Topic 2: Bash Shell Scripting

Bash shell scripting

- Considered easier than C shell
  - Inherits many features from C & Korn shells
  - Most popular shell on Linux systems
  - Linux most popular "Unix" system
- Sequence of commands carrying out a specific task
- # indicates a comment
- First line tells us which shell is to be used
  - #!/bin/bash

- Arguments can be passed to scripts
- Variables and conditions
- Basic control statement (loops for, while, until)
- Numeric and alphanumeric operations
- …
- Can do complicated things effectively

Download bash-scripts.tar from class web site

A small script about input parameters

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

#This is a comment

#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
```

Bash script must be executable to run. Use chmod +x shell_script_name.

```
mema@bowser> ./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
mema@bowser>
```

Using variables - *read*-ing from the shell

```
#!/bin/bash
# Erwthsh: Pote DEN bazoume to '$' mprosta se mia metablhth?
# Apanthsh: Otan ths ana8etoume timh

#NEVER USE SPACES BEFORE AND 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
# Ta dipla eisagwgika diathroun ta polla kena

echo '$hello'   # $hello  [Note the right-leaning direction of the quotes!]
# Ta mona eisagwgika apenergopoioun thn anafora timhs me $
# Try using left-leaning single quotes to see what you get.

echo -n "Enter \"b\" "       # Grafw hey there
read b
echo "The value of \"b\" is now $b"
# The value of "b" is now hey there

echo ${PATH}      # SWSTO - Metablhth periballontos PATH
```

$PATH or ${PATH} if it is easier to read

```
mema@bowser> ./variables
a
2334
A B C D
A B  C    D
$hello
Enter "b" hey there
The value of "b" is now hey there
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/bin/X11
mema@bowser>
```

Some numeric operations

```
#!/bin/bash

a=2334
let b=a+3  # isxyei kai to let b=$a+3
let "c = a+3"
let "d = a+ 3“    #eite me eite xwris kena

z=$(($a+3))
y=$((a+3))  # Epishs swsto

k=`expr $a + 3` # Xrhsh entolhs expr

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

- For simple integer arithmetic use let and expr.
- For decimal arithmetic use the system program bc

```
mema@bowser> ./arithmetics
2334 2337 2337 2337 2337 2337 2337
mema@bowser>
```

More arithmetic

```
#!/bin/bash

# PROSOXH: APAITOYNTAI KENA
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` # Epistrefei sfalma
a=`expr 5 \* 3`; echo $a  # 15. Me to expr, ston pollaplasiasmo \*
a=`expr $a + 5`; echo $a # Idio me let a=a+5

string=EnaMegaloString
echo "String is: ${string}"
position=4
length=6
z=`expr substr $string $position $length`
#E3agei length xarakthres apo to string.
#3ekinaei apo th 8esh position

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

```
mema@bowser> myexpr
8
2
1
15
20
String is: EnaMegaloString
Substring is: Megalo
mema@bowser>
```

bc: A general and versatile purpose calculator

```
mema@bowser> bc
bc 1.06.94
Copyright 1991 -1994 , 1997 , 1998 , 2000 , 2004 , 2006 Free So
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
mema@bowser>
```

bc

```
mema@bowser> 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
41765044911842686789344028906393526046326553
mema@bowser>
```

bc: working with different scales

```
mema@bowser> 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
mema@bowser>
```

bc: working the binary input base (ibase)

```
mema@bowser> 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
mema@bowser>
```

bc: using a different output base (obase)

```
mema@bowser> 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
mema@bowser>
```

Decimal arithmetic in *bash*

```
#!/bin/bash

# EPITREPEI ARI8MHTIKES PRA3EIS SE DEKADIKOUS
a=100.19
b=$(echo "scale=3; $a/100" | bc)
# scale ka8orizei dekadika pshfia

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
```

Με την echo στέλνουμε ως όρισμα στη bc: 0.04-0.03>0 και επιστρέφει 1/0.

```
mema@bowser> ./mybc
b = 1.001
1
0
mema@bowser>
```

Getting the return value of a program

```
#!/bin/bash

# To $? epistrefei ton kwdiko e3odou ths teleytaias
# entolhs pou ektelesthke
echo hello
echo $?     # 0 : epityxhmenh ektelesh

lsdlsd      # agnwsth entolh
echo $?     # 127 - genikws mh mhdenikh se sfalma

echo

exit 113    # Prepei na einai 0-255
```

```
mema@bowser> ./exitStatus
hello
0
./exitStatus: line 8: lsdlsd: command not found
127
mema@bowser>  echo  $?
113
mema@bowser>
```

Conditionals

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

The base for the if construction in bash is:

```
if [expression]; then
code if ‘expression’ is true.
fi
```

Example:

```
#!/bin/bash
if [ "foo" = "foo" ]; then
  echo expression evaluated as true
fi
```

Construct (( ))

- 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!!

Arithmetic Tests

```
#!/bin/bash
# Arithmetic tests.
# The (( ... )) construct evaluates and tests
# numerical expressions.
# Exit status opposite from [ ... ] construct!

(( 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.  Result > 1.
echo "Exit status of \"(( 5 / 4 ))\" is $?." # 0
(( 1 / 2 )) # Division result <1. Division is rounded off to 0
echo "Exit status of \"(( 1 / 2 ))\" is $?."  #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

```
mema@bowser> ./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.
mema@bowser>
```

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
# Omoia, to -d elegxei an prokeitai gia katalogo

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

```
mema@bowser> ls moreExpr
moreExpr*
mema@bowser> ./fileTests moreExpr
Regular File
I can read this file!!!
mema@bowser> ls -l moreExpr
-rwxr-xr-x  1 mema mema 440 Oct 11 09:37 moreExpr*
mema@bowser>
```

Forming Conditions with *Integers*

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

Forming Conditions involving *Strings*

- always use quotes
- 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)

Logical Conditions

! if [! –f “file”]	Logical NOT
-a if [“$a” –a “$b”]	Logical AND
-o if [“$a” –o “$b”]	Logical OR

The if then; elif; else fi; control statement

```
if [expression1];
      then statement1
elif [expressions2];
      then statement2
elif [expression3];
      then statement3
else
       statement4
fi
```

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

The case control statement

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

Example

```
#!/bin/bash

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

```
mema@bowser> ./animal
Enter the name of an animal: cat
The cat has four legs.
mema@bowser> ./animal
Enter the name of an animal : pig
The pig has an unknown number of legs.
mema@bowser>
```

Example script: math

```
#!/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
```

```
mema@bowser> ./math 34 + 56
Addition requested.
34 + 56 = 90
mema@bowser> ./math 34 - 23.3
Substraction requested.
Expr: non-numeric argument   Γιατί;
34 – 23.3 =
mema@bowser> ./math 34 -23
Unknown operation specified.
mema@bowser> ./math 34 - 23
Substraction requested.
34 - 23 = 11
mema@bowser> ./math 34 * 2
Unknown operation specified.   Γιατί;
mema@bowser> ./math 34 \* 2
Multiplication requested.
34 * 2 = 68
```

for Loops

```
#!/bin/bash

for koko in 1 2 3 4 5
do
  echo $koko
#Ektypwsh se diaforetikes grammes
done

for koko in "1 2 3 4 5"
do
  echo $koko
#Ektypwsh se mia grammh
done

NUMS="1 2 3 4 5"
for koko in $NUMS
do
  echo $koko
#Ektypwsh se diaforetikes grammes
done

for koko in `echo $NUMS`
do
  echo $koko
#Ektypwsh se diaforetikes grammes
done

LIMIT=8
#Diples parentheseis, LIMIT xwris $
for ((koko=1; koko <= LIMIT; koko++))
do
  echo $koko "loop me limit"
#Ektypwsh se diaforetikes grammes
done
```

```
mema@bowser> ./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
mema@bowser>
```

Another example

```
#!/bin/bash

#Xwris lista timwn epe3ergazetai tis parametrous
#tou programmatos
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
```

```
mema@bowser> ./forLoops2 aa bb cc dd ee ff gg
aabbccddeeffgg
bbccddeeffgg
bbccdd
mema@bowser>
```

while [] do done loop example

```
#!/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
```

```
mema@bowser> ./whileLoops
0 1 2 3 4 5 6 7 8 9
mema@bowser>
```

Example: breakCont

```
#!/bin/bash
LIMIT=19 # Upper limit
echo
echo "Numbers 1 through 20 (but not 3 and 11)."
a=0
while [ $a -le "$LIMIT" ]
do
a=$(($a+1))
#Agnohse ta 3, 11
if [ "$a" -eq 3 ] || [ "$a" -eq 11 ]
    then continue; # Move on to next iteration of loop
fi
echo  -n "$a " # Den ekteleitai gia ta 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
```

```
mema@bowser> ./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
mema@bowser>
```

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
```

```
mema@bowser> ./setProg ena dio tria tessera
Input parameters = 4
Input parameters = 6
one
two
three
four
five
six
mema@bowser>
```

A script that prints strings in reverse

```
#!/bin/bash
# Usage: revstrs [string1 [string2 ...]]
#
for str
do
 strlen=`expr length "$str"`
 # 8a arxhsoume ektypwsh apo to telos - Prepei na
      # 3eroume mhkos
 chind=$strlen
 while [ $chind -gt 0 ]
      do
      echo -n "`expr substr \"$str\" $chind 1`"
      chind=`expr $chind - 1`
      done
 echo -n " --> "
 echo -n "$strlen"
 echo " character(s)."
done
```

```
mema@bowser> ./revstrs mitsos kitsos aap fitsos pitsos paaa
sostim --> 6 character(s).
sostik --> 6 character(s).
paa --> 3 character(s).
sostif --> 6 character(s).
sostip --> 6 character(s).
aaap --> 4 character(s).
mema@bowser>
```

Listing of Regular Files

```
#!/bin/bash

OUTFILE=files.lst
dirName=${1-`pwd`}      # To - dhlwnei default timh
                # An den dw8ei onoma katalogou apo xrhsth
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"
#      ^^^^^^^^^^^^^^^^^^^
# Anakateu8ynsh ta3inomhmenou stdout
```

```
mema@bowser> cd dirFoo/
mema@bowser> ls
bla1 bla2  files.lst  kk1
mema@bowser> ../listRegFiles /home/mema/k24/bash-scripts/dirFoo/
The name of the directory to work in:  /home/mema/k24/bash-scripts/dirFoo/
mema@bowser> cat files.lst
Regular files in directory /home/mema/k24/bash-scripts/dirFoo/
/home/mema/k24/bash-scripts/dirFoo/bla1
/home/mema/k24/bash-scripts/dirFoo/bla2
/home/mema/k24/bash-scripts/dirFoo/files.lst
/home/mema/k24/bash-scripts/dirFoo/kk1
mema@bowser>
```

Shifting parameters in a shell script

```
#!/bin/bash
# call with > 5 arguments

echo "All args are = $*"
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
```

```
mema@bowser> ./shiftCommand ena \
? dio tria tesera pente exi
All args are = ena dio tria tesera pente exi
The value of the iterator is: ena
var = ena and args = dio tria tesera pente exi
The value of the iterator is: dio
var = dio and args = tria tesera pente exi
The value of the iterator is: tria
var = tria and args = tesera pente exi
The value of the iterator is: tesera
var = tesera and args = pente exi
The value of the iterator is: pente
var = pente and args = exi
The value of the iterator is: exi
var = exi and args =
```

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`
done
echo $fact
```

```
mema@bowser> ./factorial
Just give one numeric argument
mema@bowser> ./factorial -2
Please give positive number
mema@bowser> ./factorial 4
24
mema@bowser> ./factorial 14
87178291200
mema@bowser> ./factorial 24
expr: *: Numerical result out of range
expr: syntax error
expr: syntax error
expr: syntax error

mema@bowser>
```

Size of directories

```
#!/bin/bash
# Usage: dirSize 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"
```

```
mema@bowser>  ./dirSize dirFoo  ~/
Size of dir dirFoo/  is 8
Size of dir /home/mema/ is 19410624
/home/mema/ 19419624
```

Print out the content of a file (in unusual ways)

```
#!/bin/bash
# Loads this script into an array and prints array to stdout

text=( $(cat "$0") )

echo ${text}
echo " "; echo " "; echo “*****”;

for ((i=0; i <= ${#text[@]} - 1; i++))
do
      # ${#text[@]}
# gives number of elements in the array
# prints on a single line separated by "..."
echo -n "${text[$i]}"
echo -n " ... "
done
echo " "; echo " "; echo”*****”;

for i in `cat "${0}"`
do
#each field of the script separated by "...."
echo -n "${i}"
echo -n " .... "
done
echo " “; echo " ";  echo "*****";
```

- An array is a variable containing multiple values.
- To initialize/assign elements to an array variable named text: text = (value1 value2 value3…)
- ${text[3]} is the value of element #3 in text array
- ${text} is same as ${text[0]} which is the value of element #0
- If index number is @ or *, all members of an array are referenced. i.e., ${text[@]} or ${text[*]}

Output

```
mema@bowser> ./printContents
#!/bin/bash


*****
#!/bin/bash ... # ... Loads ... this ... script ... into ... an ... array. ... text=( ...
$(cat ... "$0") ... ) ... echo ... ${text} ... echo ... " ... "; ... echo ... " ... "; ...
echo ... "*****"; ... for ... ((i=0; ... i ... <= ... ${#text[@]} ... - ... 1; ... i++)) ...
do ... # ... ${#text[@]} ... # ... gives ... number ... of ... elements ... in ... the
... array ... # ... prints ... on ... a ... single ... line ... each ... field ... separated
... by ... "..." ... echo ...  ... "${text[$i]}" ... echo ...  ... " ... ... ... " ... done ... echo
... " ... "; ... echo ... " ... "; ... echo ... "*****"; ... for ... i ... in ... `cat ... "${0}"` ...
do ... #each ... field ... of ... the ... script ... separated ... by ... "...." ... echo ...
... "${i}" ... echo ...  ... " ... .... ... " ... done ... echo ... " ... "; ... echo ... " ... ";
... echo ... "*****"; ...

*****
#!/bin/bash .... # .... Loads .... this .... script .... into .... an .... array. .... text=( ....
$(cat .... "$0") .... ) .... echo .... ${text} .... echo .... " .... "; .... echo .... " .... "; ....
echo .... "*****"; .... for .... ((i=0; .... i .... <= .... ${#text[@]} .... - .... 1; .... i++)) ....
do .... # .... ${#text[@]} .... # .... gives .... number .... of .... elements .... in .... the
.... array .... # .... prints .... on .... a .... single .... line .... each .... field .... separated
.... by .... "..." .... echo ....  .... "${text[$i]}" .... echo ....  .... " .... ... .... " .... done .... ec
.... " .... "; .... echo .... " .... "; .... echo .... "*****"; .... for .... i .... in .... `cat .... "${0}"` .
do .... #each .... field .... of .... the .... script .... separated .... by .... "...." .... echo ....
.... "${i}" .... echo ....  .... " .... .... .... " .... done .... echo .... " .... "; .... echo .... " .... ";
.... echo .... "*****"; ....

*****
mema@bowser>
```

Reading a file line by line

```
#!/bin/bash
exec < "$1" #Take input from this file

while read line
  do
  echo $line
  done

echo  “All done with this file!”
exit 0
```

Output

```
mema@bowser> ./readFile printContents
#!/bin/bash
# Loads this script into an array and prints array to stdo

text=( $(cat "$0") )

echo ${text}
echo " "; echo " "; echo “*****”;

for ((i=0; i <= ${#text[@]} - 1; i++))
do
        # ${#text[@]}
# gives number of elements in the array
# prints on a single line separated by "..."
echo -n "${text[$i]}"
echo -n " ... "
done
echo " "; echo " "; echo”*****”;

for i in `cat "${0}"`
do
#each field of the script separated by "...."
echo -n "${i}"
echo -n " .... "
done
echo " “; echo " ";  echo "*****";

All done with this file!
```

Listing of all *.h files in a directory and 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 /usr/include/c++/5/parallel/*.h`

touch myout; rm myout; touch myout;

for file in ${FILE_LIST}
  do
  echo FILE = ${file}
     head -3 "${file}" >> myout
  done
```

Output

```
mema@bowser> ./listAndCopy
FILE = /usr/include/gnutls/compat4.h
FILE = /usr/include/gnutls/compat8.h
FILE = /usr/include/gnutls/extra.h
FILE = /usr/include/gnutls/gnutls.h
FILE = /usr/include/gnutls/openssl.h
FILE = /usr/include/gnutls/x509.h
mema@bowser> cat myout
/* defines for compatibility with older versions.
 */

#ifndef GNUTLS_COMPAT8_H
# define GNUTLS_COMPAT8_H

/*
 * Copyright (C) 2002 Nikos Mavroyanopoulos
 *
/*
 *  Copyright (C) 2000,2001,2002,2003 Nikos Mavroyanopoulos
 *
/*
 * Copyright (c) 2002 Andrew McDonald <andrew@mcdonald.org.uk>
 *
/*
 *  Copyright (C) 2003 Nikos Mavroyanopoulos
 *
mema@bowser>
```

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 $prev ]
        then
        echo "${prev}/${cons} "
     fi
     prev=${str}
     cons=1
  else
     let "cons = cons + 1"
  fi
done
if [ ! -z prev ]
   then
   echo "${prev}/${cons}"
fi
```

Output

```
mema@bowser> more test-file
this is is a test file

another example
example example of a
test test test
test file file
mema@bowser> ./countword test-file
this/1
is/2
a/1
test/1
file/1
another/1
example/3
of/1
a/1
test/4
file/2
mema@bowser>
```

A small guessing game

```
#!/bin/bash

echo -n "Enter a Number:";
read BASE;
# date +%N returns nanoseconds as output
myNumber=$(( ((`date +%N` / 1000) % ${BASE}) +1 ))
guess=-1

while [ "$guess" != "$myNumber" ];
      do
      echo -n "I am thinking of a number between 1
            and "${BASE}". Enter your guess:"
      read guess
      if [ "$guess" = "" ]; then
            echo "Please enter a number."
      elif [ "$guess" = "$myNumber" ]; then
            echo -e "\a Yes! $guess is the correct answer!"
      elif [ "$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
```

Output

```
mema@bowser>  ./game.sh
Enter a Number:34
I am thinking of a number between 1 and 34. Enter your guess:17
Sought number is larger than your guess. Try once more.
I am thinking of a number between 1 and 34. Enter your guess:25
Sought number is larger than your guess. Try once more.
I am thinking of a number between 1 and 34. Enter your guess:30
Sought number is larger than your guess. Try once more.
I am thinking of a number between 1 and 34. Enter your guess:32
 Yes! 32 is the correct answer!
```

Using the exec builtin

```
#!/bin/bash

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.
```

```
mema@bowser>  ./goalone
Exiting “./goalone”.
mema@bowser> echo $?
0
```

Spawing in-place a process with exec

```
#!/bin/bash

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!
```

Output

```
mema@bowser> ./gorepeated.sh

This line appears ONCE in the script, yet it keeps echoing.
The PID of this instance of the script is still 21072.
====================== 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 21072.
====================== 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 21072.
====================== 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 21072.
====================== Hit Ctl-C to exit ============
^C
mema@bowser> 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
```

Output

```
mema@bowser> ./goredirection.sh

Following lines read from file.
-------------------------------
today will be sunny.
let's go to the park.



Enter data  tomorrow will be rainy
Input read from stdin.
----------------------
b1 = tomorrow will be rainy
```