Computer Networks - laboratory exercise:

Bash shell programming

symbol or command	**description**
scriptName.sh	save your scripts with **.sh** extensions
chmod 775 scriptName.sh	set execution permissions for the script, eg. 775, 750 or whatever you want which will let you run the script
./scriptName.sh	execute your script from the command line (if it was saved in working directory)
#!	"sha-bang" – indicates that this file is a script
#!/bin/bash	should be the 1st line of a script; indicates which interpreter to use to execute it
#	beginning of a comment (the comment ends at the end of line)
;	separator between commands if you want to put a few of them in a single line
:	null command (*nop*, no operation)
{ }	a block of code (can have local variables)
"	(double quote) partial/weak quoting – preserves most special characters in the string from being interpreted (except **$ ` **); merges several words to one string; preserves whitespaces in **echo** command
'	(single quote) full/strong quoting – preserves all special characters in the string from being interpreted
\	escape – like quoting a single character; eg. **\"** or **\'** or **** or **\$** lets you print **"** or **'** or **** or **$** literally in a string
\n \t etc.	newline, tab and other special characters – works with **sed**, **echo -e, printf** commands
`	(backquote) command substitution, ie. assignment of command output to a variable, eg. **var=`ls -l`**
var=$(ls -l)	same as above
echo printf	Display text; **echo** automatically appends a newline at the end

read var	read user input and put it into variable **var** (creates or overwrites the variable)
var=5 var=text var="a few words"	• variable assignment (and possibly creation); no spaces next to **=** character! • bash variables are untyped
var=(1 3 8 2)	a variable which is a list (an array)
declare -i var declare -a var declare -r var=15	declares that **var** is: an integer, an array, read-only (a constant)
typeset	identical to **declare**
let "var=010" let "var=0x10" let "var=2#10" let "var=3#10"	assign to a variable: an octal value, a hexadecimal value, a binary value a trinary value etc.
let "var += 4"	perform arithmetic operations on variable **var**
((var = 15)) ((var++)) ((var += 4)) a=$((b/2))	double parentheses allow for C-style variable assignment (with spaces) and C-style arithmetics
var= var='' var=""	setting a variable to a null value, ie. no value (not zero!)
unset var	deletes a variable, in effect – similar as above
$var	variable substitution (dereferencing) – gets the value of variable **var**
${var}	similar as above
"$var"	usually it is best to add double quotes to prevent word splitting
${array[*]}	all elements of an array
${array[2]}	element number 2 in an array (indexing is zero-based)
$#	number of arguments passed to the script from the command line

$0	name of the script
$1 $2 $3 ... ${10} ${11} ...	positional parameters (arguments passed to the script from the command line): first, second, third, ... tenth, eleventh, ...
$*	all positional parameters as a single string; usually it's best to use quoted version "$*"
$@	all positional parameters as a list; sometimes it's good to use quoted version "$@"
$$	process ID of the script
shift	shifts (moves) positional parameters one position to the left ($1 gets lost, old $2 becomes $1, etc.); $0 remains unchanged
shift **3**	shifts positional parameters by **3** positions
$RANDOM	generates a pseudorandom number in range 0-32767
$?	exit status of the previous command
exit 0 exit 1 exit $CODE exit $?	finish the script and return to the shell: • 0=true=success, • error exit code 1, • exit code stored in a variable, • exit status of last command
case "$var" in ab) *commands1* ;; c) *commands2* ;; 13) *commands3* ;; *) *defaultCmds* ;; esac	• compare a variable to multiple values and execute the proper set of commands • the values to compare to may be constant numbers (eg. **13**) or constant strings (eg. **xyzzz**, **"a and b"**), or values of variables (eg. **"$var"**) • they may also contain ranges (eg. **[0-9]**, **[a-f]**) • the default action ***)** is optional
if *test1* then ... elif *test2* then ... else ... fi	(see **man test**) *test1*, *test2* might be: • test "$var" *(does var have a value? (not null))* • test -n "$string" *(non-empty string?)* • test -z "$string" *(zero-length string?)* • test -e myFile.txt *(does myFile.txt exist?)* • test $int1 -gt $int2 *(is int1 greater than int2?)* • [$int1 -lt $int2] • [$int1 \< $int2] • ["$string1" = "$string2"] *(mind the quotes and the spaces next to =)*
	• [$var] • [[$int1 -eq $int2]] • [[($int1 > $int2) && ($int3 != $int4)]] • ((var1 > var2)) *(double parentheses are for C-style writing, hence no $)* • etc.
for var in *list* do *commands* done	*list* might be: • 1 3 8 17 4 5 • "$var1" "$var2" "$var3" • ${arrayVariable[*]} • /home/* *(wildcards permitted)* • $(find *aDirectory* -name '*.png' | sort) *(command substitution)* • `seq 10` *(1, 2, 3, ..., 10)* • {1..10} *(1, 2, 3, ..., 10)*
LIM=10 for ((a=1; a <= LIM ; a++)) do *commands* done	C-style for loop; note the double parentheses and no $ by variable name LIM
while [*cond*] do *commands* done	condition is similar to the one in **if** statement

function f
{
function body
return 0
}
	• to call a function, use **f** or **f arg1 arg2**, if you want to pass arguments to it
	• to access the arguments inside the function body, use **$1 $2** ...
	• **return *exitCode*** is optional

f()
{
function body
return 0
}
	same as above