Shell Scripting
(bash)
Shell Scripting

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

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

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

```bash
#!/bin/bash
tar -z -cf /var/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 ($N, \ast$)

```bash
#!/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:~/SysProMaterial/Set002/Samples/Sources/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:~/SysProMaterial/Set002/Samples/Sources/bash-scripts$
```
Using Variables, *read-ing From the Shell*

```bash
#!/bin/bash

# NEVER USE SPACES BEFORE AND AFTER = IN ASSIGNMENTS
# No $ before variable name
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 preserve spaces

echo 'hello'     # $hello
# Single quotes quote even $
read b
```

```
ad@cairns:~/Courses/Sys.Pro09/Sources/bash-scripts$ ./variables
a
2334
A B C D
A B C D
$hello
Enter "b" alxdelis
The value of "b" is now $b
ad@cairns:~/Courses/Sys.Pro09/Sources/bash-scripts$
```
Some Arithmetic Operations

```bash
#!/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'  # expr command!

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

- For simple integer operations use `let` (preferred) or `expr`
- The quotes around “expr” are `...`. `$(expr...)` would also work
- For decimal arithmetic use the system program `bc`
# /bin/bash

# WARNING: SPACES ESSENTIAL
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 ' # fault
a=' expr 5 \* 3 '; echo $a  # 15
# need to escape *, since it goes to the shell
a=' expr $a + 5 '; echo $a  # just like a=a+5

string=EnaMegaloString
echo "String is: \${string}"
position=4
length=6
z=' expr substr $string $position $length '  
# Extract length chars from string,  
# starting at position

echo "Substring is: \$z"  # Megalo

 Execution:

ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./myexpr
8
2
1
15
20
String is: EnaMegaloString
Substring is: Megalo
An interesting system program: \textit{bc}

\begin{itemize}
  \item A general purpose calculator
\end{itemize}

```
ad@ad-desktop:~/SysProMaterial/Set002/Samples/Sources/bash-scripts$ bc
bc 1.06.94
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:~/SysProMaterial/Set002/Samples/Sources/bash-scripts$
```
Carrying out decimal arithmetic in *bash*

```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:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./mybc
b = 1.001
1
0
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$
```
Getting the Return Value of a Program ($?)

```bash
#!/bin/bash

# $? returns the exit code of the last command to execute
echo hello
echo $?  # 0 : successful

lsdlisd  # unknown command
echo $?  # 127 - nonzero for an error

echo Hello again

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

• Output:

```
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./exitStatus
hello
0

./exitStatus: line 8: lsdlsd: command not found
127
Hello again
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ echo $?
113
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$
```
More on return Values

- Assume that “dada” does not exist

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

- Output

```
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./myreturn
   rv: 1
   rv: 0
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$
```
bc: working with different scales

ad@ad-desktop:~/SysProMaterial/Set002/Samples$ bc
bc 1.06.94
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:~/SysProMaterial/Set002/Samples$
**bc: working with the binary input base (ibase)**

ad@ad-desk.../SysProMaterial/Set002/Samples$ bc
bc 1.06.94
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-desk.../SysProMaterial/Set002/Samples$
**bc**: using different output base (**obase**)

```
ad@ad-desktop:~/SysProMaterial/Set002/Samples$ bc
bc 1.06.94
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:

  \[
  [ \text{"foo"} = \text{"foo"} ]
  \]

- We may have arithmetic conditions such as:

  ‘2>1’

  which evaluates to TRUE.

- The construct \((...)\) evaluates numerical expressions to either 0 (TRUE) or 1 (FALSE)

  ∗ Opposite from C convention!!! (Think of it as translating C values to Unix exit code for success)
#!/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 $?." # 1
# What effect does the "2>/dev/null" have?
# What would happen if it were removed?
# Try removing it, then rerunning the script.
exit 0
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./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:~/Courses/Sys.Pro10/Sources/bash-scripts$
Checking Files/Directories (-e, -d, -r)

```bash
#!/bin/bash

if [ -e $1 ]  # exists file
  then if [ -f $1 ] # is a regular file
    then echo Regular File
        fi
  fi

# -d checks if it's a directory

if [ -r $1 ]  # have read rights
  then echo I can read this file!!!
  fi

# also -w, -x

◊ checking files - output

ad@cairns:~:/Courses/Sys.Pro10/Sources/bash-scripts$ ./fileTests fileTests
Regular File
I can read this file!!!
ad@cairns:~:/Courses/Sys.Pro10/Sources/bash-scripts$ ./fileTests /tmp/hhh
ad@cairns:~:/Courses/Sys.Pro10/Sources/bash-scripts$
```
Forming Conditions with Integers

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-eq</strong></td>
<td>if [ &quot;$a&quot; –eq &quot;$b&quot; ]</td>
<td>equal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(&quot;$a&quot; == &quot;$b&quot;)</td>
</tr>
<tr>
<td><strong>-ne</strong></td>
<td>if [ &quot;$a&quot; –ne &quot;$b&quot; ]</td>
<td>not-equal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(&quot;$a&quot; &lt;&gt; &quot;$b&quot;)</td>
</tr>
<tr>
<td><strong>-gt</strong></td>
<td>if [ &quot;$a&quot; –gt &quot;$b&quot; ]</td>
<td>greater than</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(&quot;$a&quot; &gt; &quot;$b&quot;)</td>
</tr>
<tr>
<td><strong>-lt</strong></td>
<td>if [ &quot;$a&quot; –lt &quot;$b&quot; ]</td>
<td>less than</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(&quot;$a&quot; &lt; &quot;$b&quot;)</td>
</tr>
<tr>
<td><strong>-le</strong></td>
<td>if [ &quot;$a&quot; –le &quot;$b&quot; ]</td>
<td>less or equal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(&quot;$a&quot; &lt;= &quot;$b&quot;)</td>
</tr>
</tbody>
</table>
Creating Conditions involving Strings

- always use quotes (confusing: [...] interprets $ or > but not =)
- even more confusing: the spaces in [ ... ] are important!

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$ = $b$</td>
<td>equal</td>
</tr>
<tr>
<td>$a$ == $b$</td>
<td>equal</td>
</tr>
<tr>
<td>$a$ != $b$</td>
<td>not-equal</td>
</tr>
<tr>
<td>$a$ &lt; $b$</td>
<td>alphanumerically less</td>
</tr>
<tr>
<td>$a$ &gt; $b$</td>
<td>alphanumerically greater</td>
</tr>
<tr>
<td>-n $a$</td>
<td>not-null</td>
</tr>
<tr>
<td>-z $a$</td>
<td>Null (size 0)</td>
</tr>
</tbody>
</table>
## Creating Conditions involving Strings

<table>
<thead>
<tr>
<th>Condition</th>
<th>Logical Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>! -f &quot;file&quot;</code></td>
<td>Logical NOT</td>
</tr>
<tr>
<td><code>&quot;$a&quot; -a &quot;$b&quot;</code></td>
<td>Logical AND</td>
</tr>
<tr>
<td><code>&quot;$a&quot; -o &quot;$b&quot;</code></td>
<td>Logical OR</td>
</tr>
</tbody>
</table>
If-then-else Control Statement

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

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

```bash
#!/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:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./math
Unknown operation specified.
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./math 34 - 56
Substraction requested.
34 - 56 = -22
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./math 34 % 22
Modulo arithmetic requested.
34 % 22 = 12
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./math 34 * 2
Unknown operation specified.
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./math 34 \* 2
Multiplication requested.
34 * 2 = 68
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./math 34 # 4
Unknown operation specified.
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$
# For Loops

```bash
#!/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
for (( koko=1; koko <= LIMIT; koko++ )) do
    echo $koko "loop with limit"
    #print in different lines
done
```
diamond Outcome:

```
ad@cairns:~$/Courses/Sys.Pro10/Sources/bash-scripts$ ./forLoops
1
2
3
4
5
1 2 3 4 5
1
2
3
4
5
1 loop with limit
2 loop with limit
3 loop with limit
4 loop with limit
5 loop with limit
6 loop with limit
7 loop with limit
8 loop with limit
ad@cairns:~$/Courses/Sys.Pro10/Sources/bash-scripts$
```
More For-Loop + implicit var ($*)

```bash
#!/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:~/.Courses/Sys.Pro10/Sources/bash-scripts$ ./forLoops2 aa bb cc dd ee ff ggg uuu
aadbcdddeeffggguuu
bbccddeefffggguuu
bbccddd
ad@cairns:~/.Courses/Sys.Pro10/Sources/bash-scripts$
```
while [ .. ] do ... done loop

#!/bin/bash
LIMIT=19  # Upper limit
echo "Numbers 1 through 20 (but not 3 or 11)."
a=0
while [ $a -le "$LIMIT" ]
  do
    a=$((a+1))
    # Ignore 3, 11
    if [ "$a" -eq 3 ] || [ "$a" -eq 11 ]
      then continue;
    fi
    echo -n "$a "  # not executed for 3 or 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 "$a 
  done
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./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:~/Courses/Sys.Pro10/Sources/bash-scripts$
More on *While* Loop

```bash
#!/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:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./whileLoops
0 1 2 3 4 5 6 7 8 9
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$
```
Setting implicit var ($*)

```bash
#!/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:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./setProg
Input parameters = 0
Input parameters = 6
one
two
three
four
five
six
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$
```
A (horrible, old-style sh) script that prints strings in reverse

```bash
#!/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:~/SysProMaterial/Set002/Samples/Sources/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:~/SysProMaterial/Set002/Samples/Sources/bash-scripts$
Same script using bash facilities (length, substring)

```bash
#!/bin/bash
# Usage: revstrs [string1 [string2 ...]]
#
for str do
  # Start from the end
  for (( chind=${#str}; chind!=0; chind-- )) do
    echo -n ${str:chind-1:1}
  done
  echo -n " --> "
  echo -n ${#str}
  echo " character(s)."
done
```
Other facilities: redirection, -

```bash
#!/bin/bash
# Listing of Regular Files
OUTFILE=files.lst
dirName=${1-'pwd '} # - declares a default value
    # i.e., if there is no $1
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"
# ^^^^^^^^^^^^^^^^^^ redirecting sorted stdout
```

◇ Outcome:

```
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts/tmp$ ls
alex asoee delis papi uoa upatras
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts/tmp$ cd ..
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./listRegFiles tmp/
The name of the directory to work in: tmp/
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ cat files.lst
Regular files in directory tmp/
tmp/alex
tmp/asoee
tmp/delis
tmp/papi
tmp/uoa
tmp/upatras
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$
```
Shifting parameters in a shellschipt

```bash
#!/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:~/SysProMaterial/Set002/Samples/Sources/bash-scripts$ ./shiftCommand one two three four five six
All args are = one two three four five six
Number of Parameters = 6
The value of the iterator is: one
var = one and args = 5
The value of the iterator is: two
var = two and args = 4
The value of the iterator is: three
var = three and args = 3
The value of the iterator is: four
var = four and args = 2
The value of the iterator is: five
var = five and args = 1
The value of the iterator is: six
var = six and args = 0
```
Overflow: Computing factorial

```bash
#!/bin/bash
# Usage: factorial number
if ("#$" != 1 || $1 < 0)
    then echo "A single non-negative integer expected"
    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:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./factorial 5
120
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./factorial -23
Please give positive number
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./factorial a
./factorial: line 9: [: a: integer expression expected
1
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./factorial 24
expr: *: Numerical result out of range
expr: syntax error
expr: syntax error
expr: syntax error
```
Computing the Factorial using *bc*

```bash
cd ~/SysProMaterial/Set002/Samples/Sources/bash-scripts$
bc
bc 1.06.94
This is free software with ABSOLUTELY NO WARRANTY.
For details type ‘warranty’.

```bash
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\
  2358923621676683115696061264020217073583522129404778259109157041165\
  14721860295199062616467307339074198149529600000000000000000000000000000000000000

f (180) 2008960624991342996569513368984668389175403407988677794043533516004\
  48609533959809411801381120973097356315941010373996096710321321863314\
  95273609598531966730972945653558819806475064353856858157445040809209\
  56035846331964466489111425643001782414179675381819233864230269332781\
  87319860396032000000000000000000000000000000000000000000000000000000000000
quit
```

ad@ad-desktop:$~/SysProMaterial/Set002/Samples/Sources/bash-scripts$
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 # if directory
set - $dirs
done
echo "$maxdir $max"
```

◇ Outcome:

```
ad@cairns:~/Courses/Sys.Pro10/Sources/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:~/Courses/Sys.Pro10/Sources/bash-scripts$
```
Lists as arrays

Printing out the contents of a file (with delimiters)

```bash
#!/bin/bash
text=$( $(cat "$1") ) # all contents in a single array
echo ${text} # just "text" means text[0]!
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 " "
```
Within the spacious, modern courtrooms of Bosnia’s war crimes chamber, the harrowing details of the country’s civil conflict in the 1990s are laid bare.
Using the `exec` builtin

```bash
#!/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:~/SysProMaterial/Set002/Samples/Sources/additional$ ./goalone
Exiting "./goalone".
ad@ad-desktop:~/SysProMaterial/Set002/Samples/Sources/additional$ echo $? 0
ad@ad-desktop:~/SysProMaterial/Set002/Samples/Sources/additional$
# 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 Ctrl-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:~:/SysProMaterial/Set002/Samples/Sources/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 Ctrl-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 Ctrl-C to exit ======================
^C
ad@ad-desktop:~:/SysProMaterial/Set002/Samples/Sources/additional$ echo $? 130
ad@ad-desktop:~:/SysProMaterial/Set002/Samples/Sources/additional$ ./gorepated
This line appears ONCE in the script, yet it keeps echoing.
The PID of this instance of the script is still 4239.
==================== Hit Ctrl-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 Ctrl-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 Ctrl-C to exit ======================
^C
ad@ad-desktop:~:/SysProMaterial/Set002/Samples/Sources/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

```bash
ad@ad-desktop:~/SysProMaterial/Set002/Samples/Sources/additional$ ./
goredirection

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

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

ad@ad-desktop:~/SysProMaterial/Set002/Samples/Sources/additional$
```
#! /bin/bash

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

# take input from $1

# IFS is an internal variable specifying
# how bash separates fields, word boundaries
# ALWAYS SAVE TO TEMP VARIABLE AND
# RESET AFTERWARDS

OLDIFS="$IFS"; echo "--Old IFS value:" "$IFS"

IFS='\n'    # IFS= also works

echo "--New IFS value:" "$IFS"

for line in 'cat "$1"'
    do
    echo "$line"
    done

IFS="$OLDIFS"

exit 0
Within the spacious, modern courtrooms of Bosnia’s war crimes chamber, the harrowing details of the country’s civil conflict in the 1990s are laid bare.