Southern Connecticut State University

SCSU

CSC424 System Administration

Instructor: Dr. Hao Wu

Week 8 Bash Scripting - 3

Outline

- Filename Matching
- Regular Expression
- Linux Text Processing Tools
- Variables
- Arithmetic
- **Control Flow**
- **Loops**
- **Function**

Conditional Statements

- Conditionals let us decide whether to perform an action or not, this decision is taken by evaluating an expression. The most basic form is:

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

- the elif (else if) and else sections are optional
- Put spaces after [and before], and around the operators and operands.

Expressions

- An expression can be: String comparison, Numeric comparison, File operators and Logical operators and it is represented by [expression]:
- String Comparisons:
  - =   compare if two strings are equal
  - !=   compare if two strings are not equal
  - -n   evaluate if string length is greater than zero
  - -z   evaluate if string length is equal to zero
- Examples:
  - [s1 = s2]   (true if s1 same as s2, else false)
  - [s1 != s2]   (true if s1 not same as s2, else false)
  - [s1]   (true if s1 is not empty, else false)
  - [-n s1]   (true if s1 has a length greater then 0, 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 n1greater 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
```

Examples

```
$ 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
  - -s check if a file has a length greater than 0
  - -w check if write permission is set for a file or directory
  - -x check if execute permission is set for a file or directory

Expressions

- 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)
  - [-w fname]	(true if fname has the write permission, else false)
  - [-x fname]	(true if fname has the execute permission, else false)

Example

```
#!/bin/bash
echo "Enter a filename: "
read filename
if [ ! –r "$filename" ]
 then
   echo "File is not read-able"
 exit 1
fi
```

Example

```
#!/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
```

Exercise

- Write a shell script which:
  - accepts a file name
  - checks if file exists
  - if file exists, copy the file to the same name + .bak + the current date (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.

Expressions

- Logical operators:
  - ! negate (NOT) a logical expression
  - && logically AND two logical expressions
  - || logically OR two logical expressions

Example: Using the ! operator

```
#!/bin/bash

read -p "Enter years of work: " Years
if [ ! "$Years" -lt 20 ]; then
   echo "You can retire now."
else
   echo "You need 20+ years to retire"
fi
```

Example: Using the && operator

```
#!/bin/bash

Bonus=500
read -p "Enter Status: " Status
read -p "Enter Shift: " Shift
if [[ "$Status" = "H" && "$Shift" = 3 ]]
then
   echo "shift $Shift gets \$$Bonus bonus"
else
   echo "only hourly workers in"
   echo "shift 3 get a bonus"
fi
```

Example: Using the || operator

```
#!/bin/bash

read -p "Enter calls handled:" CHandle
read -p "Enter calls closed: " CClose
if [[ "$CHandle" -gt 150 || "$CClose" -gt 50 ]]
  then
  echo "You are entitled to a bonus"
else
  echo "You get a bonus if the calls"
  echo "handled exceeds 150 or"
  echo "calls closed exceeds 50"
fi
```

Case Statement

- Used to execute statements based on specific values. Often used in place of an if statement if there are a large number of conditions.
- Value used can be an expression
- each set of statements must be ended by a pair of semicolons;
- a *) is used to accept any value not matched with list of values

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

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 l 10) echo “wrong number.”;;
        *) echo “Unrecognized value.”;;
  esac
```

Loop Statements

- The for structure is used when you are looping through a range of variables.

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

- statements are executed with var set to each value in the list.

Example: for loop

```
#!/bin/bash
 let sum=0
 for num in 1 2 3 4 5
    do
      let “sum = $sum + $num”
    done
 echo $sum
```

```
#!/bin/bash
 for x in paper pencil pen
  do
   echo “The value of variable x is: $x”
   sleep 1
 done
```

Iteration Statements

- if the list part is left off, var is set to each parameter passed to the script ($1, $2, $3,…)

```
$ 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
```

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

- 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
```

- First, the arithmetic expression EXPR1 is evaluated. EXPR2 is then evaluated repeatedly until it evaluates to 0. Each time EXPR2 is evaluates to a non-zero value, statements are executed and EXPR3 is evaluated.

Example: A C-like for loop

```
$ 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

- The while structure is a looping structure. Used to execute a set of commands while a specified condition is true. The loop terminates as soon as the condition becomes false. If condition never becomes false, loop will never exit.

```
while expression
do
     statements
done
```

While Statements

```
$ 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 + $i”
      i=$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 I d) date ;;
         W I w) who ;;
         P I p) pwd ;;
         Q I q) loop=n ;;
         *) echo “Illegal choice.” ;;
    esac
    echo
  done
```

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

- The continue command causes a jump to the next iteration of the loop, skipping all the remaining commands in that particular loop cycle.

```
$ 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

- The break command terminates the loop (breaks out of it).

```
$ 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
```

Until Statements

- The until structure is very similar to the while structure. The until structure loops until the condition is true. So basically it is “until this condition is true, do this”.

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

Until Statements

```
$ 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 !
```

Functions

- Functions make scripts easier to maintain. Basically it breaks up the program into smaller pieces. A function performs an action defined by you, and it can return a value if you wish.

```
#!/bin/bash
 hello()
 {
 echo “You are in function hello()”
 }

echo “Calling function hello()…”
 hello
 echo “You are now out of function hello()”
```

- In the above, we called the hello() function by name by using the line: hello . When this line is executed, bash searches the script for the line hello(). It finds it right at the top, and executes its contents.

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: 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" ll ! -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 l bc -l`
echo "Diamonds: " `echo ${hits[1]}*100/$N l bc -l`
echo "Hearts: " `echo ${hits[2]}*100/$N l bc -l`
echo "Spades: " `echo ${hits[3]}*100/$N l bc -l`
echo "======================================="
echo "Execution time: $SECONDS"
```