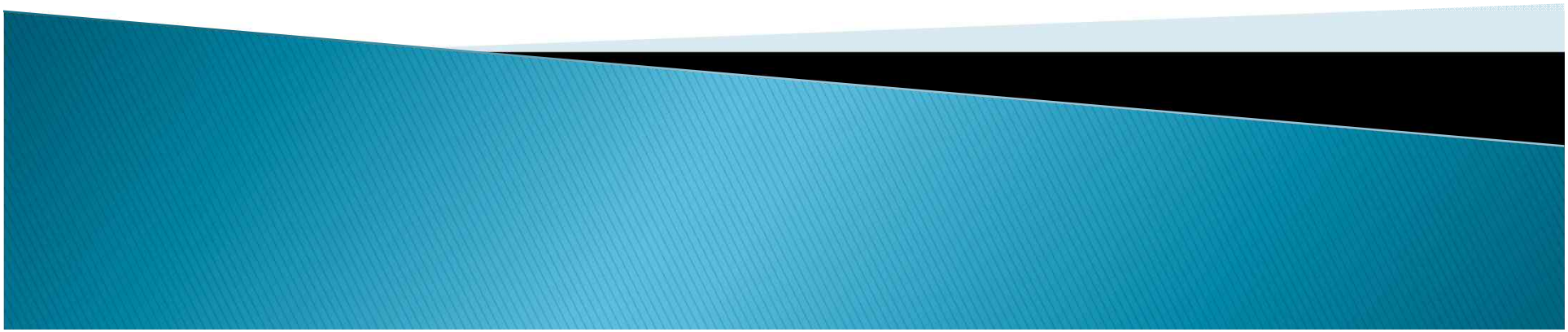
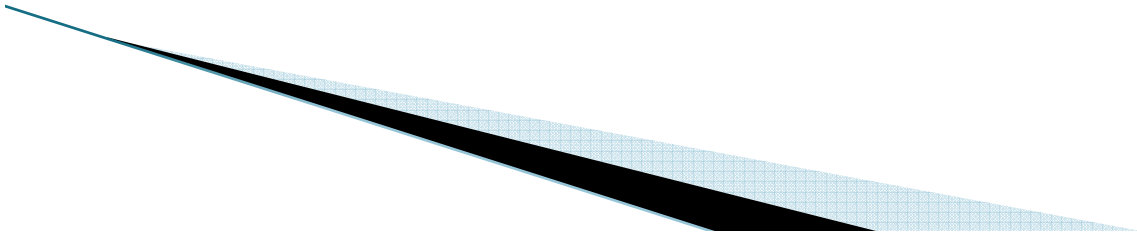
Bash Shell



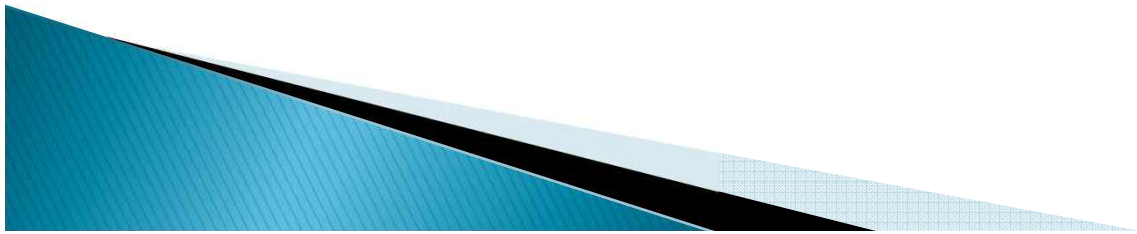
Programming or Scripting ?

- Programming languages
  - a lot more powerful and a lot faster than scripting languages.
  - compiled into an executable.
    - not easily ported into different operating systems.
- scripting language
  - an interpreter reads the instructions in the source file and executes each instruction.
  - slower than compiled programs.
  - easily port the source file to any operating system.



The first bash program

```
$ vi hello.sh

#!/bin/bash
echo "Hello World"

$ chmod 700 hello.sh
$ ./hello.sh
Hello World
```

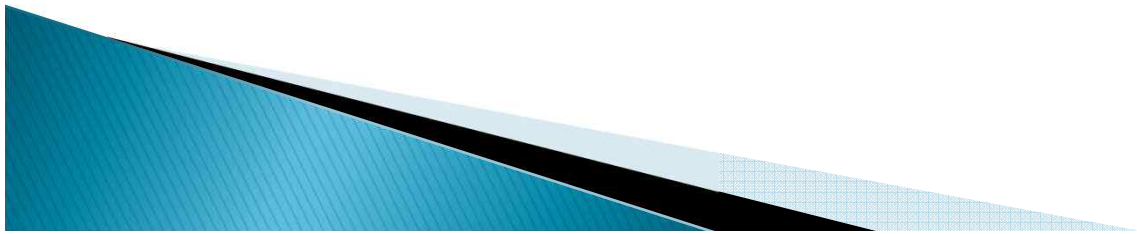


The bash program

- $ mkdir trash
  $ cp * trash
  $ rm -rf trash
  $ mkdir trash
- Instead of having to type all that interactively on the shell, write a shell program instead:

$ cat trash.sh

```
#!/bin/bash
# this script deletes some files
cp * trash
rm -rf trash
mkdir trash
echo "Deleted all files!"
```



Variables

- always stored as strings
- no need to declare a variable

Example

```
$ class=unix13
$ echo class
class
$ echo $class
unix13
$ class = unix13
class: not found
$ class= unix13
unix13: not found
```

```
$ vi hello.sh

#!/bin/bash
STR="Hello World!"
echo $STR

$ ./hello.sh
```

Comments on Variables

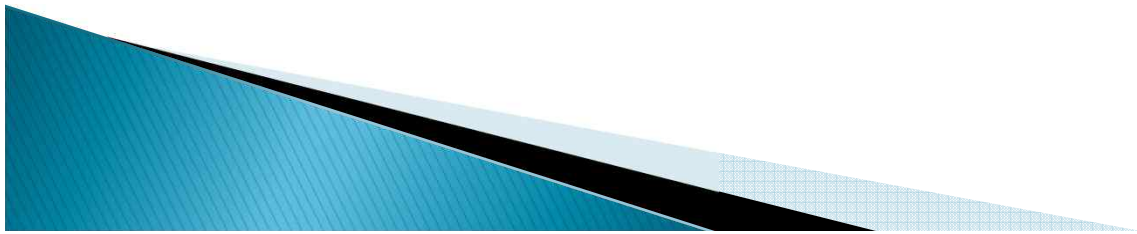
- Case Sensitive
- Notice the problem with spaces
- $ displays the value of the variable

- How do we display the $?

```
echo '$var'
echo \$var
```

  - \ (escape character)

```
$ ls \*
ls: *: No such file or directory
```



Single and Double Quote

- double quotes

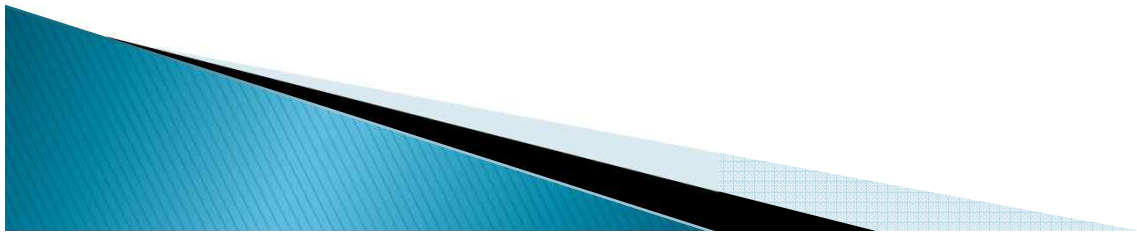
```
$ var="test string"
$ newvar="Value of var is $var"
$ echo $newvar
Value of var is test string
```

- single quotes

```
$ var='test string'
$ newvar='Value of var is $var'
$ echo $newvar
Value of var is $var
```

Readonly Variables (Constants)

```
$ class=system
$ echo $class
system
$ readonly class
$ class=unix13
class: is read only
$ readonly
readonly class
$
```

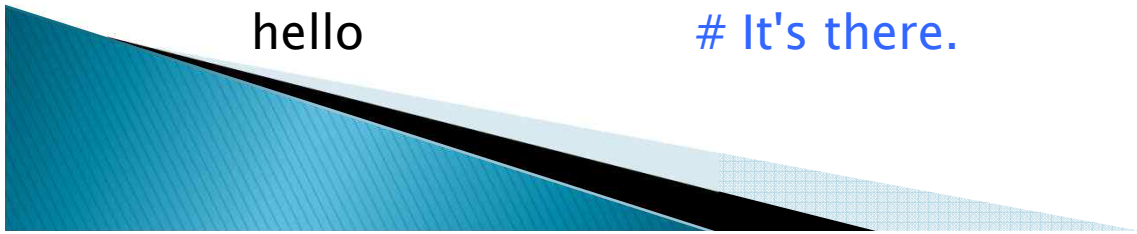


The export command

- Normally the variables you create are local.
- They disappear when the script is done or the shell is exited.
- To make available to a child process, use export variable

  $ export *variable*

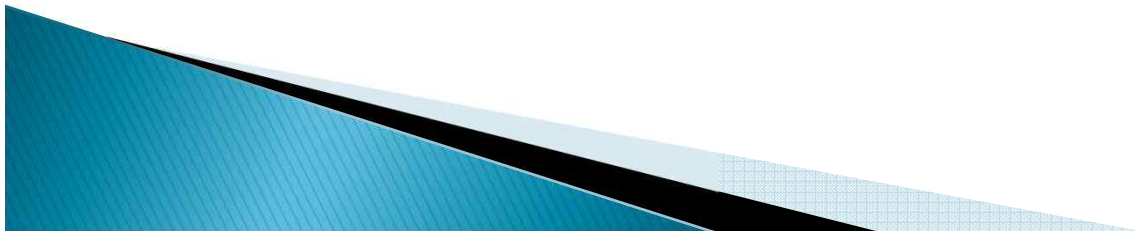
```
$ x=hello
$ bash              # Run a child shell.
$ echo $x           # Nothing in x.
$ exit              # Return to parent.
$ export x
$ bash
$ echo $x
hello               # It's there.
```



The export command

- If the child modifies x, it will not modify the parent's original value.

```
$ x=ciao
$ exit
$ echo $x
hello
```

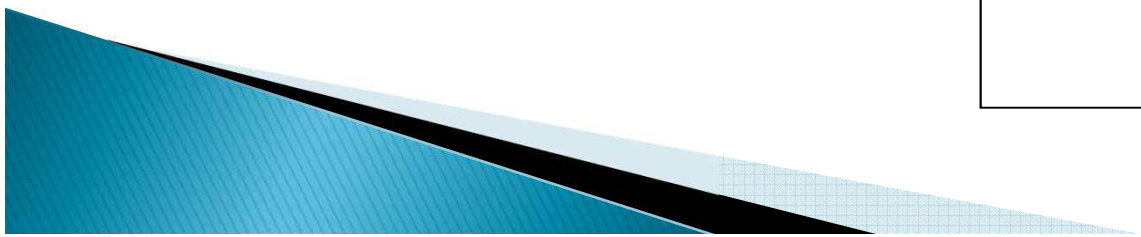


Child Process

```
$ cat p1
class=unix13
echo p1: class = $class
p2
echo p1: class = $class

$ cat p2
echo p2: class = $class
class=system
echo p2: class = $class
```

output

```
$ ./ p1
p1: class = unix13
p2: class =
p2: class = system
p1: class = unix13
```



Child Process

```
$ cat p1
export class
class=unix13
echo p1: class = $class
p2
echo p1: class = $class

$ cat p2
echo p2: class = $class
class=system
echo p2: class = $class
```

output

```
$ ,/ p1
p1: class = unix13
p2: class = unix13
p2: class = system
p1: class = unix13
```

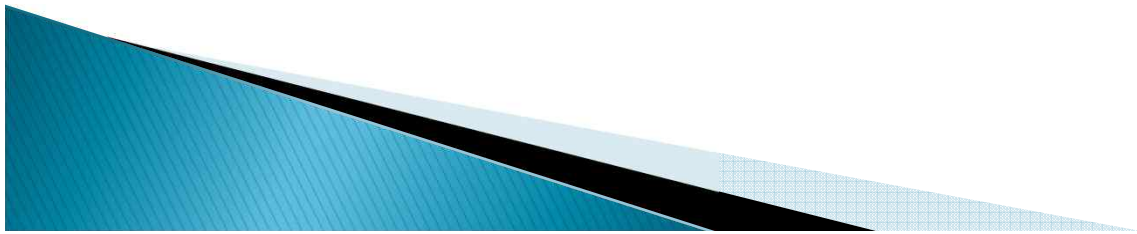
Notice the modification is not passed back.

Environmental Variables

- There are two types of variables:
  - Local variables
  - Environmental variables

```
$ echo $SHELL
/bin/bash
$ echo $PATH
/usr/lib/qt3.3/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/us
  r/sbin:/usr/bin:/root/bin
```

- defined in /etc/profile, /etc/profile.d/ and ~/.bash_profile.
- When a login shell exits, bash reads ~/.bash_logout

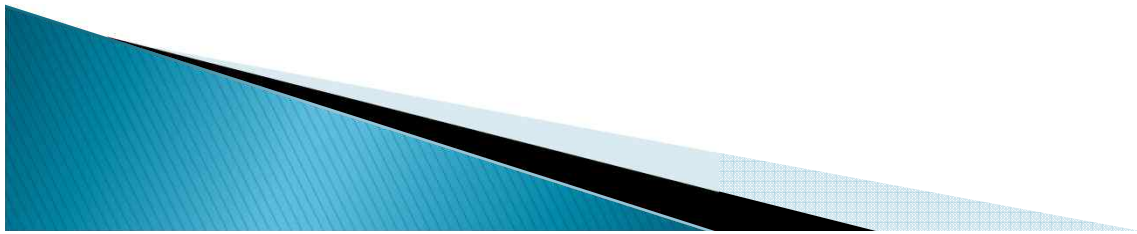


Read command

- Read the input, store in variable
- Can read more than one at a time
  - read v1 v2 v3

- Example:

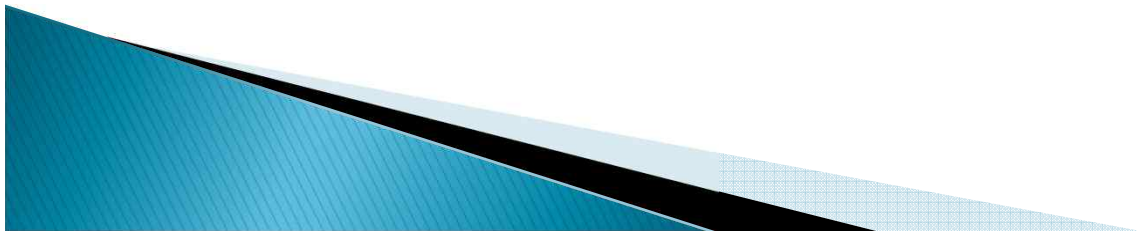
```
$ cat readtest
echo -n "Enter your name: "
read uname
echo "Hello $uname"
```



Command Substitution

- Use the ` (backquote) character to redirect the output of a command to a variable.

```
$ LIST=`ls`
$ echo $LIST
hello.sh read.sh

$ stime=`date`
$ echo $stime
Sat Nov 20 23:23:23 KST 2001
```



Command Substitution

- perform the command substitution by means of $(command)

```
$ LIST=$(ls)
$ echo $LIST
hello.sh read.sh

$ rm $( find / -name "*.tmp" )

$ cat > backup.sh
#!/bin/bash
BCKUP=/home/userid/backup-$(date +%d-%m-%y).tar.gz
tar -czf $BCKUP $HOME
```

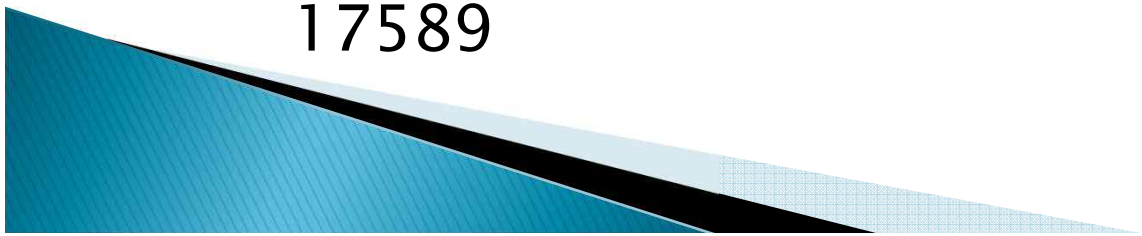
Arithmetic Evaluation

- let statement

```
$ let X=10+2*7
$ echo $X
24
```

- An arithmetic expression can be evaluated by $[expression] or $((expression))

```
$ echo "$((123+20))"
143
$ VALORE=$[123+20]
$ echo "$[123*$VALORE]"
17589
```



Arithmetic Evaluation

- Available operators: +, -, /, *, %

```
$ cat arithmetic.sh
echo -n "Enter the first number: "; read x
echo -n "Enter the second number: "; read y
add=$(($x + $y))
sub=$(($x - $y))
mul=$(($x * $y))
div=$(($x / $y))
mod=$(($x % $y))
echo "Sum: $add"          # print out the answers:
echo "Difference: $sub"
echo "Product: $mul"
echo "Quotient: $div"
echo "Remainder: $mod"
```

Conditional Statements

```
if [ expression ];
then
      statements
elif [ expression ];
then
      statements
else
      statements
fi
```

- the elif (else if) and else sections are optional

Expressions

- String comparison
- Numeric comparison
- File operators

- String Comparisons:

  =, != compare if two strings are "equal" or "not equal"

  -n, -z evaluate if string length is "greater than zero" or "equal to zero"

- Examples:

[s1 = s2] (true if s1 same as s2, else false)

[s1] (true if s1 is not empty, else false)

[-z s2] (true if s2 has a length of 0, otherwise false)

Expressions

- Number Comparisons:

-eq compare if two numbers are equal
-ge compare if one number is greater than or equal to a number
-le compare if one number is less than or equal to a number
-ne compare if two numbers are not equal
-gt compare if one number is greater than another number
-lt compare if one number is less than another number

- Examples:

[n1 -eq n2] (true if n1 same as n2, else false)
[n1 -ge n2] (true if n1 greater then or equal to n2, else false)
[n1 -le n2] (true if n1 less then or equal to n2, else false)
[n1 -ne n2] (true if n1 is not same as n2, else false)
[n1 -gt n2] (true if n1 greater then n2, else false)
[n1 -lt n2] (true if n1 less then n2, else false)

Examples

```
$ cat user.sh
 #!/bin/bash
    echo -n "Enter your login name: "
    read name
    if [ "$name" = "$USER" ];
    then
            echo "Hello, $name. How are you today ?"
    else
            echo "You are not $USER, so who are you ?"
    fi

$ cat number.sh
#!/bin/bash
    echo -n "Enter a number 1 < x < 10: "
    read num
    if [ "$num" -lt 10 ]; then
            if [ "$num" -gt 1 ]; then
                    echo "$num*$num=$(($num*$num))"
            else
                    echo "Wrong insertion !"
            fi
    else
            echo "Wrong insertion !"
    fi
```

Expressions

- Files operators:

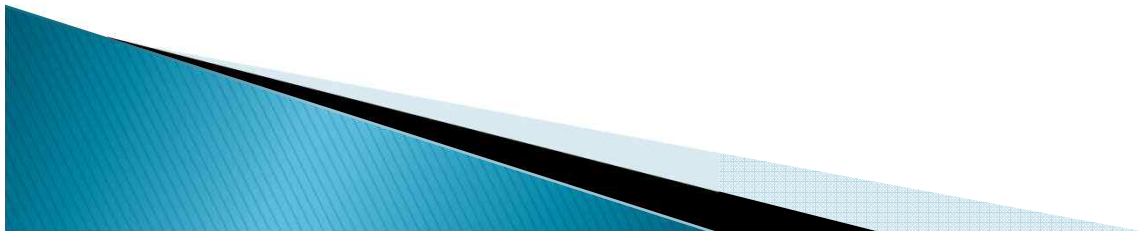
-d check if path given is a directory
-f check if path given is a file
-e check if file name exists
-r check if read permission is set for file or directory
-w check if write permission is set for a file or directory
-x check if execute permission is set for a file or directory
-s check if a file has a length greater than 0

- Examples:

[-d fname] (true if fname is a directory, otherwise false)
[-f fname] (true if fname is a file, otherwise false)
[-e fname] (true if fname exists, otherwise false)
[-s fname] (true if fname length is greater then 0, else false)
[-r fname] (true if fname has the read permission, else false)

Example

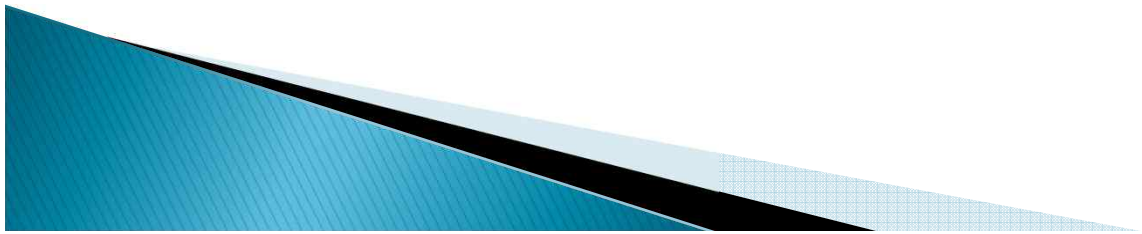
```
#!/bin/bash
  if [ -f /etc/fstab ];
  then
       cp /etc/fstab .
       echo "Done."
  else
       echo "This file does not exist."
       exit 1
  fi
```



In Class

- Write a script that asks for a number greater than 10. (gt10.sh)

- If it is, print
  Good Going Chester
- If not
  Try again zippy



In Class

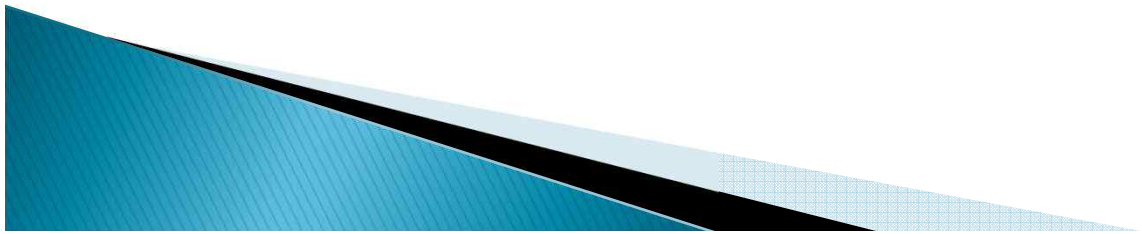
- Let's write a script to tell use about a file...

$ about.sh *filename*

we can read it

we can write it

it is executable



In Class

- Write a program to display:
  - Good Morning(<12)
  - Good Afternoon(<18)
  - Good Evening(<22)
- Depending on the time
- use “elif”
- Hint: date +%H

- **date.sh**

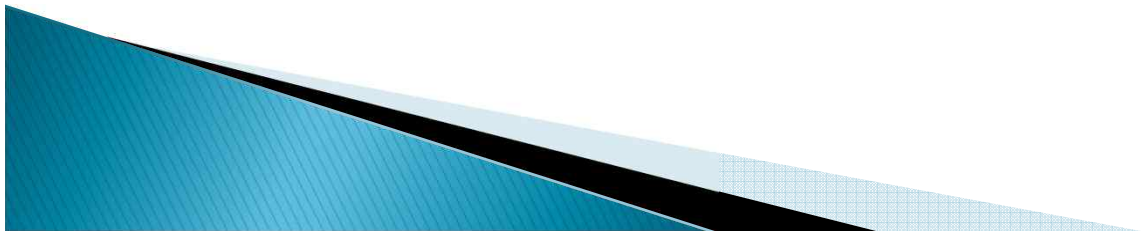
homework

- Write a shell script “bakcp.sh” which:
  - accepts a file name as parameter
  - checks if file exists
  - if file exists, copy the file to the same name + .bak (if the backup file already exists ask if you want to replace it).
- When done you should have the original file and one with a .bak at the end
- Echo your student id on the first line.
- protect it: chmod 700 bakcp.sh

homework

- Present a multiple choice question, (4 choices), gets the user's response, and reports back whether the answer is right, wrong, or not one of the choices.

- Echo your student id on the first line.
- Call it mc.sh
- protect it: chmod 700 mc.sh



Expressions

- Logical operators:

! negate (NOT) a logical expression

-a logically AND two logical expressions

-o logically OR two logical expressions

Example:

```
$ cat chnum.sh
   echo -n "Enter a number 1 < x < 10:"
   read num
   if [ "$num" -gt 1 -a "$num" -lt 10 ];
   then
         echo "$num*$num=$(($num*$num))"
   else
         echo "Wrong insertion !"
   fi
```

Expressions

- Logical operators:

&& logically AND two logical expressions

|| logically OR two logical expressions

Example:

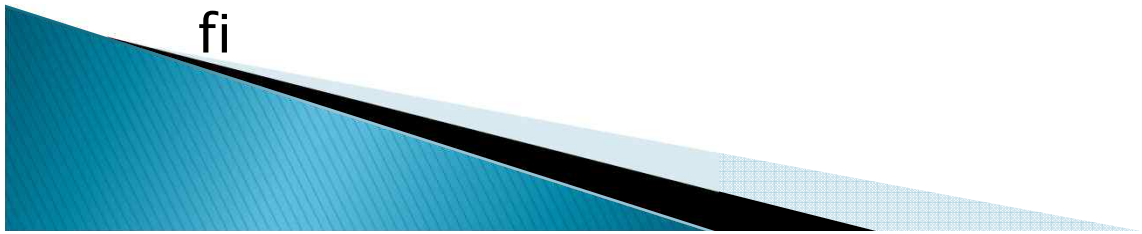
```
$ cat chnum.sh
   echo -n "Enter a number 1 < x < 10: "
   read num
   if [ "$num" -gt 1 ] && [ "$num" -lt 10 ];
   then
         echo "$num*$num=$(($num*$num))"
   else
         echo "Wrong insertion !"
   fi
```

Example

```
$ cat iftrue.sh
   #!/bin/bash
   echo "Enter a path: "; read x
   if cd $x; then
         echo "I am in $x and it contains"; ls
   else
         echo "The directory $x does not exist";
         exit 1
   fi

$ iftrue.sh
Enter a path: /home
userid anotherid ...
$ iftrue.sh
Enter a path: blah
The directory blah does not exist
```

Trash

```
$ cat trash.sh
  #!/bin/bash
  if [ $# -eq 1 ];
  then
        if [ ! -d "$HOME/trash" ];
        then
                mkdir "$HOME/trash"
        fi
        mv $1 "$HOME/trash"
  else
        echo "Use: $0 filename"
        exit 1
  fi
```



Shell Parameters

- Positional parameters(read only)

$0 Name of the program

$1 First argument
$2 second argument

$* All the arguments

$# How many arguments

$@ All the arguments of array

```
$ cat sparameters.sh
echo "$#; $0; $1; $2; $*; $@"
$ sparameters.sh arg1 arg2
2; sparameters.sh; arg1; arg2; arg1 arg2; arg1 arg2
```

Shell Parameters

shift

- You can access only 9 command line arguments
- Moves them all down by 1
  - $2 becomes $1…
  - Tenth variable becomes $9

set

- Set can be used to the set $1 – $9

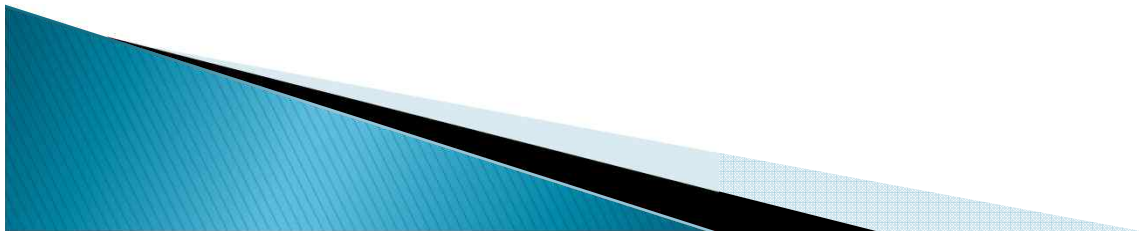
  set first second third
  - $1 = first
  - $2 = second
  - $3 = third

```
$ cat t
set `date`
echo $4

$ t
23:23:15
```

set

- Write a script called owns that displays the owner of any file.

```
$ ./owns.sh nums
Unix00
```



set

- A possible solution

```
#!/bin/sh
set `ls -l $1`
echo $3
```

  - But it always produces an error

- The Real Solution

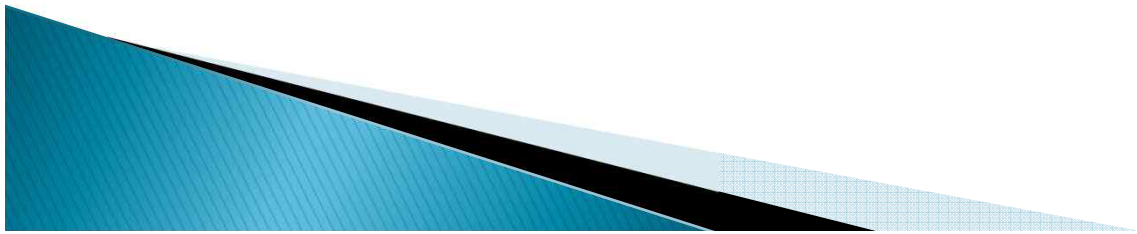
```
#!/bin/sh
set -- `ls -ld $1`
echo $3
```

  - - - ignore any more options
  - -ld long display, don't display dir

Case Statement

```
case $var in
   val1)
         statements;;
   val2)
         statements;;
   *)
         statements;;
   esac
```

- * matches everything (default)
- [ ] Character class
- | Alternative Choices



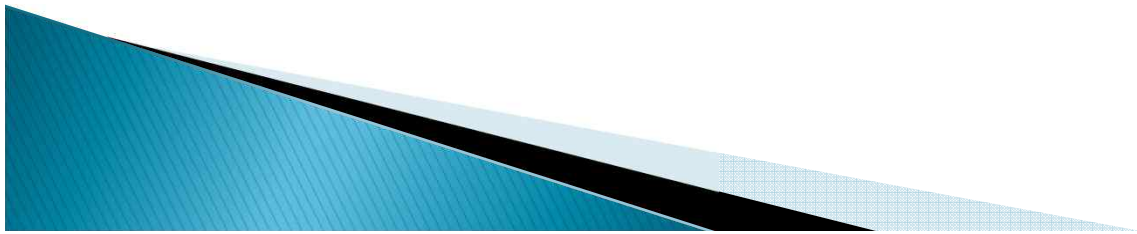
Example (case.sh)

```
$ cat case.sh
#!/bin/bash
   echo -n "Enter a number 1 < x < 10: "
  read x
   case $x in
          1) echo "Value of x is 1.";;
          2) echo "Value of x is 2.";;
          3) echo "Value of x is 3.";;
          4) echo "Value of x is 4.";;
          5) echo "Value of x is 5.";;
          6) echo "Value of x is 6.";;
          7) echo "Value of x is 7.";;
          8) echo "Value of x is 8.";;
          9) echo "Value of x is 9.";;
          0 | 10) echo "wrong number.";;
          *) echo "Unrecognized value.";;
   esac
```

homework

- rewrite the homework(mc.sh) to use a case statement instead of an if.

- Echo your student id on the first line.
- Put it in your home directory
- Call it mccase.sh
- protect it: chmod 700 mccase.sh

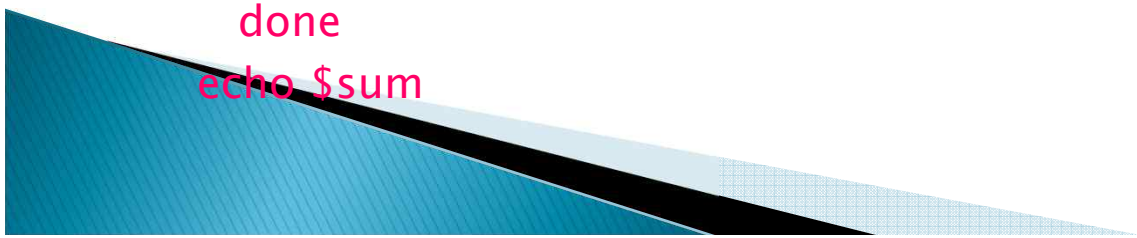


Iteration Statements

```
for var in list
   do
     statements
   done
```

- Unlike other languages, the for loop in Unix is to process multiple arguments – not for counting.
- Example

```
$ forln.sh
  #!/bin/bash
   let sum=0
   for num in 1 2 3 4 5
      do
        let "sum = $sum + $num"
      done
   echo $sum
```



Iteration Statements

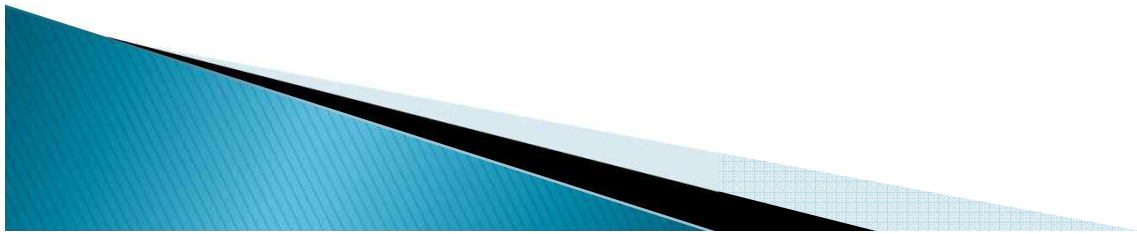
- if the list part is left off, The default is the arguments on the command line.
- Example

```
$ cat for1.sh
  #!/bin/bash
  for x
  do
    echo "The value of variable x is: $x"
    sleep 1
  done
$ for1.sh arg1 arg2
  The value of variable x is: arg1
  The value of variable x is: arg2
```

In Class

- Write a program display the arguments on a line like this:

```
$ dargs.sh this is a test
1) this
2) is
3) a
4) test
```



Example (old.sh)

```
$ cat old.sh
#!/bin/bash
# Move the command line arg files to old directory.
if [ $# -eq 0 ]                    #check for command line arguments
then
  echo "Usage: $0 file ..."
  exit 1
fi
if [ ! -d "$HOME/old" ]
then
  mkdir "$HOME/old"
fi
echo The following files will be saved in the old directory:
echo $*
for file in $*                     #loop through all command line arguments
do
  mv $file "$HOME/old/"
  chmod 400 "$HOME/old/$file"
done
ls -l "$HOME/old"
```

Example (args.sh)

```
$ cat args.sh
#!/bin/bash
# Invoke this script with several arguments: "one two three"
if [ ! -n "$1" ]; then
   echo "Usage: $0 arg1 arg2 ..." ; exit 1
fi
echo ; index=1 ;
echo "Listing args with \"\$*\":"
for arg in "$*" ;
do
   echo "Arg $index = $arg"
   let "index+=1"            # increase variable index by one
done
echo "Entire arg list seen as single word."
echo ; index=1 ;
echo "Listing args with \"\$@\":"
for arg in "$@" ; do
   echo "Arg $index = $arg"
   let "index+=1"
done
echo "Arg list seen as separate words." ; exit 0
```

Using Arrays with Loops

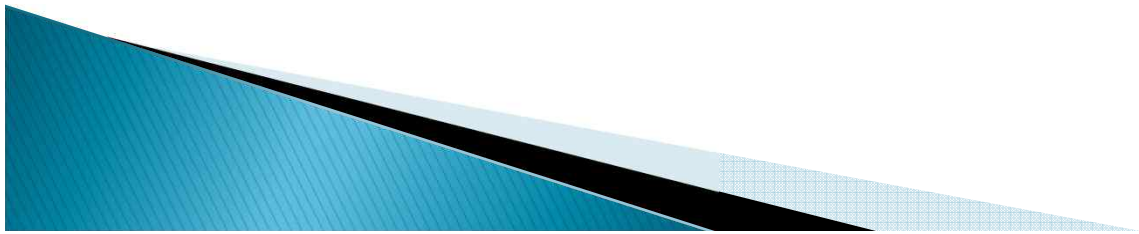
```
pet[0]=dog
pet[1]=cat
pet[2]=fish
pet=(dog cat fish)

$ echo ${pet[0]}
   dog
```

- To extract all the elements, use an asterisk as:

```
echo ${arrayname[*]}
```

- We can combine arrays with loops using a for loop:

```
for x in ${arrayname[*]}
   do
         ...
   done
```



A C-like for loop

- An alternative form of the for structure is

```
for (( EXPR1 ; EXPR2 ; EXPR3 ))
   do
        statements
   done
```

```
$ cat for2.sh
   #!/bin/bash
   echo -n "Enter a number: "; read x
   let sum=0
   for (( i=1 ; $i<=$x ; i=$i+1 )) ; do
     let "sum = $sum + $i"
   done
   echo "the sum of the first $x numbers is: $sum"
```

While Statements

```
while expression
  do
      statements
  done
```

```
$ cat while.sh
  #!/bin/bash
  echo -n "Enter a number: "; read x
  let sum=0; let i=1
  while [ $i -le $x ]; do
    let "sum = $sum + $i"
    let i=$i+1
  done
  echo "the sum of the first $x numbers is: $sum"
```

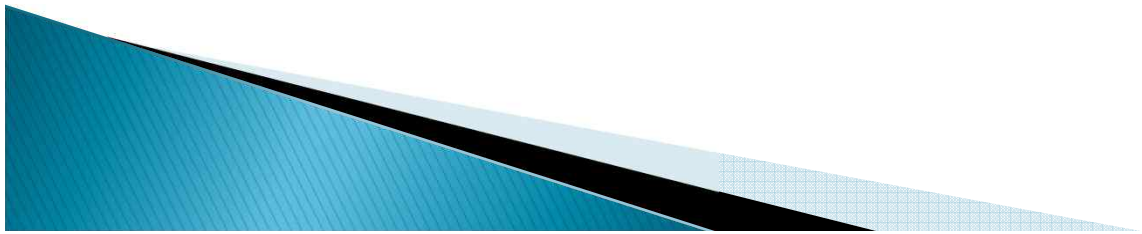
Menu

```
$ cat menu.sh
#!/bin/bash
    clear ; loop=y
    while [ "$loop" = y ] ;
    do
      echo "Menu";  echo "===="
      echo "D: print the date"
      echo "W: print the users who are currently log on."
      echo "P: print the working directory"
      echo "Q: quit."
      echo
      read -s choice                    # silent mode: no echo to terminal
      case $choice in
            D | d) date ;;
            W | w) who ;;
            P | p) pwd ;;
            Q | q) loop=n ;;
            *) echo "Illegal choice." ;;
      esac
      echo
    done
```

In Class

- Read in numbers until a 0 is entered. Then display the sum and the average.

- Call it “while.sh”



Until Statements

```
until [expression]
   do
        statements
   done
```

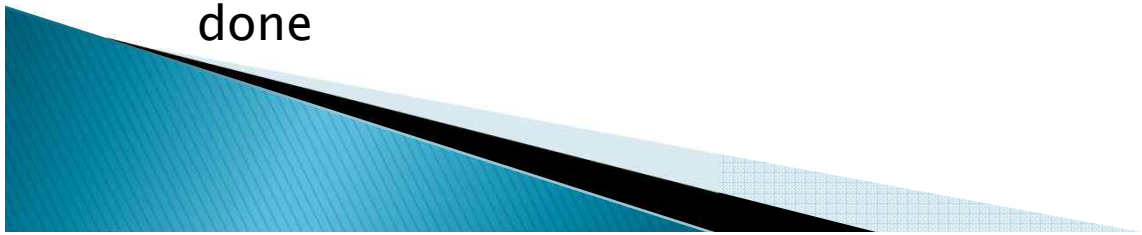
```
$ cat countdown.sh
   #!/bin/bash
   echo "Enter a number: "; read x
   echo ; echo Count Down
   until [ "$x" -le 0 ]; do
         echo $x
         x=$(($x -1))
         sleep 1
   done
   echo ; echo GO !
```

Find a Pattern and Edit

```
$ cat grepedit.sh
#!/bin/bash
# Edit argument files $2 ..., that contain pattern $1
if [ $# -le 1 ]
then
  echo "Usage: $0 pattern file ..." ; exit 1
else
  pattern=$1                # Save original $1
  shift                     # shift the positional parameter to the left by 1
  while [ $# -gt 0 ]        # New $1 is first filename
  do
    grep "$pattern" $1 > /dev/null
    if [ $? -eq 0 ] ; then  # If grep found pattern
      vi $1                 # then vi the file
    fi
    shift
  done
fi
$ grepedit.sh while ~
```

Continue Statements

```
$ cat continue.sh
#!/bin/bash
   LIMIT=19
   echo
   echo "Printing Numbers 1 through 20 (but not 3 and 11)"
   a=0
   while [ $a -le "$LIMIT" ]; do
     a=$(($a+1))
     if [ "$a" -eq 3 ] || [ "$a" -eq 11 ]
     then
          continue
     fi
     echo -n "$a "
   done
```



Break Statements

```
$ cat break.sh
#!/bin/bash
   LIMIT=19
   echo
   echo "Printing Numbers 1 through 20, but something happens after
   2 ... "
   a=0
   while [ $a -le "$LIMIT" ]
   do
     a=$(($a+1))
     if [ "$a" -gt 2 ]
     then
      break
     fi
    echo -n "$a "
  done
  echo; echo; echo
  exit 0
```

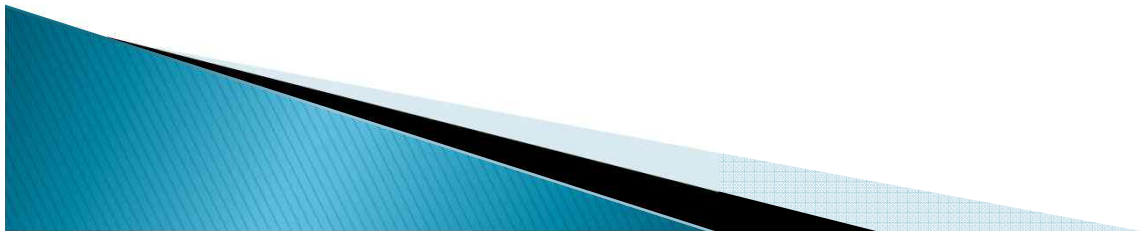
Debugging

-x : displays each line of the script with variable substitution and before execution

- Usage: #!/bin/bash -x or $bash -x or (sh -x)

$ cat for3.sh

```
#!/bin/bash -x
echo -n "Enter a number: "; read x
let sum=0
for (( i=1 ; $i<$x ; i=$i+1 )) ; do
  let "sum = $sum + $i"
done
echo "the sum of the first $x numbers is: $sum"
```



Debugging

```
$ for3.sh
+ echo -n 'Enter a number: '
Enter a number: + read x
3
+ let sum=0
+ (( i=0 ))
+ (( 0<=3 ))
+ let 'sum = 0 + 0'
+ (( i=0+1 ))
+ (( 1<=3 ))
+ let 'sum = 0 + 1'
+ (( i=1+1 ))
+ (( 2<=3 ))
+ let 'sum = 1 + 2'
+ (( i=2+1 ))
+ (( 3<=3 ))
+ let 'sum = 3 + 3'
+ (( i=3+1 ))
+ (( 4<=3 ))
+ echo 'the sum of the first 3 numbers is: 6'
the sum of the first 3 numbers is: 6
```

Manipulating Strings

${#string} gives the string length

${string:position} extracts sub-string from $string at $position

${string:position:length} extracts $length characters of sub-string from $string at $position

```
$ st=0123456789
$ echo ${#st}
  10
$ echo ${st:6}
  6789
$ echo ${st:6:2}
  67
```

Parameter Substitution

- Manipulating and/or expanding variables

${parameter-default}, if parameter not set, use default.

```
$ echo ${username-`whoami`}
    alice
$ username=bob
$ echo ${username-`whoami`}
    bob
```

${parameter=default}, if parameter not set, set it to default.

```
$ unset username
$ echo ${username=`whoami`}
$ echo $username
    alice
```

${parameter+value}, if parameter set, use value, else use null string.

```
$ echo ${username+bob}
    bob
```

Parameter Substitution

${parameter?msg}, if parameter set, use it, else print msg

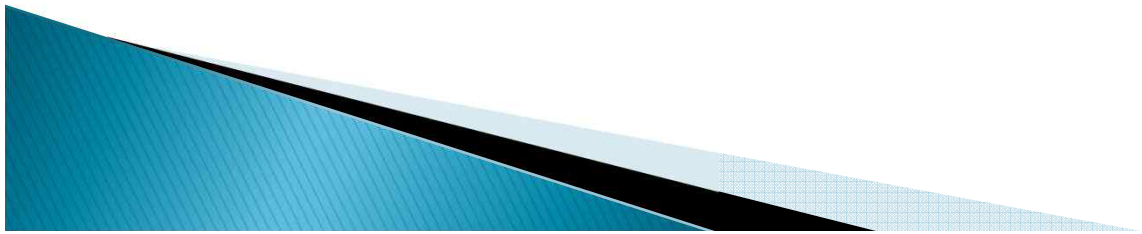
```
$ value=${total?'total is not set'}
   total: total is not set
$ total=10
$ value=${total?'total is not set'}
$ echo $value
   10
```

- Example

```
#!/bin/bash
OUTFILE=symlinks.list                    # save file
directory=${1-`pwd`}
for file in "$( find $directory -type l )"
                                         # -type l == symbolic links
do
  echo "$file"
done | sort >> "$HOME/$OUTFILE"
exit 0
```

Functions

```
#!/bin/bash
   hello()
   {
   echo "You are in function hello()"
   }
   echo "Calling function hello()..."
   hello
   echo "You are now out of function hello()"
```



Functions

```
$ cat function.sh
#!/bin/bash
function check() {
if [ -e "/home/$1" ]
then
  return 0
else
  return 1
fi
}
echo "Enter the name of the file: " ; read x
if check $x
then
  echo "$x exists !"
else
  echo "$x does not exists !"
fi.
```

Example: Picking a random card from a deck

```
#!/bin/bash
# Count how many elements.

Suites="Clubs Diamonds Hearts Spades"
Denominations="2 3 4 5 6 7 8 9 10 Jack Queen King Ace"

# Read into array variable.
suite=($Suites)
denomination=($Denominations)

# Count how many elements.
num_suites=${#suite[*]}
num_denominations=${#denomination[*]}
echo -n "${denomination[$((RANDOM%num_denominations))]}
   of "
echo ${suite[$((RANDOM%num_suites))]}
exit 0
```

Example: Changes all filenames to lowercase

```
#!/bin/bash
for filename in *
                        # Traverse all files in directory.
do
                        # Get the file name without the path.
  fname=`basename $filename`
                # Change name to lowercase.
  n=`echo $fname | tr A-Z a-z`
  if [ "$fname" != "$n" ]
                # Rename only files not already lowercase.
  then
    mv $fname $n
  fi
done
exit 0
```

Example: Compare two files with a script

```
#!/bin/bash
ARGS=2                                          # Two args to script expected.
if [ $# -ne "$ARGS" ]; then
  echo "Usage: `basename $0` file1 file2" ; exit 1
fi
if [[ ! -r "$1" || ! -r "$2" ]] ;  then
  echo "Both files must exist and be readable." ; exit 2
fi
                                # /dev/null buries the output of the "cmp" command.
cmp $1 $2 &> /dev/null
                                # Also works with 'diff', i.e., diff $1 $2 &> /dev/null
if [ $? -eq 0 ]                 # Test exit status of "cmp" command.
then
  echo "File \"$1\" is identical to file \"$2\"."
else
  echo "File \"$1\" differs from file \"$2\"."
fi
exit 0
```

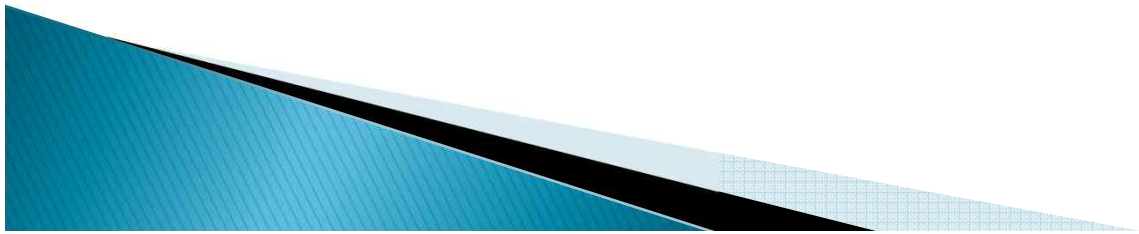
Example: Suite drawing statistics

```
$ cat cardstats.sh
#!/bin/sh # -xv
N=100000
hits=(0 0 0 0)                                  # initialize hit counters
if [ $# -gt 0 ]; then                           # check whether there is an argument
      N=$1
else                                            # ask for the number if no argument
      echo "Enter the number of trials: "
      TMOUT=5                                   # 5 seconds to give the input
      read N
fi
i=$N
echo "Generating $N random numbers... please wait."
SECONDS=0 # here is where we really start
while [ $i -gt 0 ]; do  # run until the counter gets to zero
      case $((RANDOM%4)) in                     # randmize from 0 to 3
            0) let "hits[0]+=1";;               # count the hits
            1) let "hits[1]=${hits[1]}+1";;
            2) let hits[2]=$((${hits[2]}+1));;
            3) let hits[3]=$((${hits[3]}+1));;
      esac
      let "i-=1"                                # count down
done
echo "Probabilities of drawing a specific color:"
                                                # use bc - bash does not support fractions
echo "Clubs: " `echo ${hits[0]}*100/$N | bc -l`
echo "Diamonds: " `echo ${hits[1]}*100/$N | bc -l`
echo "Hearts: " `echo ${hits[2]}*100/$N | bc -l`
echo "Spades: " `echo ${hits[3]}*100/$N | bc -l`
echo "==========  ================================="
echo "Execution time: $SECONDS"
```

recursion

$ cat rhead.sh

```
cd $1
for file in *
do
        if [ -f $file ]
        then
                echo "============ $file ==============="
                head -5 $file
        fi
        if [ -d $file ]
        then
                /home/unix00/bash/rhead.sh $file
        fi
done
```



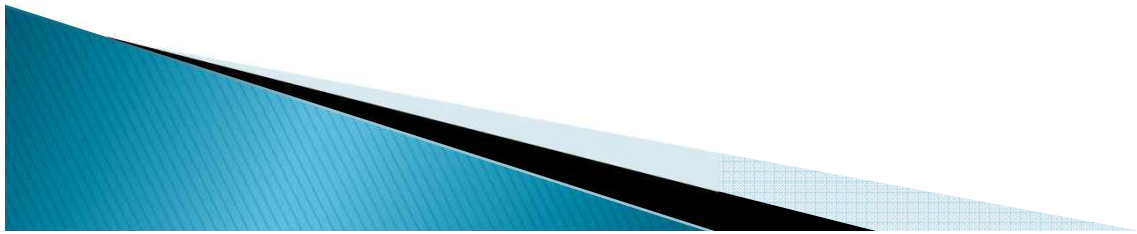
HOMEWORK- whoson.sh

- Take a login name as an argument and report whether or not that user is logged on or not, and if so on which terminal. The output should say something like "billybob is logged from 172.30.92.99" if the user is logged on, and say "billybob is not logged on" if the user is not logged on.

- Some Hints

  who | grep bjgleas

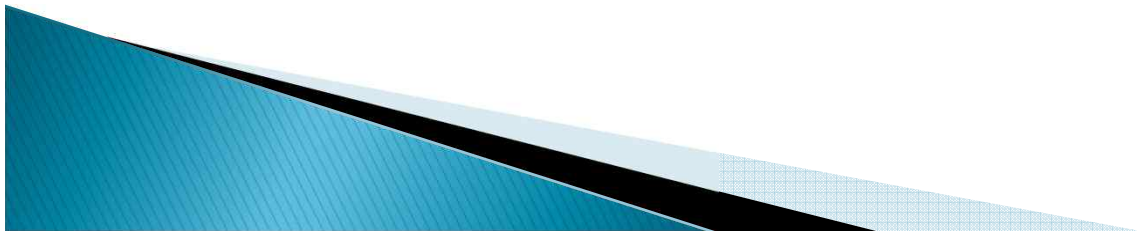
- Exit status ($?) is 0 if successful, 1 if not.



HOMEWORK-hilo.sh

- Ask the user to guess a number between 0 and 59 and guides the user to the correct answer by repeatedly telling the user if the new guess is less than or greater than the number.

- Hint: use the seconds



HOMEWORK – listexe.sh

- Use recursion.
- List all executable files in a certain directory provided as an argument and its subdirectory. If directory is not provided, use current directory.