EECS2031 Software Tools

F 2021

Dec 3,6 Lecture 24

90

CONTENTS

These utilities/commands constitutes basic components for a programming language

- variable (set / get)
- read from the user
- command line arguments
- arithmetic operation
- branching -- if else
- looping -- while / for loop
- enhanced I/O redirection > /dev/null 2>&1 …
- shift
- functions, recursions
- read files

91

YORK UNIVERSITÉ UNIVERSITY

91

Ch. 4. The Bourne Shell

- **CONDITIONAL EXPRESSIONS**

- The control structures often branch based on the value of a logical expression-that is, an expression that evaluates to true or false.

Utility: $ test expression

test returns a zero exit code if expression evaluates to true; otherwise, it returns a nonzero exit status.

```
$ test  3 -eq 3 ;     echo $?      0     true
$ test  3 -eq 33 ;    echo $?      1     false
```

The exit status is typically used by shell control structures for branching purposes. $ if test $x –eq 3 .....

Some **Bourne shells** supports test as a built-in command, in which case they support the second form of evaluation as well.

$ [3 –eq 3] $ if [3 –eq 3]

92

92

Conditions

- Test/check files/directories (easier)
- Compare strings (easier)
- Compare integers

- *Note*: in shells all user inputs are treated as strings.

93

YORK UNIVERSITÉ UNIVERSITY

93

- The brackets of the second from must be surrounded by spaces in order for it to work. $ [3 -eq 4] ⇔ $ test 3 –eq 4

Space!

A test expression may take the following forms:

Form	Meaning
-b filename	True if filename exists as a block special file.
-c filename	True if filename exists as a character special file.
-d filename	True if filename exists as a directory.
-f filename	True if filename exists as an ordinary file
-g filename	True if filename exists as a "set group ID" file.
-h filename	True if filename exists as a symbolic link.
-k filename	True if filename exists and has its sticky bit set.
-p filename	True if filename exists as a named pipe.
-u filename	True if filename exists as a "set user ID" file.
-s filename	True if filename file contains at least 1 char (none empty)
-r filename	True if filename exists as a readable file.
-w filename	True if filename exists as a writeable file.
-x filename	True if filename exists as an executable file.
-e filename	True if filename exists as file or dir bach

Space!

[-d classlist]

test -f classlist

test -s emptyF

Form	Meaning
-n string	True if string contains at least one character.
-z string	True if string contains no characters. empty string
str1 = str2	True if str1 is equal to str2. str1 == str2 bash
str1 != str2	True if str1 is not equal to str2.
int1 -eq int2	True if integer int1 is equal to integer int2.
int1 -ne int2	True if integer int1 is not equal to integer int2.
int1 -gt int2	True if integer int1 is greater than integer int2.
int1 -ge int2	True if integer int1 is greater than or equal to integer int2.
int1 -lt int2	True if integer int1 is less than integer int2.
int1 -le int2	True if integer int1 is less than or equal to integer int2.
! expr	True is expr is False
expr1 -a expr2	True if expr1 and expr2 are both true logical AND
expr1 -o expr2	True if expr1 or expr2 are true logical OR
&& II bash	

test 4 -eq 5

[4 -ne 5]

[4 -gt 5]

((4 <= 5)) bash

one test

multiple tests

Argument	Test is true if . . .
-d *file*	*file* is a directory
-f *file*	*file* is an ordinary file
-r *file*	*file* is readable
-s *file*	*file* size is greater than zero
-w *file*	*file* is writable
-x *file*	*file* is executable
! -d *file*	*file* is not a directory
! -f *file*	*file* is not an ordinary file
! -r *file*	*file* is not readable
! -s *file*	*file* size is not greater than zero
! -w *file*	*file* is not writable
! -x *file*	*file* is not executable
n1 -eq *n2*	integer *n1* equals integer *n2*
n1 -ge *n2*	integer *n1* is greater than or equal to integer *n2*
n1 -gt *n2*	integer *n1* is greater than integer *n2*
n1 -le *n2*	integer *n1* is less than or equal to integer *n2*
n1 -ne *n2*	integer *n1* is not equal to integer *n2*
n1 -lt *n2*	integer *n1* is less than integer *n2*
s1 = *s2*	string *s1* equals string *s2*
s1 != *s2*	string *s1* is not equal to string *s2*

test -d classlist [-d classlist]

test -f classlist [-f classlist]

test -s classlist [-s classlist]

Bash: -e file True if file exists as file or dir

Memorize?

empty file

Bash:

test $n1 -eq $n2 ((n1==n2))

[$n1 -ge $n2] ((n1 >= n2))

[$n1 -gt $n2] ((n1 > n2))

[$n1 -le $n2] ((n1 <= n2))

[$n1 -ne $n2] ((n1 != n2))

[$n1 -lt $n2] ((n1 < n2))

$, space

s1 == s2

97

Ch. 4. The Bourne Shell

- **CONTROL STRUCTURES**

- **The Bourne shell** supports a wide range of control structures that make it suitable as a high-level programming tool.

Shell programs are usually stored in scripts and are commonly used to automate maintenance and installation tasks.

Branch: **if then elif else fi** **case..in..esac**

Loop: **while do done** **for do done** **until do done**

98

98

Ch. 4. The Bourne Shell

- **if…then…fi**

The *if* command supports nested conditional branches and has the following syntax:

```
if list1
then
   list2
elif  list3        optional,.
then                  the elif part may be repeated several times
   list4
else               optional,
   list5              the else part may occur zero times or one time.
fi
```


Ch. 4. The Bourne Shell

- Here's an example of a script that uses an *if* control structure:

```
$ cat numberScript.sh
echo  -n  "enter a number:  "
read num
if test  $num  -lt  0
then
  echo "$num is a negative number"
elif  [ $num  -eq  0 ]
then
  echo "zero"
else
  echo $num is a positive number
fi
```

Read as string but can compare directly

Bash:

```
if (( num < 0))

elif (( num ==0))
```

```
$ numberScript.sh
  enter a number: 1
  1 is a positive number
$ numberScript.sh
  enter a number: -2
  -2 is a negative number
$ numberScript.sh
  enter a number: 0
  zero
$
```

Ch. 4. The Bourne Shell

- Another example of a script that uses an *if* control structure:

```
$ cat week.sh    # version 1
echo  -n  "enter a date:  "
read dat
if  test $dat = "Fri"          #   if [ $dat = Fri ]   compare string is easy
then
  echo "Thank God it is Friday"
elif  [ $dat  = Sat ]
then
  echo "You should not be here working, go home!!! "
elif  [ $dat = "Sun" ]
then
  echo "You should not be here working, go home!!! "
else
  echo "Not weekend yet. Get to work"
fi
```

Comparing strings is easier than comparing integers

```
$ week.sh
enter a date: Wed
Not weekend yet. Get to work
$
```


Ch. 4. The Bourne Shell

- Another example of a script that uses an *if* control structure:

```
$ cat week.sh    # version 3
echo  -n  "enter a date:  "
read dat
if  test $dat = "Fri"          #   if [ $dat = Fri ]   compare string easy
then
  echo "Thank God it is Friday"
elif  [ $dat  = Sat  -o  $dat  = "Sun" ]
then                                   # if test  $dat = sat  -o $dat = Sun
  echo "You should not be here working, go home!!! "
else
  echo "Not weekend yet. Get to work"
fi
```

one test

```
$ week.sh
enter a date: Wed
Not weekend yet. Get to work
$ week.sh
enter a date: Fri
Thank God it is Friday
$
```

Ch. 4. The Bourne Shell

- Another example of a script that uses an *if* control structure:

```
$  cat week.sh     # version 2
echo  -n  "enter a date:  "
read dat
if  test $dat = "Fri"          #   if [ $dat = Fri ]   compare string easy
then
  echo "Thank God it is Friday"
elif  [ $dat  = Sat ] || [ $dat  = "Sun" ]
then                                  # if test  $dat = sat  || test $dat = Sun
  echo "You should not be here working, go home!!! "
else
  echo "Not weekend yet. Get to work"
fi
```

two tests

```
$ week.sh
enter a date: Wed
Not weekend yet. Get to work
$ week.sh
enter a date: Fri
Thank God it is Friday
$
```


Ch. 4. The Bourne Shell

-Here's another example of a script that uses an *if* control structure:
-If a file empty, echo, else ls it

```
$  cat if.sh
echo  -n  "enter a file name:  "
read name
if [ ! –s $name ]      # if test  ! –s $name
then
  echo "File $name is empty"
  exit 1
else
  ls –l $name
fi
```

Argument	Test is true if . . .
-d *file*	*file* is a directory
-f *file*	*file* is an ordinary file
-r *file*	*file* is readable
-s *file*	*file* size is greater than zero
-w *file*	*file* is writable
-x *file*	*file* is executable

```
$ cat > emptyF
^D
$ touch  emptyF2
```

Ch. 4. The Bourne Shell

- Here's another example of a script that uses an *if* control structure

```
$ cat if2.sh
echo  -n  "enter a file name:  "
read name
if [ –d $name ]     # if test –d $name
then
  echo $name is a directory
  ls –ld $name
elif [ -x $name ]
then
  echo "File $name is executable"
else
  echo "File $name is not executable"
  chmod u+x $name
  echo "File $name is executable now"
fi
```

Argument	Test is true if . . .
-d *file*	*file* is a directory
-f *file*	*file* is an ordinary file
-r *file*	*file* is readable
-s *file*	*file* size is greater than zer
-w *file*	*file* is writable
-x *file*	*file* is executable

Ch. 4. The Bourne Shell

- Here's an example of a script that uses an *if* control structure: improved version of grep

```
$ cat mygrepNotFound.sh
echo –n "Please enter file to search: "
read file                          # read two variables.
echo –n "Please enter search key: "
read pattern
egrep $pattern $file
```

Q: How to check if grep found or not programmatically
A: Exit Code of egrep execution!

```
$ mygrepNotFound.sh  Leung classlist
  pattern Leung not found in file classlist, try another
```

Ch. 4. The Bourne Shell

- Here's an example of a script that uses an *if* control structure: improved version of grep

```
$ cat mygrepNotFound.sh
  echo –n "Please enter file to search: "
  read file                          # read two variables.
  echo –n "Please enter search key: "
  read pattern
  egrep $pattern $file
  if [ $?  -ne 0 ]          #  not 0  --- not successful  [ $?  -gt 0 ]
  then
    echo pattern $pattern not found in file $file, try another
  else
    echo pattern $pattern is found in file $file
  fi

$ mygrepNotFound.sh  Leung classlist
  pattern Leung not found in file classlist, try another
```

YORK UNIVERSITÉ UNIVERSITY

111

Ch. 4. The Bourne Shell

- Here's an example of a script that uses an *if* control structure: improved version of grep

```
$ cat mygrepNotFound.sh        # list the script.
  echo –n "Please enter file to search: "
  read file                          # read two variables.
  echo –n "Please enter search key: "
  read pattern
  egrep $pattern $file
  if [ $?  -eq 0 ]          # 0  --- successful
  then
    echo pattern $pattern is found in file $file
  else
    echo pattern $pattern not found in file $file, try another
  fi

$ mygrepNotFound.sh  Leung classlist
  pattern Leung not found in file classlist, try another
```

Turn off Raw output?

112

Ch. 4. The Bourne Shell

- **CONTROL STRUCTURES**

- **The Bourne shell** supports a wide range of control structures that make it suitable as a high-level programming tool.

Shell programs are usually stored in scripts and are commonly used to automate maintenance and installation tasks.

Branch: **if then elif else fi** **case..in..esac**

Loop: **while do done** **for do done** **until do done**

Ch. 4. The Bourne Shell

- **while…do…done**

- The *while* command executes one series of commands as long as another series of commands succeeds.

Here's its syntax:

```
while  list1
do
   list2
done
```

The *while* command executes the commands in *list1* and ends if the last command in *list1* fails;
otherwise, the commands in *list2* are executed and the process is repeated.

Ch. 4. The Bourne Shell

- If list2 is empty, the do keyword should be omitted.
  A **break** command causes *the loop to end immediately*, and a **continue** command causes *the loop to immediately jump to the next iteration*.

- Here's an example of a script that uses a while control structure to generate hello several times:

$ cat repeat0.sh

```
count=0
while [ $count -lt 5 ]
do
  echo Hello  $count
  count=`expr  $count + 1`
done
```

```
$ repeat0.sh
Hello  0
Hello  1
Hello  2
Hello  3
Hello  4
$
```

115 Bash: ((count < 5))
count=$((count+1)) or ((count=count+1)) or ((count++))


```
int main(int argc, char *argv[])
{
    printf("Hello, world\n");
    int max = atoi(argv[1]);
    int count =1;
    int sum = 0;
    while( count <=max){
        sum = sum + count;
        count = count +1;
    }
    printf("Sum of 0 to %d is %d", max, sum);
}
```

$ cat mySum.sh

```
max=$1
count=1
sum=0
while [ $count -le $max ]
do
  sum=`expr $sum + $count`
  count=`expr  $count + 1`
done
echo Sum of 0 to $max is $sum.
```

while ((counmt <= max))

sum= $((sum + count))
count=$((count+1)) ((count++))

```
$ mySum.sh  4
  Sum of 0 to 4 is 10
$ mySum.sh  6
  Sum of 0 to 6 is 21
```

117

Ch. 4. The Bourne Shell

- Here's an example of a script that uses a while control structure to generate hello several times:

```
$ cat repeat.sh
read word
while [ $word  !=  "quit" ]
do
  echo $word
  read word      # read again
done
```

```
$ repeat.sh
Hello
Hello
The
The
World
World
quit
$
```

Terminate at EOF?

Example: Loop + branch

```
#! /bin/sh
echo -n "enter a number or 'quit': "
read num

while [  $num  != "quit"  ]
do
      if    [ $num  -lt  0 ]
      then
         echo "$num is a negative number"
      elif  [ $num -eq 0  ]
      then
         echo "this is zero"
      else
         echo "$ num is a positive number"
      fi

   # read again
    echo -n "enter a number or 'quit': "
    read num

done
```


if ((num < 0))

elif ((num ==0))

```
$ loopbranch.sh
  enter a number or 'quit': 1
     1 is a positive number
  enter a number or 'quit': -2
     -2 is a negative number
  enter a number or 'quit': 0
       this is zero
  enter a number or 'quit': quit
$
```

Ch. 4. The Bourne Shell

Write a script matrix.sh to generate a matrix

$1

```
$ matrix.sh  7
1-1   1-2   1-3   1-4   1-5   1-6   1-7
2-1   2-2   2-3   2-4   2-5   2-6   2-7
3-1   3-2   3-3   3-4   3-5   3-6   3-7
4-1   4-2   4-3   4-4   4-5   4-6   4-7
5-1   5-2   5-3   5-4   5-5   5-6   5-7
6-1   6-2   6-3   6-4   6-5   6-6   6-7
7-1   7-2   7-3   7-4   7-5   7-6   7-7
$
$ matrix.sh  4
1-1   1-2   1-3   1-4
2-1   2-2   2-3   2-4
3-1   3-2   3-3   3-4
4-1   4-2   4-3   4-4
```

```
C, Java, C++:
a = atoi(argv[1])
x = 1
while  (x < a)
{
   y = 1
   while (y < a)
  {
      printf  ("%d-%d",x,y)
      y = y+1
   }
   x = x + 1
}
```


Ch. 4. The Bourne Shell

```
$ matrix.sh  4
1-1   1-2   1-3   1-4
2-1   2-2   2-3   2-4
3-1   3-2   3-3   3-4
4-1   4-2   4-3   4-4
```

$ cat matrix.sh

or, if $# - eq 0
better $# - ne 1

```
if [  -z $1 ];  then     # if $1 is empty string  -z
    echo  "Usage: matrix number"
    exit
fi
x=1
while [ $x -le $1 ]        # outer loop
do
   y=1
   while [ $y -le $1 ]
   do
      echo  -n "$x-$y       "
      y=`expr  $y + 1`          # y++  update inner-loop count
   done
   echo
   x=`expr $x + 1`              # x++  update inner-loop count
done
```

Indent not mandatory

```
x = 1
while  (x < 7){
   y = 1
   while (y < 7){
      printf( "%d-%d", x,y)
       y = y+1
   }
   x = x + 1
}
```

For your information

- while true, break, continue, and :

```
#!  /bin/sh
echo   menu  test program

while  :    #  while true
do
  echo -n  'your choice? '     # prompt.
  read reply                   # read response.
  if [  $reply = q ]
  then
    break
  fi
  .....

done
```

```
#!  /bin/sh
x=0
while true
do
  x=`expr $x + 1`
  if [ $x -eq 100 ]
  then
    break
  else
    :      # continue
  fi
done
echo $x
```

100

CONTENTS

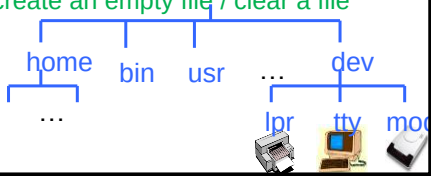
These utilities/commands constitutes basic components for a programming language

- variable (set / get)
- read from the user
- command line arguments
- arithmetic operation
- branching -- if else
- looping -- while / for loop
- enhanced I/O redirection > /dev/null 2>&1 …
- shift
- functions, recursions
- read files

Last topic: enhanced I/O redirection, /dev/null - bit bucket, black hole, Dave Null

- Special file found in device directory (**/dev**)
- Discard data written to (but report writing succeed
  - Write to it: everything disappears (absorbed)
- Provide no data to reading process (EOF immediately)
  - Read from it: get nothing

```
ls –l   >  /dev/null          # standard output  suppressed
who > /dev/null
cat   /dev/null > emptyF      # create an empty file / clear a file
cat < /dev/null > emptyF      # create an empty file / clear a file
```

127

127

Ch. 4. The Bourne Shell

- Here's an example of a script that uses an *if* control structure: improved version of grep

```
$  cat mygrepNotFound.sh        # list the script.
  echo –n "Please enter file to search: "
  read file                     # read two variables.
  echo –n "Please enter search key: "
  read pattern
  egrep $pattern  $file
  if [ $?  -ne 0 ]          #  not 0  --- not successful  [ $?  gt 0 ]
  then
    echo pattern $pattern not found in file $file, try another
  else
    echo pattern $pattern is found in file $file
  fi
```

```
$  mygrepNotFound.sh  Wang classlist
  ....
  pattern Wang is found in file classlist
```

How to turn off row output from egrep

128

128

Ch. 4. The Bourne Shell

- Here's an example of a script that uses an *if* control structure: improved version of grep

```
$  cat mygrepNotFoundDavNull.sh        # list the script.
  echo –n "Please enter file to search: "
  read file                           # read two variables.
  echo –n "Please enter search key: "
  read pattern
  egrep $pattern  $file  > /dev/null
  if [ $?  -ne 0 ]          #  not 0  --- not successful  [ $?  gt 0 ]
  then
    echo pattern $pattern not found in file $file, try another
  else
    echo pattern $pattern is found in file $file
  fi

$  mygrepNotFoundDavNull.sh  Wang classlist
   pattern Wang is found in file classlist
```


Do it

This concludes the topics on Unix

SUMMARY

- **These utilities/commands constitutes basic components for a programming language**
  - variable (set / get) — x=4 x=hello echo $x
  - read from the user — read x
  - command line arguments — $0, $1-9, $#, $*
  - arithmetic operation — x=`expr 3 + 4` x=$((3 + 4))
  - branching -- if then else fi — if test $x -lt 4 then if [$x -lt 4] then
  - looping -- while / for loop — while .. do .. done
  - enhanced I/O redirection — > /dev/null 2>&1
  - parameter shifting — shift shift 3
  - functions, recursions …

Now see 2 examples