Mahasarakham University

มหาวิทยาลัยมหาสารคาม

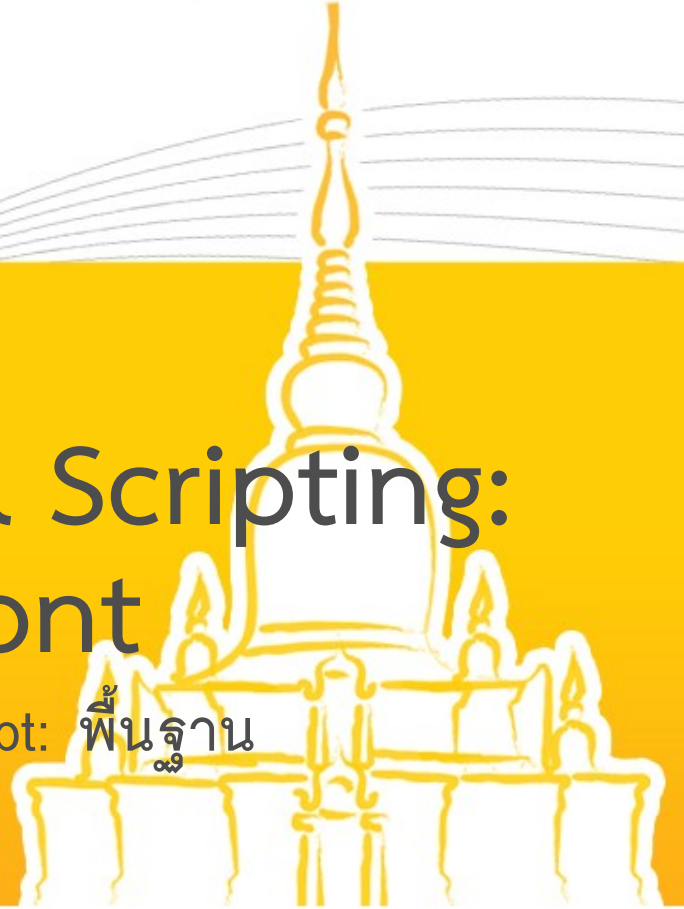
Heart of
the Northeast

www.msu.ac.th

Mahasarakham University
มหาวิทยาลัยมหาสารคาม

Introduction to Shell Scripting: The Basics: cont

แนะนำการทำงานของ Shell Script: พื้นฐาน



Heart of the Northeast

www.msu.ac.th

โดย อาจารย์โอฬาริก สุรินต๊ะ

Jan 18, 2017

Agenda

- While loop
- Nested while and for loop
- Basic Functions
- I/O redirection
- Piping

While Loop

- What is it?
  - The while construct allows for repetitive execution of a list of commands, as long as the command controlling the while loop executes successfully (exit status of zero).
  - Syntax:

```
while [ condition ]
do
    command1
    command2
    command3
done
```

While Loop

- Command 1 to 3 will be executed repeatedly till condition is **true**.
- The argument for a while loop can be any **boolean expression**.

What is boolean expression?

- **Infinite loops** occur when the conditional never evaluates to **false**.

While Loop: sample

$ sudo nano while-loop.sh

```
x=1
while [ $x -le 5 ]
do
   echo "Welcome $x times"
   x=$(( $x + 1 ))
done
```

!!! หากต้องการใช้คำสั่งนี้ในการ execu โปรแกรมจะต้องทำอย่างไร

$ ./while-loop.sh

$ bash while-loop.sh

While Loop: sample

$ sudo nano while-loop.sh

```
#!/bin/bash
x=1
while [ $x -le 5 ]
do
   echo "Welcome $x times"
   x=$(( $x + 1 ))
done

$ chmod +x while-loop.sh
$ bash while-loop.sh
```

While Loop: sample

$ sudo nano factorial.sh

```
#!/bin/bash

counter=$1
factorial=1
while [ $counter -gt 0 ]
do
    factorial=$(( $factorial * $counter ))
    counter=$(( $counter - 1 ))
done
echo $factorial
```

!!! หากต้องการ execu
โปรแกรมจะต้องทำอย่างไร

While Loop: sample

```
$ bash factorial.sh
```

or

```
$ chmod +x factorial.sh
$ ./factorial.sh
```

Do you still get any problem?

While Loop: sample

$ bash factorial.sh

or

$ ./factorial.sh

factorial.sh: line 5: [: -gt: unary operator expected

Becasue of the argument
counter=$1

$./factorial.sh 5

120

While & case

```
#!/bin/bash
while(true)
do
    clear
    printf "Choose from the following operations: \n"
    printf "[a]ddition\n[b]Subtraction\n[c]Multiplication\n[d]Division\n"
    printf "#####################################\n"
    read -p "Your choice: " choice
    case $choice in
    [aA])
        read -p "Enter first integer: " int1
        read -p "Enter second integer: " int2
        res=$(($int1+$int2))
    ;;
```

While & case

```
[bB])
        read -p "Enter first integer: " int1
        read -p "Enter second integer: " int2
        res=$(($int1-$int2))
;;
[cC])
        read -p "Enter first integer: " int1
        read -p "Enter second integer: " int2
        res=$(($int1*$int2))
;;
[dD])
        read -p "Enter first integer: " int1
        read -p "Enter second integer: " int2
        res=$(($int1/$int2))
;;
```

While & case

```
*)
        res=0
        echo "wrong choice!"
esac

echo "The result is: " $res
read -p "Do you wish to continue? [y]es or [n]o: " ans
if [ $ans == 'n' ]
        then
          echo "Exiting the script. Have a nice day!"
        break
else
        continue
fi

done
```

While & case: output

Choose from the following operations:

[a]ddition

[b]Subtraction

[c]Multiplication

[d]Division

##

Your choice:

While & case 2 function

```
#!/bin/bash
inputs(){
        read -p "Enter first integer: " int1
        read -p "Enter second integer: " int2
}

exitPrompt(){
      read -p "Do you wish to continue? [y]es or [n]o: " ans
      if [ $ans == 'n' ]
      then
            echo "Exiting the script. Have a nice day!"
            break
      else
            continue
      fi
}
```

While & case 2 function

```
while(true)
    do
    clear
    printf "Choose from the following operations: \n"
    printf "[a]Addition\n[b]Subtraction\n[c]Multiplication\n[d]Division\n"
    printf "#####################################\n"
    read -p "Your choice: " choice

    case $choice in
    [aA])
        inputs
        res=$(($int1+$int2))
    ;;
```

While & case 2 function

```
[bB])
        inputs
        res=$(($int1-$int2))
;;
[cC])
        inputs
        res=$(($int1*$int2))
;;
[dD])
        inputs
        res=$(($int1/$int2))
;;
```

While & case 2 function

```
        *)
                res=0
                echo "wrong choice!"
        esac

        echo "The result is: " $res
        exitPrompt
done
```

Passing parameters on functions

```
#!/bin/bash

myfunction(){
        echo  $1
        echo  $2
}

myfunction  "Hello"  "World"
```

Passing parameters on functions

```
#!/bin/bash
myfunction(){
        echo $1
        echo $2
}

myfunction "Hello" "World"
echo $1
echo $2
```

$ bash program.sh olarik surinta

Returning Values from Functions

- shell scripts cannot return multiple values from a function. Let's take a look in this example:

```
#!/bin/bash

add(){
      sum=$(($1+$2))
      return $sum
}

read -p "Enter an integer: " int1
read -p "Enter an integer: " int2
add $int1 $int2
echo "The result is: " $?
```

Function: sample

```
#!/bin/bash

clear(){
        clear
}

bin(){
        bin1=$(echo "obase=2;$1"|bc)
        echo $bin1
}

dec(){
        dec1=$(echo "ibase=2;$1"|bc)
        return $dec1
}
```

```
########Main#########
        printf "Choose from the following
                operations:\n[1]Decimal to Binary Conversion\n"
        printf "[2]Binary to Decimal Conversion\n"
        read -p "Your choice: " op
        case $op in

        1)
                read -p "Enter integer number: " int
                bin $int
        ;;

        2)
                read -p "Enter binary number: " int
                dec $int
                echo "The decimal equivalent of $int is $?"
        ;;

        *)
                echo "Wrong Choice!"
        esac
```

Function: sample

Choose from the following operations:

[1]Decimal to Binary Conversion

[2]Binary to Decimal Conversion

Your choice:

Bash command: trick

Q: What bash command you can use to show detail of a selected FILE?

A: cat (concatenate)

Q: And what about "tail" command?

A: "tail" command is "print the last 10 lines of each FILE"

Bash command: trick

Q: So, If I would like to show detail of "factorial.sh" file, What can I do?

A: From the terminal program, just type:

$ cat factorial.sh

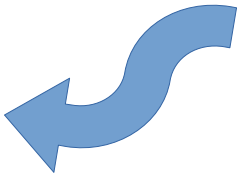
Q: I would like to write a bash program to show detail of selected FILE, but don't want to use the command **'CAT'**, How can I do it?

While loop: sample

```
#!/bin/bash
value=$1

while read line
do
        echo $line
done < $value
```

$ bash read_file.sh xxx.xxx



I/O redirection: input

- **Input redirection**
  - Instead of controlling a loop by testing the result of a command or by user input, you can specify a file from which to read input that controls the loop.
  - The redirection should occur after the **done** statement

command < file

While loop: sample

- While loops are frequently used for reading data line by line from file.

$ sudo nano read_file.sh

```
#!/bin/bash

FILE=$1
# read $FILE using the file descriptors
exec 3<&0
exec 0<$FILE

while read line
do
    # use $line variable to process line
    echo $line
done
exec 0<&3
```

$ bash read_file.sh while-loop.sh

While loop: use wile as infinite loops

```
#!/bin/bash


while :
do
    echo "infinite loops [hit CTRL+C to stop]"
done
```

While Loop: Conditional while loop exit with break statement

- You can do early exit with the break statement inside the while loop.
- You can exit from within a WHILE using **break**

While Loop: Conditional while loop exit with break statement

```
#!/bin/bash

while :
do
        read -p "Enter two numnbers ( - 1 to quit ) : " a b
        if [ $a -eq -1 ]
        then
                break
        fi
        ans=$(( a + b ))
        echo $ans
done
```

$ read -p

-p <prompt>

While Loop: Early continuation with the continue statement

- To resume the next iteration of the enclosing WHILE loop use t he continue statement as follows:

```
while [ condition ]
do
   statements1          #Executed as long as condition is true and/or, up to a disaster-condition if any.
   statements2
   if (condition)
   then
            continue     #Go to next iteration of I in the loop and skip statements3
   fi
   statements3
done
```

While Loop: Nested while loops

- A **nested loop** is a loop within a loop, an inner loop within the body of an outer one.
- How this works is that the first pass of the outer loop triggers the inner loop, which executes to completion. Then the second pass of the outer loop triggers the inner loop again.
- This repeats until the outer loop finishes.
- A **break** with in either the inner or outer loop would interrupt this process.

Nested loop: for

```
#!/bin/bash

# nested-loop.sh: Nested "for" loops.

outer=1 # Set outer loop counter.

# Beginning of outer loop.
for a in 1 2 3 4 5
do
  echo "Pass $outer in outer loop."
  echo "---------------------"
  Inner=1 # Reset inner loop counter.

  # ===================================
  # Beginning of inner loop.
  for b in 1 2 3 4 5
  do
    echo "Pass $inner in inner loop."
    let "inner+=1"  # Increment inner loop counter.
  done
  # End of inner loop.
  # ===================================

  let "outer+=1"     # Increment outer loop counter.
  echo       # Space between output blocks in pass of outer loop.
done
# End of outer loop.

exit 0
```

output

```
Pass 1 in outer loop.
----------
Pass 1 in inner loop.
Pass 2 in inner loop.
Pass 3 in inner loop.
Pass 4 in inner loop.
Pass 5 in inner loop.

Pass 2 in outer loop.
----------
Pass 1 in inner loop.
Pass 2 in inner loop.
Pass 3 in inner loop.
Pass 4 in inner loop.
Pass 5 in inner loop.

Pass 3 in outer loop.
----------
Pass 1 in inner loop.
Pass 2 in inner loop.
Pass 3 in inner loop.
Pass 4 in inner loop.
Pass 5 in inner loop.

Pass 4 in outer loop.
----------
Pass 1 in inner loop.
Pass 2 in inner loop.
Pass 3 in inner loop.
Pass 4 in inner loop.
Pass 5 in inner loop.

Pass 5 in outer loop.
----------
Pass 1 in inner loop.
Pass 2 in inner loop.
Pass 3 in inner loop.
Pass 4 in inner loop.
Pass 5 in inner loop.
```

Nested loop

- Can we change from "nested for loop" to "nested while loop"?

Let's do it!

Nested loop: while loop

```
outer=1

while [ $outer -lt 6 ]
do
    echo "Pass $outer in outer loop."
    echo "---------------------"
    inner=1

    while [ $inner -lt 6 ]
    do
        echo "Pass $inner in inner loop."
        let "inner+=1"
    done

    let "outer+=1"
    echo
done
```

inner=$(($inner + 1))

outer=$(($outer + 1))

While Loop: calculating an average

```
#!/bin/bash

SCORE="0"
AVERAGE="0"
SUM="0"
NUM="0"

while true; do

  echo -n "Enter your score [0-100%] ('q' for quit): "; read SCORE;

  if (("$SCORE" < "0"))  || (("$SCORE" > "100")); then
    echo "Be serious.  Common, try again: "
  elif [ "$SCORE" == "q" ]; then
    echo "Average rating: $AVERAGE%."
    break
  else
    SUM=$[$SUM + $SCORE]
    NUM=$[$NUM + 1]
    AVERAGE=$[$SUM / $NUM]
  fi

done

echo "Exiting."
```

output

```
Enter your score [0-100%] ('q' for quit): 100
Enter your score [0-100%] ('q' for quit): 95
Enter your score [0-100%] ('q' for quit): 95
Enter your score [0-100%] ('q' for quit): 97
Enter your score [0-100%] ('q' for quit): 100
Enter your score [0-100%] ('q' for quit): q
Average rating: 97%.
Exiting.
```

Piping and Redirection

- By piping, you can send the result of a command to another command.
- By redirection, you can determine where the command should send its results.

Piping

- By using piping, you can combine the abilities of two or more commands to create a kind of super command that offers even more capabilities.

```
$ kill $(ps aux | grep y2 | gerp -v grep | awk '{ print $2 }')
```

What is the output of this command?

Piping

- By using piping, you can combine the abilities of two or more commands to create a kind of super command that offers even more capabilities.

$ kill 'ps aux | grep y2 | gerp -v grep | awk '{ print $2 }"

What is the output of this command?

Piping

$ ps aux

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	4600	2572	?	Ss	10:07	0:01	/sbin/init
root	2	0.0	0.0	0	0	?	S	10:07	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	S	10:07	0:00	[ksoftirqd/0]
root	4	0.0	0.0	0	0	?	S	10:07	0:00	[kworker/0:0]
root	5	0.0	0.0	0	0	?	S<	10:07	0:00	[kworker/0:0H]

Piping

$ ps aux | grep y2

```
root        1298  0.0  0.0     4644     824  tty2        Ss+
      10:07     0:00  /sbin/getty  -8  38400  tty2
pro         4671  0.0  0.0     4692     836  pts/0       S+
      12:25     0:00  grep  --color=auto  y2
```

Piping

```
$ ps aux | grep y2 | grep -v grep
root        1298  0.0  0.0   4644   824
      tty2        Ss+  10:07   0:00
      /sbin/getty -8 38400 tty2

$ ps aux | grep y2 | grep -v grep | awk
'{ print $2 }'
1298
```

Piping

```
$ kill $(ps aux | grep y2 | gerp -v grep | awk '{ print $2 }')
```

I/O Redirection: sample

```
#!/bin/bash
# Reading lines in /etc/fstab.

File=/etc/fstab

{
read line1
read line2
} < $File

echo "First line in $File is:"
echo "$line1"
echo
echo "Second line in $File is:"
echo "$line2"

exit 0
```

I/O Redirection

- There are always three default files open,
  - **stdin** (the keyboard)
  - **stdout** (the screen)
  - **stderr** (error message output to the screen.

std = standard

I/O Redirection

- Basically you can:
  - Redirect **stdout** to a file
  - Redirect **stderr** to a file
  - Redirect **stdout** to a **stderr**
  - Redirect **stderr** to a **stdout**
  - Redirect **stderr** and **stdout** to a file
  - Redirect **stderr** and **stdout** to **stdout**
  - Redirect **stderr** and **stdout** to **stderr**

1 'represents' stdout and 2 stderr

I/O Redirection

- Basically you can:
  - Redirect **stdout** to a file
  - Redirect **stderr** to a file
  - Redirect **stdout** to a **stderr**
  - Redirect **stderr** to a **stdout**
  - Redirect **stderr** and **stdout** to a file
  - Redirect **stderr** and **stdout** to **stdout**
  - Redirect **stderr** and **stdout** to **stderr**

I/O Redirection: stdout 2 file

- This will cause the output of a program to be written to a file.

```
$ ls -l > ls-l.txt
```

I/O Redirection: stderr 2 file

```
$ find / -name core > find-core.txt
```

```
$ find / -name core &> find-core1.txt
```

I/O Redirection: stderr 2 file

- This will cause the **stderr** output of a program to be written to a file.

$ ls tmp * 2> ls-errors1.txt

What is the different between these two command?

$ ls tmp * > ls-errors2.txt

I/O Redirection: stderr 2 file

```
$ ls tmp * >> ls-errors3.txt
```

I/O Redirection: stderr 2 file

$ ls -l message.txt abc > errors.txt

ls: cannot access 'abc': No such file or directory

-rw-rw-r-- 1 mrolarik mrolarik 40587471 Jul 18 2016 message.txt

$ cat errors.txt

-rw-rw-r-- 1 mrolarik mrolarik 40587471 Jul 18 2016 message.txt

I/O Redirection: stderr 2 file

$ ls -l message.txt abc 2> errors.txt

-rw-rw-r-- 1 mrolarik mrolarik 40587471 Jul 18 2016 message.txt

$ cat errors.txt

ls: cannot access 'abc': No such file or directory

I/O Redirection: stderr 2 file

$ ls -l message.txt abc > errors.txt 2>&1

$ cat errors.txt

ls: cannot access 'abc': No such file or directory

-rw-rw-r-- 1 mrolarik mrolarik 40587471 Jul 18 2016 message.txt

Piping and Redirection: Question

- Q1: How can I print text as show below to screen (stdout)?

```
$ ********
Ibrahimovic
Messi
Ronaldo
Neymar
```

Piping and Redirection: Question

```
$ echo -e "Ibrahimovic\nMessi\nRonaldo"
```

Ibrahimovic

Messi

Ronaldo

Piping and Redirection: Question

- Q2: Can I insert text before each name

```
$ *******
Hi Ibrahimovic
Hi Messi
Hi Ronaldo
```

Piping and Redirection: Question

```
$ echo -e "Ibrahimovic\nMessi\nRonaldo"
| while read n; do echo "Hi $n"; done
```

Hi Ibrahimovic

Hi Messi

Hi Ronaldo

Do you have another way?

Piping and Redirection: Question

```
$ echo -e "Ibrahimovic\nMessi\nRonaldo"
| awk '{print "Hi "$1" "}'
```

Hi Ibrahimovic

Hi Messi

Hi Ronaldo

Piping and Redirection

- Passing input by stdin:

```
$ ls | wc -l
```

- Passing input by command line arguments

```
$ wc -l $(ls)
```

Did you see the different?

Piping

```
$ ls -l | head -5 | tail -2
```

??? Result ???

Piping: more example

```
$ ls -l /etc/ | tail -5
drwxr-xr-x   9 root root      4096 May 24   2016 xdg
drwxr-xr-x   2 root root      4096 May 24   2016 xfce4
drwxr-xr-x   2 root root      4096 Jan 28 20:35 xml
drwxr-xr-x   5 root root      4096 Jun 20   2016 yum
-rw-r--r--   1 root root       477 Jul 20   2015
zsh_command_not_found
```

Piping: more example

$ ls -l /etc/ | tail -5 | sort

drwxr-xr-x 2 root root 4096 Jan 28 20:35 xml

drwxr-xr-x 2 root root 4096 May 24 2016 xfce4

drwxr-xr-x 5 root root 4096 Jun 20 2016 yum

drwxr-xr-x 9 root root 4096 May 24 2016 xdg

-rw-r--r-- 1 root root 477 Jul 20 2015 zsh_command_not_found

Stuff we learnt

> Save output to a file.

>> Append output to a file.

< Read input from a file.

2> Redirect error messages.

| Send the output from one program as input to another program.

- http://ryanstutorials.net/linuxtutorial/piping.php
- http://stackoverflow.com/questions/11917708/pipe-multiple-commands-to-a-single-command
- ก
-

References:

- http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-3.html
- http://tldp.org/LDP/abs/html/io-redirection.html
- http://tldp.org/LDP/abs/html/x17974.html#REDIR1

References:

- https://www.cyberciti.biz/faq/bash-while-loop/
- https://linux.die.net/Bash-Beginners-Guide/sect_09_02.html
- http://tldp.org/LDP/abs/html/special-chars.html
- http://tldp.org/LDP/abs/html/io-redirection.html
- http://tldp.org/LDP/abs/html/special-chars.html#EX8
- https://www.howtoforge.com/tutorial/linux-shell-scripting-lessons-5/
-
- d

References:

- http://unix.stackexchange.com/questions/113515/how-to-modify-output-in-bash-command-pipeline
- http://unix.stackexchange.com/questions/108782/pass-the-output-of-previous-command-to-next-as-an-argument
-
- s