Chapter 5
The Bourne Shell

YUAN LONG

CSC 3320 SYSTEM LEVEL PROGRAMMING

FALL 2016

Updated based on original notes from Raj Sunderraman and Michael Weeks

What will be covered?

- Variable
- Read input
- Arithmetic expression
- If…else
- Case structure
- For loop/while loop

Creating/Assigning a Variable

- Name=value
  - Variable created if it does not exist
  - Need quotes if value contains spaces
  - E.g. x=”one two”
- Access variable with $ in front
  - $x or **${x}**

Brace must be used when variable followed by some alphabetic numerical characters

```
$ x="one two"
$ echo $x
one two
$ echo ${x}three
one twothree
```

Reading from Standard Input

- Command *read*
  - Reads 1 line
  - Assign successive words to the specified variables
  - Examples

```
$ read v1 v2
one two
$ echo $v1
one
$ echo $v2
two
```

```
$ read v1 v2
one two three four
$ echo $v1
one
$ echo $v2
two three four
```

Any words left over are assigned to the last variable

Example - Reading Multiple Lines


```
$ cat readme.sh
#!/bin/bash
# read multiple lines

read v1
read v2
echo "you said $v1"
echo "then you said $v2"
```

```
$ ./readme.sh
one two
three four five
you said one two
then you said three four five
```

Input for v1

Input for v2

Exporting Variables

- Command *export*
- Makes variables available in environment
- e.g. *export x*

Change to another shell

```
$ v1="one two"
$ export v1
$ sh
sh-3.1$ echo $v1
one two
sh-3.1$
```

Predefined Locals


```
$ cat predefined.sh
echo You passed $# parameters.
echo These are: "$@"
echo process ID of last background process = $!
echo process ID of this shell = $$
notAcommand
echo Last command returned with $? as the status.
```

```
$ ./predefined.sh one two three four
You passed 4 parameters.
These are: one two three four
process ID of last background process =
process ID of this shell = 21712
./predefined.sh: line 7: notAcommand: command not
found
Last command returned with 127 as the status.
```

Arithmetic

- Bourne shell does not directly do math
- Command ***expr*** evaluates expressions
  - Supports
    - Multiplication (*), Division (/), Remainder (%)
    - Add (+), Subtract (-)
    - Equal (=), Not Equal (!=)
    - Less (<), Less/Eq (<=), Greater (>), Greater/Eq (>=)
    - And (&), Or (|)
    - Index (locate substring)
    - Match **Basic regular expression**

expr Command

- *expr* also evaluates expressions
  - Locate substring
    - expr index string charList
      - E.g. $expr index “donkey” “ke”
      - $4
  - Test for a match (returns 0 or length)
    - expr match string regExp
      - E.g. $expr match “donkey” “ke”
      - $0
    - expr string : regExp
      - E.g. $expr “donkey” “donkey”
      - $6
  - Length of string
    - expr length string
      - E.g. $expr length “cat”
      - $3

Example - Arithmetic

- $x=1 **x=1**
- $x=`expr $x + 1` **x=x+1**
- $echo $x **Print out value of x, which is 2**
- $x=`expr 2 + 3 \* 5` **x=2+3*5**
- expr "swimming" : 'sw.*ing' **Attempt a match**

Note:
All of the components of expression must be **separated by blanks**.
All of the shell metacharacters must be **escaped by backslash** \ .

test Command

- Command ***test*** *expression* OR just *expression*
  - Returns 0 if true
  - Returns nonzero if false
- Examples
  - File exists: ***-e filename***
  - Strings are equal: ***str1 = str2***
  - Two interges not equals: ***int1 –ne int2***
- See page 193 for a more complete list or http://wiki.bash-hackers.org/commands/classictest

Example – test command

- Check if file CSc_course.txt exits in ~/public

```
$test -e  ~/public/CSc_course.txt
$echo $?
```

Check Return value

- Check if string matches

```
$test "donkey"  =  "ke"
$echo $?
```

Remember to put spaces between each component

- Check if two integer not equals

```
$test 23 -ne 3
$echo $?
```

If .. Then

- Execute *list1*
- If last command succeeds, do *list2*
- If last command (of *list1*) fails, try *list3*, etc.

```
if list1
then
   list2
elif list3
then
   list4
else
   list5
fi
```

If .. Then Example


```
$ cat testif.sh
echo "enter a word: "
read v1
if [ -e $v1 ]
then
  echo "A file by that name exists."
else
  echo "No file by that name exists."
fi
```

You can replace it with
if test -e $v1

```
$ ./testif.sh
enter a word:
CSC_Course.txt
A file by that name exists.
$ ./testif.sh
enter a word:
two
No file by that name exists.
```

Case Structure


```
case expression in
  pattern1)
    list
    ;;
  pattern2)
    list2
    ;;
...
  *) # default
    list_n
    ;;
esac
```

Example – Case Structure


```
$ cat testcase.sh

echo "Type out the word for 1
or 2:"
read v1
case $v1 in
  [Oo]ne)
    echo "You entered 1"
    ;;
  [Tt]wo)
    echo "You entered 2"
    ;;
  *)
    echo "sorry"
    ;;
esac
```

```
$ ./testcase.sh

Type out the word for 1 or 2:
two
You entered 2
$ ./testcase.sh
Type out the word for 1 or 2:
Two
You entered 2
$ ./testcase.sh
Type out the word for 1 or 2:
three
Sorry
```

For Loop

- Loop where *name* gets each value in *word*, in turn
- Uses *$@* if no word given
- End loop: *break*
- Go to next iteration: *continue*

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

Example – For loop


```
$ cat testfor.sh

params=$@
for value in $params
do
  echo param: $value
done
```

```
$ ./testfor.sh one two three
param: one
param: two
param: tree
```

While Loop

- Execute *list2* as long as the last command of *list1* succeeds
- End loop: *break*
- Go to next iteration: *continue*

```
while  list1
do
   list2
done
```

Example - While Loop Example


```
$ cat testwhile.sh

x=1
while [ $x -lt 4 ]      #While X<4
do
  echo x = ${x}, less than four
  x=`expr $x + 1`
done
```

```
$ ./testwhile.sh
x = 1, less than four
x = 2, less than four
x = 3, less than four
```

Until .. do .. done

- Keep doing *list1* until its last line works
- Otherwise, do commands in *list2*
- End loop: *break*
- Go to next iteration: *continue*

```
until list1
do
   list2
done
```

Example - Until .. do .. done


```
$ cat testuntil.sh

x=1
until [ $x -gt 3 ]    # While X>3 fails
do
  echo x = $x
  x=`expr $x + 1`
done
```

```
$ ./testuntil.sh
x = 1
x = 2
x = 3
```

Review

- Variable assignment and access
- Reading standard input
- Arithmetic and pattern matching (expr)
- Control structures (case, for, if, while, until)

Example – test command

- Check if file CSc_course.txt exits in ~/public

```
if test -e  ~/public/CSc_course.txt
then
        echo " CSc_course.txt in ~/public "
fi


if [ -e  ~/public/CSc_course.txt ]
then
        echo " CSc_course.txt in ~/public "
fi
```