Shell Scripting

(bash)

Shell Scripting

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

```
#!/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

```
#!/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*, $*)

```
#!/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

```
#!/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 $
echo -n "Enter \"b\" "
read b
echo "The value of \"b\" is now $b"
echo ${PATH}        # PATH environment variable
```

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

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‘ # 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*

```
ad@ad-desktop:~/SysProMaterial/Set002/Samples/Sources/bash-scripts$ ./
    arithmetics
2334 2337 2337 2337 2367 2347 2357
ad@ad-desktop:~/SysProMaterial/Set002/Samples/Sources/bash-scripts$
```

More Arithmetic

```
#!/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: *bc*

◇ A general purpose calculator

```
ad@ad-desktop:~/SysProMaterial/Set002/Samples/Sources/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
116900821501443291746534857888750680076954115726 7
quit
ad@ad-desktop:~/SysProMaterial/Set002/Samples/Sources/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:~/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 ($?)

```
#!/bin/bash

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

lsdlsd       # 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"

```
#!/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
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:~/SysProMaterial/Set002/Samples$
```

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

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

bc: using different output base (*obase*)

```
ad@ad-desktop:~/SysProMaterial/Set002/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 to either 0 (TRUE) or 1 (FALSE)
  - 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:~/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)

```
#!/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

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

Creating Conditions involving Strings

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

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

Creating Conditions involving Strings

! if [! -f "file"]	Logical NOT
-a if ["$a" -a "$b"]	Logical AND
-o if ["$a" -o "$b"]	Logical OR

If-then-else 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:~/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

```
#!/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
```

◇ 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
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 ($*)

```
#!/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
aabbccddeeffggguuu
bbccddeeffggguuu
bbccdd
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
echo
```

```
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

```
#!/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 ($*)

```
#!/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

```
#!/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)

```
#!/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, -

```
#!/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

```
#!/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

```
#!/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*

```
ad@ad-desktop:~/SysProMaterial/Set002/Samples/Sources/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
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
    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)

```
#!/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 " "
```

Output

```
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ cat tmp/bosnia
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.
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./printContents tmp/bosnia
Within


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.#

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.!!

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

Using the *exec* builtin

```
#!/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 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:~/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 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:~/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 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:~/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

```
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$
```

Reading a file line by line (vs. as a sequence)

```
#!/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
```

Output

```
ad@cairns:~/Courses/Sys.Pro10/Sources/bash-scripts$ ./printContents2 bosnia
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.
-------------
--Old IFS value:

--New IFS value:

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.
```