CHAPTER 4
THE BOURNE SHELL

The start up file named “.profile” is run as you login to Bourne shell. You may change its content as needed.

$ cat .profile
TERM=vt100   #set terminal type
export TERM #copy to environment
ssty erase "^?" kill "^U" intr "^C" eof "^D" #set metacharacters
path=".:/bin:/home122/bin" #set path

CREATING /ASSIGNING A VARIABLE

A variable is created and/or assigned value using the following syntax:

{name=value}+

$ firstName=Ugur lastName=Halici
$ echo $firstName $lastName
$ name=Ugur Halici         ... syntax error
Halici: not found
$ name="Ugur Halici"
$ echo $name
Ugur Halici

ACCESSING A VARIABLE

Variables can be accessed in several ways as explained below.

$name          : replaced by the value of the variable name
${name}        : same as above
${name-word}   : replaced by the value of name if it was set, and word otherwise
${name+word}   : replaced by word if name was set, and nothing otherwise
${name=word}   : assigns word to name if it was not already set, and then is replaced by the value of name
${name?word}   : replaced by name if name was set, word is displayed at the standard error channel. If word is omitted, then the standard error message is displayed instead.
$ verb=sing
$ echo I like $verbing  ... there is no variable named verbing
I like
$ echo I like ${verb}ing
$

$ startDate=${startDate-`date`} #if not set, assign by `date`
$ echo $startDate
Tue May 12
$

$ echo x=${x=10}
x=10
$ echo $x
10
$

$ flag=1
$ echo ${flag+'flag is set'}
flag is set
$ echo ${flag2+'flag2 is set'}  ... nothing to be displayed
$

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

$ cat script.sh
value=${grandTotal?'grand total is not set'}
echo done
$ script.sh
grandTotal: grand total is not set
$

Note that, script terminated when the access error occurred, so “done” is not displayed

READING A VARIABLE FROM STANDARD INPUT

read  {variable}+

reads one line from standard input and then assigns successive words from the line to the specified variables.
$ cat script.sh
echo -n "please enter your name:"
read name
echo your name is $name
$script.sh
please enter your name: ugur
your name is ugur
$script.sh
please enter your name: ugur halici
your name is ugur halici

$ cat script1.sh
echo -n "please enter your name: "
read firstname lastname #read two variables
echo your firstname is $firstname
echo your lastname is $lastname
$script1.sh
please enter your name: Ugur Halici
echo your firstname is Ugur
echo your lastname is Halici
$script1.sh
please enter your name: Ugur
echo your firstname is Ugur
echo your lastname is
$script1.sh
please enter your name: Ayse Nese Can
echo your firstname is Ayse
echo your lastname is Nese Can

EXPORTING VARIABLES

export {variable}* 

The export command marks the specified variables for export to the environment. If no variables are specified, a list of all the variables marked for export during the shell session is displayed.

env {variable=value}* [command]

The env command assigns values to specified environment variables, and then executes an optional command using the new environment. If no variables or command are specified, a list of the current environment variables is displayed.

$ export
export term
$ DATABASE=/dbase/db
$ export DATABASE
$ export
export DATABASE
export TERM
$ env
DATABASE=dbase/db
HOME=/home122/halici
LOGNAME=halici
PATH=.:/bin:/home122/bin
SHELL=/bin/sh
TERM=vt100
USER=halici
$ sh # create a new shell
$ echo $DATABASE
/dbase/db
$ ^D # terminate subshell
$

PREDEFINED LOCAL VARIABLES

Some predefined local variables in Bourne shell having special meaning are listed below:

$@ : an individually quoted list of all the positional parameters
$# : the number of positional parameters
$? : the exit value of the last command
$! : the process id of the last background command
$- : the current shell option assigned from the command line

$ cat script.sh
echo there are $# command line arguments: $@
$ script.sh nofile tmpfile
there are 2 command line arguments: nofile tmpfile

$ sleep 1000 &
29452
$ kill 29452
29452 terminated
$ sleep 1000 &
29455
$ kill $!
29455 terminated
$ echo $!
29455 ...
... the process id still remembered
ARITHMETIC

`expr  expression`

The command `expr` evaluates `expression` and sends the result to the standard output. All of the components of the expression must be separated by blanks, and all of the shell metacharacters must be escaped by a `\`. In an expression the following operators may be used.

```
\* / % : the number of positional parameters
+ - : the exit value of the last command
= \> \>= \< \<= != : the comparison operators
\& : logical “and”
\| : logical “or”
```

Escaped parantheses `\(` and `\)` may be used to explicitly control the order of evaluation.

```
$ x=1
$ x=`expr $x + 1`
$ echo $x
2
$ x=`expr 2 + 3 \* 5`
$ echo $x
17
$ echo `expr \( 4 \> 5 \)` # Is 4>5 ?
0
$ echo `expr \( 4 \> 5 \) \| \( 6 \< 7 \)` # Is 4>5 or 6<7 ?
1
```

The following operators may be used in expressions related to strings.

```
string : regularExp : returns the length of string if both sides match, returns 0 otherwise
match string regularExp : same as the previous one
sbstr string start length : returns the substring of string starting from start and consisting of length characters
index string charlist : returns index of the first character in string that appears in charlist
length string : Returns length of string
```

Above `regularExp` is regular expression in which

```
. : matches any single character
[...] : matches any of the single character enclosed in brackets
[^...] : matches any of the single character which is not enclosed in brackets
* : zero or more occurrences of the character that precedes it
```
$ echo `expr length "cat"`
3
$ echo `expr substr "monkey" 4 3`
key
$ echo `expr match "transputer" ".*lk"`
0
$ echo `expr match "smalltalk" ".*lk"`
9
$ echo `expr match "smalltalk" "a*lk"`
0

CONDITIONAL EXPRESSIONS

The utility `test` returns a 0 exit status if the given expression evaluates to true; it returns a non-zero exit status otherwise. The exit status of the test command is typically used by the shell control structures for branching purposes. The syntax is as follows:

`test expression`

or equivalently the following may be used instead of the above form

`[expression]`

The expression may be written in the following forms

- `str1=str2` : true if `str1` is equal to `str2`
- `str1!=str2` : true if `str1` is not equal to `str2`
- `string` : true if `string` is not null
- `int1 -eq int2` : true if `int1` is equal to `int2`
- `int1 -ne int2` : true if `int1` is not equal to `int2`
- `int1 -gt int2` : true if `int1` is greater than `int2`
- `int1 -ge int2` : true if `int1` is greater or equal to `int2`
- `int1 -lt int2` : true if `int1` is less than `int2`
- `int1 -le int2` : true if `int1` is less or equal to `int2`
- `!expr` : true if `expr` is false
- `expr1 -a expr2` : true if `expr1` and `expr2` are both true
- `expr1 -o expr2` : true if `expr1` or `expr2` is true
- `(expr)` : escaped parantheses are used for grouping expressions
CONTROL STRUCTURES

**while ... do ... done**

The `while` command executes the commands in `list2` as long as the last command in `list1` succeeds.

```bash
while list1
do
  list2
done
```

```bash
$ cat multish
x=1
do
  y=1
  while [ $x -le $1]
do
    echo -n `expr $x \* $y` " 
    y=`expr $y + 1`
done
echo
  x=`expr $x + 1`
done
$ cat multish
x=1
do
  y=1
  while [ $x -le $1]
do
    echo -n `expr $x \* $y`
    y=`expr $y + 1`
done
echo
  x=`expr $x + 1`
done
$ multi.sh 4
1 2 3 4
2 4 6 8
3 6 9 12
4 8 12 16
$ 
```

The following commands can be used to control loops

- **break**: causes the loop to end immediately
- **loop**: causes the loop jump immediately to the next iteration

**until ... do ... done**

The `until` command executes the commands in `list2` as long as the last command in `list1` fails.

```bash
until list1
do
  list2
done
```

```bash
$ cat until.sh
x=1
until [ $x -gt ] 3
do
```
```
  echo x=$x
  x=`expr $x+1`
done
$ until.sh
x=1
x=2
x=3
$

```case . in . esac```

The case command supports multi-way branching based on the value of a single string and has the following syntax:

```
case expression in
  pattern{ | pattern}* 
  list
  ;;
esac
```

```
$ cat menu.sh
#!/bin/sh
echo menu test program
stop=0
while test $stop –eq 0
do
cat <<ENDOFMENU
1: print the date
2,3: print the current working directory
4: exit
ENDOFMENU
echo
echo –n ’your choice?’
read reply
echo
case $reply in
  ”1”)
  date
  ;;
  ”2”|”3”)
pwd
  ;;
  ”4”)
stop=1
  ;;
  *)
echo illegal choice
  ;;
esac
done
```
**for ... do ... done**

The `for` command allows a list of commands to be executed several times, using a different value of the loop variable during each iteration.

```bash
for name [in {word}]*
do
  list
done
```

The `for` command loops the value of the variable `name` through each word in the word list, evaluating the commands list after each iteration. If no word list is supplied, `$@` (i.e. all positional parameters) is used instead.

```bash
$ cat for.sh
for color in red yellow blue
do
  echo one color is $color
done
$ for.sh
one color is red
one color is yellow
one color is blue
```

**if ... then ... fi**

The `if` command supports nested conditional branches, and has the following syntax

```bash
if list1
then
  list2
elif
  list3 ... optional, elif part can be repeated several times
then
  list4
else
  list5 ... else part may be omitted
fi
```

```bash
$ cat if.sh
echo –n ‘enter a number:’
read number
if [ $number -lt 0]
then
  echo negative
elif [ $number -eq 0]
```
then
  echo zero
else
  echo positive
fi
$ if.sh
enter a number: 1
positive
$ if.sh
enter a number: -1
negative
$

trap

The `trap` command allows you to specify a `command` that should be executed when the shell receives a `signal` of a particular type. The syntax is as follows:

```
trap [ [command] {signal}+ ]
```

$ cat trap.sh
```
trap 'echo control-C is pressed;exit 1' 2 #trap control-C signal 2
while [1 -eq 1]
do
  echo infinite loop
  sleep 3
done
```

$ cat trap.sh
```
trap 'echo control-C is pressed;exit 1' 2 #trap control-C signal 2
while [1 -eq 1]
do
  echo infinite loop
  sleep 3
done
```

$ trap.sh
infinite loop
^C
cotrol-C is pressed
$