

Shell Scripting

Spring 2022

Shell Scripting

- ▶ A sequence of system programs carrying out a specific task
- ▶ The simplest script is:

```
#!/bin/bash  
echo Hello World;  
ls ~/ |fmt -w 40
```

- ▶ “#” indicates a comment
- ▶ first line says which shell is to be used (here is bash)
- ▶ Can do complicated things effectively

```
#!/bin/bash  
# tar-ring and compress-ing together  
tar -z -cf /tmp/my-backup.tgz /home/asimina/
```

Creating Shell Scripts

- ▶ Parameters to scripts are designated
- ▶ Variables and Conditions are used
- ▶ Basic Control Statements (loops `for`, `while` and `until`)
- ▶ Numeric and alphanumeric operations
- ▶ Functions and pipes

A small Script About Input Parameters (\$n, \$*)

```
#!/bin/bash
# all scripts start like this

#will give 11 arguments to this program
# a b c d e f g h i j k

echo Number of input parameters = $#      # 11
echo Program Name = $0                    # ./parameters

echo Other Parameters = $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11
#Other Parameters = a b c d e f g h i a0 a1

echo Other Parameters = $1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11}
#Other Parameters = a b c d e f g h i j k

echo All Arguments = $*
#All Arguments = a b c d e f g h i j k
```

◇ Output:

```
antoulas@sazerac:~/bash-scripts$ ./parameters a b c d e f g h i j k
Number of input parameters = 11
Program Name = ./parameters
Other Parameters = a b c d e f g h i a0 a1
Other Parameters = a b c d e f g h i j k
All Arguments = a b c d e f g h i j k
antoulas@sazerac:~/bash-scripts$
```

Using variables — *read*-ing from the shell

```
#!/bin/bash
# Never place a '$' ahead of a variable when the latter gets
#   assigned a value!

# Never use spaces immediately before or after = in assignments
a=2334          # Integer - Only digits
echo a         # a
echo $a        # 2334

hello="A B C   D"
echo $hello    # A B C D
echo "$hello"  # A B C D
# Double quotes help retain multiple spaces

echo '$hello'  # $hello
# Single quotes de-activate reference to the value indicated by $
echo -n "Enter \"b\" "      # Grafw kati...
read b
echo "The value of \"b\" is now $b"
# The value of "b" is now lala koko
echo ${PATH}   # Environment Variable PATH
```

```
antoulas@sazerac:~/bash-scripts$ ./variables
a
2334
A B C D
A B C   D
$hello
Enter "b" alxntoulas
The value of "b" is now alxntoulas
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
antoulas@sazerac:~/bash-scripts$
```

Some Arithmetic Operations

```
#!/bin/bash
a=2334
let b=a+3 # b=$a+3 also works
let "c = a+3"
let "d = a + 3"

z=$((a+13))
y=$((a+23)) # Also works

k='expr $a + 33' # use of expr command in bash

echo $a $b $c $d $k $z $y
#2334 2337 2337 2337 2367 2347 2357
```

- ▶ For simple integer operations use *let* and *expr*
- ▶ For decimal arithmetic use the system program *bc*

```
antoulas@sazerac:~/bash-scripts$ ./arithmetics
2334 2337 2337 2337 2367 2347 2357
antoulas@sazerac:~/bash-scripts$
```

More Arithmetic

```
#!/bin/bash

# Spaces are essential below!
a='expr 3 + 5'; echo $a      # 8
a='expr 5 % 3'; echo $a     # 2
a='expr 5 / 3'; echo $a     # 1
# a='expr 1 / 0' # Returns a error value
a='expr 5 \* 3'; echo $a    # 15
# need to escape *, as otherwise script goes to the shell
a='expr $a + 5'; echo $a # let a=a+5 also works!

string=EnaMegaloString
echo "String is: ${string}"
position=4
length=6
z='expr substr $string $position $length'
#Extracts length number of characters from string.
#Starts off from position

echo "Substring is: $z" # Megalo
```

◇ Execution:

```
antoulas@sazerac:~/bash-scripts$ ./myexpr
8
2
1
15
20
String is: EnaMegaloString
Substring is: Megalo
antoulas@sazerac:~/bash-scripts$
```

An interesting system program: *bc*

- ◇ A general and versatile purpose calculator

```
antoulas@sazerac:~/bash-scripts$ bc
bc 1.06.94
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
1
1
0
0
1 > 0
1
0 > 1
0
12 > 8
1
8 > 12
0
123^23
1169008215014432917465348578887506800769541157267
quit
antoulas@sazerac:~/bash-scripts$
```

Carrying out decimal arithmetic in *bash*

```
#!/bin/bash
# Allows arithmetic on decimals
a=100.19
b=$(echo "scale=3; $a/100" | bc)
# scale determines decimal digits in fractional part

echo b = $b # b = 1.001

#perform inequality tests
A=0.04
B=0.03
let "comp='echo $A-$B\>0 | bc'"
echo $comp # 1

let "comp='echo $B-$A\>0 | bc'"
echo $comp # 0
```

◇ Execution:

```
antoulas@sazerac:~/bash-scripts$ ./mybc
b = 1.001
1
0
antoulas@sazerac:~/bash-scripts$
```

Getting the Return Value of a Program

```
#!/bin/bash

# $? returns the exit code of the last command to execute

echo hello
echo $?      # 0 : successful completion

lsdlsd      # unknown program
echo $?     # 127 - nonzero for an error

echo Hello

exit 113    # Must be in range: 0-255
echo $?
```

- Output:

```
antoulas@sazerac:~/bash-scripts$ ./exitStatus
hello
0
./exitStatus: line 8: lsdlsd: command not found
127
Gia sou
antoulas@sazerac:~/bash-scripts$ echo $?
113
antoulas@sazerac:~/bash-scripts$
```

More on return Values

- Assume that “dada” does not exist”

```
#!/bin/bash
cd /dada >& /dev/null
echo rv: $?
cd $(pwd) >& /dev/null
echo rv: $?
```

- Output

```
d@cairns:~/bash-scripts$ ./myreturn
rv: 1
rv: 0
antoulas@sazerac:~/bash-scripts$
```

An interesting program: *bc* – a versatile calculator

```
antoulas@sazerac:~/Samples$ bc
bc 1.06.94
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free
  Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
12+23
35
45-456
-411
34+ 67/2
67
34+(67/2)
67
67/2
33
8%3
2
24.5^35
4176504491184268678934402890639352604632655383880.6
antoulas@sazerac:~/Samples$
```

bc: working with different scales

```
antoulas@sazerac:~/Samples$ bc
bc 1.06.94
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free
  Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
21/2
10
scale=4
21/2
10.5000
scale=8
193/32.23456
5.98736263
19/3
6.33333333
scale=0
19/3
6
antoulas@sazerac:~/Samples$
```

bc: working with the binary input base (*ibase*)

```
antoulas@sazerac:~/Samples$ bc
bc 1.06.94
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free
  Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
ibase=16
1A
26
10 * 10
256
ibase=8
10
8
10 * 11
72
ibase=2
1111
15
111 * 111
49
antoulas@sazerac:~/Samples$
```

bc: using different output base (*obase*)

```
antoulas@sazerac:~/Samples$ bc
bc 1.06.94
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free
  Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
obase=2
5
101
15/3
101
obase=8
9
11
99/10
11
obase=16
26
1A
256
100
16 * 16
100
antoulas@sazerac:~/Samples$
```

Conditionals

- ▶ Conditionals let you decide whether to perform an action.
- ▶ The decision above is taken by evaluating an expression.
- ▶ Conditions are of the form [...]; for example:

```
[ "foo" = "foo" ]
```

- ▶ We may have arithmetic conditions such as:

```
'2>1'
```

which evaluates to TRUE.

- ▶ The construct (()) evaluates numerical expressions **and returns exit code**:
 - ▶ 0 or TRUE when the value inside the parentheses (()) evaluates to *non-zero*
 - ▶ 1 or FALSE when the value inside the parentheses (()) evaluates to *zero*
 - ▶ Opposite from C convention! – Think of it as translating C values to Unix exit code for success!

Arithmetic Tests

```
#!/bin/bash
# Arithmetic tests. The (( ... )) construct evaluates and tests
# numerical expressions.

(( 0 ))
echo "Exit status of \"(( 0 ))\" is $?." # 1
(( 1 ))
echo "Exit status of \"(( 1 ))\" is $?." # 0
(( 5 > 4 )) # true
echo "Exit status of \"(( 5 > 4 ))\" is $?." # 0
(( 5 > 9 )) # false
echo "Exit status of \"(( 5 > 9 ))\" is $?." # 1
(( 5 - 5 )) # 0
echo "Exit status of \"(( 5 - 5 ))\" is $?." # 1
(( 5 / 4 )) # Division o.k.
echo "Exit status of \"(( 5 / 4 ))\" is $?." # 0
(( 1 / 2 )) # Division result < 1.
echo "Exit status of \"(( 1 / 2 ))\" is $?."
# Division is rounded off to 0.
# 1
(( 1 / 0 )) 2>/dev/null # Illegal division by 0.
#
echo "Exit status of \"(( 1 / 0 ))\" is $?." # 1
# What effect does the "2>/dev/null" have?
# What would happen if it were removed?
# Try removing it, then rerunning the script.
exit 0
```

Output

```
antoulas@sazerac:~/bash-scripts$ ./arithmeticTests
Exit status of "(( 0 ))" is 1.
Exit status of "(( 1 ))" is 0.
Exit status of "(( 5 > 4 ))" is 0.
Exit status of "(( 5 > 9 ))" is 1.
Exit status of "(( 5 - 5 ))" is 1.
Exit status of "(( 5 / 4 ))" is 0.
Exit status of "(( 1 / 2 ))" is 1.
Exit status of "(( 1 / 0 ))" is 1.
antoulas@sazerac:~/bash-scripts$
```

Checking Files/Directories with flags `-e`, `-d`, `-r`

```
#!/bin/bash

if [ -e $1 ]      # exists file
  then if [ -f $1 ] # is a regular file
    then echo Regular File
    fi
  fi
# flag -d checks if it's a directory

if [ -r $1 ]      # have read rights
  then echo I can read this file!!!
fi
# Omoia to -w kai -x
```

◇ checking files - output

```
antoulas@sazerac:~/bash-scripts$ ./fileTests fileTests
Regular File
I can read this file!!!
antoulas@sazerac:~/bash-scripts$ ./fileTests /tmp/hhh
antoulas@sazerac:~/bash-scripts$
```

Forming Conditions with *Integers*

<code>-eq</code> <code>if ["\$a" --eq "\$b"]</code>	<code>equal</code> <code>(("\$a" = "\$b"))</code>
<code>-ne</code> <code>if ["\$a" --ne "\$b"]</code>	<code>not-equal</code> <code>(("\$a" <> "\$b"))</code>
<code>-gt</code> <code>if ["\$a" --gr "\$b"]</code>	<code>greater than</code> <code>(("\$a" > "\$b"))</code>
<code>-lt</code> <code>if ["\$a" --lt "\$b"]</code>	<code>less than</code> <code>(("\$a" < "\$b"))</code>
<code>-le</code> <code>if ["\$a" --le "\$b"]</code>	<code>less or equal</code> <code>(("\$a" <= "\$b"))</code>

Creating Conditions involving *Strings*

- always use quotes
- even more confusing: the spaces in [...] are important!

<code>=</code> <code>if ["\$a" = "\$b"]</code>	equal
<code>==</code> <code>if ["\$a" == "\$b"]</code>	equal
<code>!=</code> <code>if ["\$a" != "\$b"]</code>	not-equal
<code><</code> <code>if ["\$a" \<< "\$b"]</code>	alphanumerically less
<code>></code> <code>if ["\$a" \> "\$b"]</code>	alphanumerically greater
<code>-n</code> <code>if [-n "a"]</code>	not-null
<code>-z</code> <code>if [-z "a"]</code>	Null (size 0)

Creating Conditions involving *Expressions*

<pre>! if [! "\$a"]</pre>	Logical NOT
<pre>-a if ["\$a" -a "\$b"]</pre>	Logical AND
<pre>-o if ["\$a" -o "\$b"]</pre>	Logical OR

The if then; elif; else fi; control statement

```
if [expression1];
    then statement1
elif [expression2];
    then statement2
elif [expression3];
    then statement3
else
    statement4
fi
```

- The sections “else if” and “else” are optional.

```
#!/bin/bash

T1="foo"
T2="bar"

if [ "$T1" = "$T2" ]; then
    echo expression evaluated as true
else
    echo expression evaluated as false
fi
```

The case control statement

```
case $variable in
$condition1)
    statements1;;
$condition2)
    statements2;;
$condition3)
    statements3;;
    ....
esac
```

An example:

```
echo -n "Enter the name of an animal: "
read ANIMAL
echo -n "The $ANIMAL has "
case $ANIMAL in
    horse | dog | cat) echo -n "four";;
    man | kangaroo ) echo -n "two";;
    *) echo -n "an unknown number of";;
esac
echo " legs."
```