

---

# Shell Scripting

March 2017

# 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:

```
ad@ad-desktop:~/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
ad@ad-desktop:~/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
```

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

## Some Arithmetic Operations

```
#!/bin/bash
a=2334
let b=a+3 # b=$a+3 aslo 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*

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

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

## An interesting system program: *bc*

- ◇ A general and versatile purpose calculator

```
ad@ad-desktop:~/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
ad@ad-desktop:~/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:

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

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

## 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

```
ad@cairns:~/bash-scripts$ ./myreturn
rv: 1
rv: 0
ad@cairns:~/bash-scripts$
```

## An interesting program: *bc* – a versatile calculator

```
ad@ad-desktop:~/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
ad@ad-desktop:~/Samples$
```

## bc: working with different scales

```
ad@ad-desktop:~/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
ad@ad-desktop:~/Samples$
```

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

```
ad@ad-desktop:~/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
ad@ad-desktop:~/Samples$
```

## bc: using different output base (*obase*)

```
ad@ad-desktop:~/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
```

## 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

```
ad@cairns:~/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.
ad@cairns:~/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

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

## Forming Conditions with *Integers*

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

## Creating Conditions involving *Strings*

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

<pre>      = if [ "\$a" = "\$b" ]</pre>	equal
<pre>     == if [ "\$a" == "\$b" ]</pre>	equal
<pre>     != if [ "\$a" != "\$b" ]</pre>	not-equal
<pre>      &lt; if [ "\$a" \&lt;&lt; "\$b" ]</pre>	alphanumerically less
<pre>      &gt; if [ "\$a" \&gt; "\$b" ]</pre>	alphanumerically greater
<pre>     -n if [ -n "a" ]</pre>	not-null
<pre>     -z if [ -z "a" ]</pre>	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."
```



```
#!/bin/bash
# Usage: math n1 op n2
case $2 in
+)      echo "Addition requested."
        echo "$1 + $3 = `expr $1 + $3`" ;;
-)      echo "Substraction requested."
        echo "$1 - $3 = `expr $1 - $3`" ;;
\*)     echo "Multiplication requested."
        echo "$1 * $3 = `expr $1 \* $3`" ;;
/)      echo "Division requested."
        echo "$1 / $3 = `expr $1 / $3`" ;;
%)      echo "Modulo arithmetic requested."
        echo "$1 % $3 = `expr $1 % $3`" ;;
*)      echo "Unknown operation specified." ;;
esac
```

## Outcome:

```
ad@cairns:~/bash-scripts$ ./math
Unknown operation specified.
ad@cairns:~/bash-scripts$ ./math 34 - 56
Substraction requested.
34 - 56 = -22
ad@cairns:~/bash-scripts$ ./math 34 % 22
Modulo arithmetic requested.
34 % 22 = 12
ad@cairns:~/bash-scripts$ ./math 34 * 2
Unknown operation specified.
ad@cairns:~/bash-scripts$ ./math 34 \* 2
Multiplication requested.
34 * 2 = 68
ad@cairns:~/bash-scripts$ ./math 34 # 4
Unknown operation specified.
ad@cairns:~/bash-scripts$
```

# for Loops

```
#!/bin/bash

for koko in 1 2 3 4 5 do
    echo $koko
    # print in different lines
done

for koko in "1 2 3 4 5" do
    echo $koko
    # print in one line
done

NUMS="1 2 3 4 5"
for koko in $NUMS do
    echo $koko
    # print in different lines
done

for koko in `echo $NUMS` do
    echo $koko
    # print in different lines
done

LIMIT=8
# Double parentheses, LIMIT without $
for ((koko=1; koko <= LIMIT; koko++)) do
    echo $koko "loop me limit"
    # print in different lines
done
```

## ◇ Outcome:

```
ad@cairns:~/bash-scripts$ ./forLoops
1
2
3
4
5
1 2 3 4 5
1
2
3
4
5
1
2
3
4
5
1 loop me limit
2 loop me limit
3 loop me limit
4 loop me limit
5 loop me limit
6 loop me limit
7 loop me limit
8 loop me limit
ad@cairns:~/bash-scripts$
```

## One more example of for loop

```
#!/bin/bash

# Without a value list, it processes
# the program's parameter list (implicit var)
for koko
do
    echo -n $koko;
done
echo

#how to parse some arguments from $2 until the end
for j in ${*:2}
do
    echo -n $j;
done
echo

#$2 to $4 - start at position 2 and use 3 args
for j in ${*:2:3}
do
    echo -n $j
done
echo
```

```
ad@cairns:~/bash-scripts$ ./forLoops2 aa bb cc dd ee ff ggg uuu
aabbccddeeffggguuu
bbccddeeffggguuu
bbccdd
ad@cairns:~/bash-scripts$
```

## while [] do done loop

```
#!/bin/bash
LIMIT=19 # Upper limit
echo "Numbers 1 through 20 (but not 3 and 11)."  
a=0  
while [ $a -le "$LIMIT" ]  
do  
  a=$((a+1))  
  # Ignore Numbers: 3, 11  
  if [ "$a" -eq 3 ] || [ "$a" -eq 11 ]  
  then continue;  
  fi  
  echo -n "$a " # not executed for 3 and 11  
done  
echo  
a=0  
while [ "$a" -le "$LIMIT" ]  
do  
  a=$((a+1))  
  if [ "$a" -gt 2 ]  
  then break; # Skip entire rest of loop.  
  fi  
  echo -n "$a "  
done  
echo
```

```
ad@cairns:~/bash-scripts$ ./breakCont  
Numbers 1 through 20 (but not 3 and 11).  
1 2 4 5 6 7 8 9 10 12 13 14 15 16 17 18 19 20  
1 2  
ad@cairns:~/bash-scripts$
```

## More on while Loop

```
#!/bin/bash
var0=0
LIMIT=10

while [ "$var0" -lt "$LIMIT" ]
do
  echo -n "$var0 "
  var0='expr $var0 + 1'
  # var0=$((var0+1)) also works.
  # var0=$((var0 + 1)) also works.
  # let "var0 += 1" also works.
done
echo
exit 0
```

### ◇ Outcome:

```
ad@cairns:~/bash-scripts$ ./whileLoops
0 1 2 3 4 5 6 7 8 9
ad@cairns:~/bash-scripts$
```

## The command: `set $myvar`

```
#!/bin/bash

echo Input parameters = $#
myvar="one two three four five six"

#split based on blank chars
#assign to input parameters!!
set $myvar

echo Input parameters = $#
#Now prints 6

for koko
do
    echo $koko
done
```

### ● Outcome

```
ad@cairns:~/bash-scripts$ ./setProg
Input parameters = 0
Input parameters = 6
one
two
three
four
five
six
ad@cairns:~/bash-scripts$
```

## An shell--script that prints strings in reverse

```
#!/bin/bash
# Usage: revstrs [string1 [string2 ...]]
#
for str
do
  strlen='expr length "$str"'
  # Start from the end. Need to know length
  chind=$strlen
  while test $chind -gt 0
  do
    echo -n "'expr substr \"$str\" $chind 1'"
    chind='expr $chind - 1'
  done
  echo -n " --> "
  echo -n "$strlen"
  echo " character(s)."
done
```

```
ad@ad-desktop:~/bash-scripts$ ./revstrs system programming k24 operating systems
k22
metsys --> 6 character(s).
gnimmargorp --> 11 character(s).
42k --> 3 character(s).
gnitarepo --> 9 character(s).
smetsys --> 7 character(s).
22k --> 3 character(s).
ad@ad-desktop:~/bash-scripts$
```



## Listing of Regular Files

```
#!/bin/bash
OUTFILE=files.lst
dirName=${1-'pwd'}      # - declares a default value
                        # if no directory value gets submitted
echo "The name of the directory to work in:  ${dirName}"
echo "Regular files in directory ${dirName}" > $OUTFILE
# -type f means regular files
for file in "$( find $dirName -type f )"
do
    echo "$file"
done | sort >> "$OUTFILE"
#
# redirection of sorted output
```

### ◇ Outcome:

```
ad@cairns:~/bash-scripts/tmp$ ls
alex asoee delis papi uoa upatras
ad@cairns:~/bash-scripts/tmp$ cd ..
ad@cairns:~/bash-scripts$ ./listRegFiles tmp/
The name of the directory to work in:  tmp/
ad@cairns:~/bash-scripts$ cat files.lst
Regular files in directory tmp/
tmp/alex
tmp/asoee
tmp/delis
tmp/papi
tmp/uoa
tmp/upatras
ad@cairns:~/bash-scripts$
```

## Shifting parameters in a shell-script

```
#!/bin/bash
# call with > 5 arguments
echo "All args are = $*" ;
echo "Number of Parameters = $#"
```

*for str # prints OK even with change*

```
do
  echo "The value of the iterator is: ${str} "
  var=$1
  shift
  echo "var = $var and args = $#"
```

*done*

### ◇ Outcome:

```
ad@ad-desktop:~/bash-scripts$ ./shiftCommand ena dio tria tessera pente exi efta
All args are = ena dio tria tessera pente exi efta
The value of the iterator is: ena
var = ena and args = 6
The value of the iterator is: dio
var = dio and args = 5
The value of the iterator is: tria
var = tria and args = 4
The value of the iterator is: tessera
var = tessera and args = 3
The value of the iterator is: pente
var = pente and args = 2
The value of the iterator is: exi
var = exi and args = 1
The value of the iterator is: efta
var = efta and args = 0
ad@ad-desktop:~/bash-scripts$
```

# Computing the Factorial

```
#!/bin/bash
# Usage: factorial number
if [ "$#" -ne 1 ] then echo "Just give one numeric argument"
    exit 1
fi
if [ "$1" -lt 0 ] then echo Please give positive number
    exit 1
fi
fact=1
for ((i = 1; i <= $1; i++)) do
    fact='expr $fact \* $i' # What about fact = $((fact * i)) ?
done
echo $fact
```

## ◇ Outcome:

```
ad@cairns:~/bash-scripts$ ./factorial 3
6
ad@cairns:~/bash-scripts$ ./factorial 5
120
ad@cairns:~/bash-scripts$ ./factorial -23
Please give positive number
ad@cairns:~/bash-scripts$ ./factorial a
./factorial: line 9: [: a: integer expression expected
1
ad@cairns:~/bash-scripts$ ./factorial 24
expr: *: Numerical result out of range
expr: syntax error
expr: syntax error
expr: syntax error
ad@cairns:~/bash-scripts$
```

## Computing the Factorial using *bc*

```
ad@ad-desktop:~/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'.

define f (x) {
    if (x <= 1) return (1);
    return (f(x-1) * x);
}

f(3)
6
f(5)
120
f(6)
720
f(123)
12146304367025329675766243241881295855454217088483382315328918161829\
23589236216766883115696061264020217073583522129404778259109157041165\
147218602951990626164673073390741981495296000000000000000000000000\
00

f(180)
20089606249913429965695133689846683891754034079886777794043533516004\
48609533959809411801381120973097356315941010373996096710321321863314\
95273609598531966730972945653558819806475064353856858157445040809209\
56035846331964466489111425643001782414179675381819233864230269332781\
873198603960320000000000000000000000000000000000000000000000000000\
quit
ad@ad-desktop:~/bash-scripts$
```

## Size of directories

```
#!/bin/bash
# Usage: maxsize dirName1 ... dirNameN
#
max=0; maxdir=$1; dirs=$*;
for dir do
  if [ ! -d $dir ]
    then echo "No directory with name $dir"
  else
    size='du -sk $dir | cut -f1'
    echo "Size of dir $dir is $size"
    if [ $size -ge $max ]
      then
        max=$size ; maxdir=$dir
      fi # if size...
    fi # if directory
  done
echo "$maxdir $max"
```

### ◇ Outcome:

```
ad@cairns:~/bash-scripts$ ./dirSize ~/~/Correspondence/~/EditingProceedings/
Size of dir /home/ad/ is 16711548
Size of dir /home/ad/Correspondence/ is 62456
Size of dir /home/ad/EditingProceedings/ is 69368
/home/ad/ 16711548
ad@cairns:~/bash-scripts$
```

## Printing out the content of a file (in unusual ways)

```
#!/bin/bash
# Loads this script into an array.
text=( $(cat "$1") )

echo Number of parameters is      : ${(#text[@]} - 1)
echo The entire string is        : ${text[@]}
echo Words in the string         : ${#text[@]}
echo The first lexeme encountered: ${text};
echo " "; echo " ";

for element in $(seq 0 ${(#text[@]} - 1))
do
    echo -n "${text[$element]}"; echo -n "#"
done

echo " "; echo " ";

for ((i=0; i <= ${#text[@]} - 1; i++))
do
    echo -n "${text[$i]}"; echo -n "!!"
done
echo " "; echo " ";

for i in `cat "${1}"`
do
    echo -n "${i}"; echo -n "."
done
echo " "
```

# Output

```
ad@haiku:~/src-set002$ ./printContents-p38.sh bosnia
Number of parameters is : 31
The entire string is : Within the spacious , modern courtrooms of Bosnia      s
    war crimes chamber , the harrowing details of the country      s civil
    conflict in the 1990 s are laid bare .
Words in the string : 32
The first lexeme encountered: Within

Within#the#spacious#,#modern#courtrooms#of#Bosnia#    #s#war#crimes#chamber#,#the
#harrowing#details#of#the#country#    #s#civil#conflict#in#the#1990#s#are#
laid#bare#.#

Within!!the!!spacious!! ,!!modern!!courtrooms!!of!!Bosnia!!    !!s!!war!!crimes!!
chamber!! ,!!the!!harrowing!!details!!of!!the!!country!!    !!s!!civil!!
conflict!!in!!the!!1990!!s!!are!!laid!!bare!! .!!

Within.the.spacious.,.modern.courtrooms.of.Bosnia.    .s.war.crimes.chamber.,.the
.harrowing.details.of.the.country.    .s.civil.conflict.in.the.1990.s.are.
laid.bare...
ad@haiku:~/src-set002$
```

## listing of all \*.h files in a directory - output to a file

```
#!/bin/sh
#search for .h files in a specific directory
#For each file in this dir, list first 3 lines in the file
#into the file "myout"
FILE_LIST='ls /home/ad/Dropbox/k24/Transparencies/Set001/src/Sample-C/SampleGCC
/*h'

rm -f myout; touch myout; ls -l myout;
for file in ${FILE_LIST} do
    echo ${file};
    head -3 "${file}" >> myout;
done
```

```
ad@haiku:~/src-set002$ ./listAndCopy-p40.sh
-rw-rw-r-- 1 ad ad 0      7 23:34 myout
/home/ad/Dropbox/k24/Transparencies/Set001/src/Sample-C/SampleGCC/LinkedList.h
/home/ad/Dropbox/k24/Transparencies/Set001/src/Sample-C/SampleGCC/MyHeader.h
/home/ad/Dropbox/k24/Transparencies/Set001/src/Sample-C/SampleGCC/
OrderedLinkedList.h
ad@haiku:~/Dropbox/k24/Transparencies/Set002/src-set002$ cat myout
struct item      *CreateItem(struct item *);
struct item      *InsertItem(struct item *, char *);
struct item      *DeleteItem(struct item *, char *);
#include          <stdio.h>

#define          YES 1
struct item      *OrderedCreateItem(struct item *);
struct item      *OrderedInsertItem(struct item *, char *);
struct item      *OrderedDeleteItem(struct item *, char *);
ad@haiku:~/src-set002$
```



## Read a file and ...

- report contiguous appearances of the same word.
- reporting format: word/#of contiguous occurrences

```
#!/bin/bash
prev=""; cons=1;
for str in `cat ${1}`
do
  if [ "${str}" != "$prev" ]
  then
    if [ ! -z $prev ] # if prev is not of null (0) size
    then
      echo "${prev}/${cons} "
    fi
    prev=${str}
    cons=1
  else
    let "cons = cons + 1"
  fi
done
if [ ! -z prev ] #if prev is not of null (0) size
then
  echo "${prev}/${cons}"
fi
```

```
ad@haiku:~/src-set002$ cat bosnia3
```

```
Within Within Within the spacious , modern modern modern modern courtrooms  
courtrooms of of of
```

```
Bosnia s war crimes crimes chamber chamber , the the the harrowing details  
of the country s civil conflict in the 1990 s are laid bare .
```

```
ad@haiku:~/src-set002$
```

```
ad@haiku:~/src-set002$ ./countword-p41.sh bosnia3
```

```
Within/3
```

```
the/1
```

```
spacious/1
```

```
,/1
```

```
modern/4
```

```
courtrooms/2
```

```
of/3
```

```
Bosnia/1
```

```
 /1
```

```
s/1
```

```
war/1
```

```
crimes/2
```

```
..... etc etc etc
```

## A small guessing game

```
#!/bin/bash

echo -n "Enter a Number:";
read BASE;

myNumber=$(( ((`date +%N` / 1000) % ${BASE}) + 1 ))

guess=-1

while [ "$guess" != "$myNumber" ];
do
    echo -n "I am thinking of a number between 1 and "${BASE}"
    echo -n ". Enter your guess:"
    read guess
    if [ "$guess" = "" ]; then
        echo "Please enter a number."
    elif [ "$guess" = "$myNumber" ]; then
        echo -e "\aYes! $guess is the correct answer!"
    elif [ "$myNumber" -gt "$guess" ]; then
        echo "Sought number is larger than your guess. Try once more."
    else
        echo "Sought number is smaller than your guess. Try once more."
    fi
done
```

## Playing the game

```
ad@ad-desktop:~/Samples$ ./game
Enter a Number:50
I am thinking of a number between 1 and 50. Enter your guess:30
Sought number is smaller than your guess. Try once more.
I am thinking of a number between 1 and 50. Enter your guess:15
Sought number is larger than your guess. Try once more.
I am thinking of a number between 1 and 50. Enter your guess:25
Sought number is smaller than your guess. Try once more.
I am thinking of a number between 1 and 50. Enter your guess:21
Sought number is smaller than your guess. Try once more.
I am thinking of a number between 1 and 50. Enter your guess:18
Sought number is larger than your guess. Try once more.
I am thinking of a number between 1 and 50. Enter your guess:19
Yes! 19 is the correct answer!
ad@ad-desktop:~/Samples$
```

## Using the exec builtin

- ▶ What follows the `exec` replaces the shell without creating a new process!

```
#!/bin/bash
# filename: goalone

exec echo "Exiting \"${0}\"." # Exit from script here.
# -----
# The following lines never execute.

echo "This echo will never echo."
exit 99 # This script will not exit here.
        # Check exit value after script terminates
        #+ with an 'echo $?'.
        # It will *not* be 99.
```

### Running it...

```
ad@ad-desktop:~/additional$ ./goalone
Exiting "./goalone".
ad@ad-desktop:~/additional$ echo $?
0
ad@ad-desktop:~/additional$
```

## Spawning in-place a process with exec

```
#!/bin/bash
# filename" gorepeated
echo
echo "This line appears ONCE in the script, yet it keeps echoing."
echo "The PID of this instance of the script is still $$."
#   Demonstrates that a subshell is not forked off.

echo "===== Hit Ctl-C to exit ====="

sleep 1

exec $0 # Spawns another instance of this same script
        #+ that replaces the previous one.

echo "This line will never echo!" # Why not?

exit 99 # Will not exit here!
        # Exit code will not be 99!
```

# Outcome

```
ad@ad-desktop:~/additional$ ./gorepated

This line appears ONCE in the script, yet it keeps echoing.
The PID of this instance of the script is still 4235.
===== Hit Ctl-C to exit =====

This line appears ONCE in the script, yet it keeps echoing.
The PID of this instance of the script is still 4235.
===== Hit Ctl-C to exit =====
^C
ad@ad-desktop:~/additional$ echo $?
130
ad@ad-desktop:~/additional$ ./gorepated

This line appears ONCE in the script, yet it keeps echoing.
The PID of this instance of the script is still 4239.
===== Hit Ctl-C to exit =====

This line appears ONCE in the script, yet it keeps echoing.
The PID of this instance of the script is still 4239.
===== Hit Ctl-C to exit =====

This line appears ONCE in the script, yet it keeps echoing.
The PID of this instance of the script is still 4239.
===== Hit Ctl-C to exit =====
^C
ad@ad-desktop:~/additional$ echo $?
130
```

```

#!/bin/bash
# Redirecting stdin using 'exec'.

exec 6<&0          # Link file descriptor #6 with stdin.
                  # Saves stdin.

exec < data-file  # stdin replaced by file "data-file"

read a1           # Reads first line of file "data-file".
read a2           # Reads second line of file "data-file."

echo
echo "Following lines read from file."
echo "-----"
echo $a1
echo $a2

echo; echo; echo

exec 0<&6 6<&-
# Now restore stdin from fd #6, where it had been saved,
#+ and close fd #6 ( 6<&- ) to free it for other processes to use.
# <&6 6<&- also works.

echo -n "Enter data "
read b1 # Now "read" functions as expected, reading from normal stdin.
echo "Input read from stdin."
echo "-----"
echo "b1 = $b1"

echo
exit 0

```



# Outcome

```
ad@ad-desktop:~/additional$ ./goredirection
```

```
Following lines read from file.
```

```
-----
```

```
alex
```

```
delis athens monastiraki
```

```
Enter data pyrosvestio
```

```
Input read from stdin.
```

```
-----
```

```
b1 = pyrosvestio
```

```
ad@ad-desktop:~/additional$
```