntax:

while <i>condition</i>	until <i>condition</i>
do	do
statement(s) to be executed	statement(s) to be executed
done	done

- examples: while_1.sh, while_2.sh, while_3.sh, while_4.sh, while_5.sh

```
#!/bin/bash

# Read user input until the input is stop
# The command read stores the user input in
# the variable given as argument

while true
do
    read X
    if test "$X" == "stop"
    then
        break
    fi
done
```

```
#!/bin/bash

# Find all the files in the directory given
# as first command line argument, larger
# in size than the second command line argument

D=$1
S=$2

find $D -type f | while read F
do
    N=`ls -l $F | awk '{print $5}'`
    if test $N -gt $S
    then
        echo $F
    fi
done
```

```
#!/bin/bash

# Read the console input until the user
# provides a filename that exists and can be read

F=""
while [ -z "$F" ] || [ ! -f "$F" ] || [ ! -r "$F" ]
do
    read -p "Provide an existing and readable file path:" F
done
```

```
#!/bin/bash

if [ $# -eq 0 ]
then
    echo "Error: You must enter a filename."
    exit 1
elif !(test -r $1)
then
    echo "File $1 does not exist."
    exit 1
fi

while read -r line
do
    echo $line
done <$1
```

```
#!/bin/bash

echo -n "Enter a filename: "
read filename

while read -r line
do
    echo $line
done < $filename
```

7. CASE/ESAC

- syntax:

```
case var in
    pattern_1)
        statement(s) to be executed if pattern_1 is matched;;
    pattern_2)
        statement(s) to be executed if pattern_2 is matched;;
    ...
    *)
        default condition to be executed;;
esac
```

- examples: case_1.sh

```
#!/bin/bash

if [ $# -eq 0 ]
then
    echo "Error: You must enter an option."
    exit 1
fi

case $1 in
    -[a-z] ) echo "You enter a letter";;
    -[0-9] ) echo "You enter a digit";;
    * ) echo "Unknown option";;
esac
```

You can use C like syntax expressions with (()), example: `if (($num <= 5 && $a > 9)) ...` or `for ((i=2; %i<=$N; i++)) ; do ...` or `f=$((f*$i)) ...`

8. OTHER USEFUL COMMANDS

8.a. Command `cut`

```
cut -d: -f 1 /etc/passwd
cut -d ":" -f 5 /etc/passwd
who | cut -d " " -f 1,11
```

8.b. Command `find`

```
find . -type f -name "*.sh"
find /tmp -type d -empty
```

8.c. Command `shift`

```
shift [n]  shift left with n positions the arguments given in command line
shift 2    (first two command line arguments are deleted)
```

8.d. Command `sleep`

```
sleep [n]  suspend execution for n seconds
```

8.e. Command `exit`

```
exit [n]  terminate execution and return to the process that launched it
exit 0 - SUCCESS ; exit 1 - ERROR CODE 1
```

8.f. Command `read`

```
read var
read -p "Give a number: " n # read with prompt
```

– to read from a file line by line we need to change IFS internal field separator variable. See more in `man read`; Example:

```
#!/bin/bash

IFS="" # to read to the EOL, field separator is made empty
while read line
do
    echo $line
    #make for to parse word by word, change separator to space
    IFS=" "
    for w in $line
    do
        echo "[$w] "
    done
    #back to reading line by line, to keep spaces change IFS
    IFS=""
done < $1
#$1 is input file, must be provided as command line argument
```

REFERENCES:

- Shell programming: <https://ryanstutorials.net/bash-scripting-tutorial/>

SHELL 1 Assignment problems (Deadline week 5)

Lab 3 - Homework

No.	Assignment
1	Write a shell script which takes as parameters two integers. The script will display the sum, the difference and the product of the two given integers.
2	Write a shell script which takes as parameters two names of text files. The script will compare the two text files line by line and display the first 3 text lines that differ.
3	Write a shell script which takes as parameter a natural number. The script will check whether the given number is prime or not.
4	Write a shell script that reads numbers from keyboard (the reading stops when the user has entered the number 0). The script will display the sum of numbers entered from the keyboard.
5	Write a shell script that reads user names from keyboard. For each user, the script will display the number of times it was logged in to the server in the current month. If he/she has not logged in at all during the current month, the script will display the message: <i>"User X has never logged in during the current month"</i> .
6	Write a shell script which displays the full username and IP address of each user currently logged on to the server.
7	Write a shell script which takes as parameter a directory name. The script will determine the total number of lines in all ASCII text files in this directory and its subdirectories. It is assumed that any directory will only contain ASCII text files.
8	Write a shell script which takes as parameters two file names (a file that contains user names and a file that contains any text). The script will send a mail to each user in the first file (the mail message will be the text in the second file).
9	Write a shell script which takes as parameter an username. The script will count and display the number of processes that belong to that user.
10	Write a shell script which takes as parameter an existing group number (ex: 821). The script will display the user name, the full name and the home directory of each student in that group.
11	Write a shell script which lists the content of the current directory. The script will list the files as follows: sorted by file names, sorted by last modified date and sorted by file size.
12	Write a shell script which takes as parameter a directory name. The script will display the filename and the first 3 lines of each ASCII text file found in that directory.
13	Write a shell script which takes as parameter a username (ex: bdae0198). The script will determine the full name of that user and if he/she is currently connected to the server.
14	Write a shell script which displays all files in the current directory and its subdirectories that have write permission for the group of which the owner belongs.
15	Write a shell script that reads words from keyboard (the reading stops when the user has entered the word "stop"). The script will display the list of words entered from the keyboard.