```
#!/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:

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

```
antoulas@sazerac:~/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
antoulas@sazerac:~/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
```

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

```
antoulas@sazerac:~/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
antoulas@sazerac:~/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:

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

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

```
antoulas@sazerac:~/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).
antoulas@sazerac:~/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:

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

```
antoulas@sazerac:~/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
antoulas@sazerac:~/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:

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

Computing the Factorial using *bc*

```
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’.

define f (x) {
                if (x <= 1) return (1);
                return (f(x-1) * x);
              }
f(3)
6
f(5)
120
f(6)
720
f(123)
121463043670253296757662432418812958554542170884833823153289181618291\
235892362167668831156960612640202170735835221294047782591091570411651\
147218602951990626164673073390741981495296000000000000000000000000000\
00

f(180)
200896062499134299656951336898466838917540340798867777940435335160041\
486095339598094118013811209730973563159410103739960967103213218633141\
952736095985319667309729456535588198064750643538568581574450408092091\
560358463319644664891114256430017824141796753818192338642302693327811\
873198603960320000000000000000000000000000000000000000000000000000000
quit
antoulas@sazerac:~/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:

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

Printing out the content of a file (in unusal 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 encoutered: ${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

```
antoulas@sazerac:~/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 encoutered: 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...
antoulas@sazerac:~/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/antoulas/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
```

```
antoulas@sazerac:~/src-set002$ ./listAndCopy-p40.sh
-rw-rw-r-- 1 ad ad 0          7 23:34 myout
/home/antoulas/Dropbox/k24/Transparencies/Set001/src/Sample-C/SampleGCC/
     LinkedList.h
/home/antoulas/Dropbox/k24/Transparencies/Set001/src/Sample-C/SampleGCC/MyHeader
     .h
/home/antoulas/Dropbox/k24/Transparencies/Set001/src/Sample-C/SampleGCC/
     OrderedLinkedList.h
antoulas@sazerac:~/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 *);
antoulas@sazerac:~/src-set002$
```

Read a file and ...

→ report contiguous appearances of the same word.

→ reporting format: `word/#of contiguous occurences`

```
#!/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
```

```
antoulas@sazerac:~/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 .
antoulas@sazerac:~/src-set002$
```

```
antoulas@sazerac:~/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

```
antoulas@sazerac:~/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!
antoulas@sazerac:~/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...

```
antoulas@sazerac:~/additional$ ./goalone
Exiting "./goalone".
antoulas@sazerac:~/additional$ echo $?
0
antoulas@sazerac:~/additional$
```

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

```
antoulas@sazerac:~/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
antoulas@sazerac:~/additional$ echo $?
130
antoulas@sazerac:~/additional$ ./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
antoulas@sazerac:~/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

```
antoulas@sazerac:~/additional$ ./goredirection

Following lines read from file.
-------------------------------
alex
ntoulas athens monastiraki



Enter data  pyrosvestio
Input read from stdin.
----------------------
b1 = pyrosvestio

antoulas@sazerac:~/additional$
```