สารบัญ

BASIC COMMAND

คำสั่งทั่วไป

man คำสั่ง ดูว่าคำสั่งนั้นๆ ใช้อย่างไรและมีรูปแบบคำสั่งอย่างไร หรือ เป็น manual ของคำสั่งนั้นๆนั่นเอง

more ไฟล์ เป็นการดูข้อมูลใน ไฟล์นั้นๆ กด Enter เพื่อดูบรรทัดต่อไป กด Space เพื่อดูหน้าต่อไป

คำสั่ง1 | คำสั่ง 2 output ของคำสั่งที่ 1 จะเป็น Input ของคำสั่งที่ 2 เช่น **ls | more**

cat ไฟล์ เป็นการดูรายละเอียดของ file ทีเดียวทั้งหมดที่มีของ file นั้นๆ

tac ไฟล์ อ่านไฟล์จากล่างขึ้นบน

id เป็นคำสั่งที่แสดงชื่อและกลุ่มของผู้ใช้งานในขณะนั้น

logname เป็นคำสั่งแสดงชื่อผู้ใช้งานในขณะนั้น

tty เป็นการดูหมายเลขของ terminal ที่ผู้ใช้ใช้อยู่ ผลจะออกมาเป็น /dev/pts/หมายเลข

hostname เป็นการดูชื่อเครื่องที่เรากำลังใช้งานอยู่

uname –a แสดงรายละเอียดเวอร์ชั่นของ sunos

set –o vi เป็นการเซ็ทเป็นโหมดของ vi การใช้งาน ต้องกด Esc ก่อนแล้ว กด j , k เพื่อเรียกคำสั่งเดิมมาใช้โดยสามารถแก้ไขได้แบบ vi **ใช้ได้เฉพาะบน ksh,bash เท่านั้น**

echo ข้อความ เป็นการแสดงข้อความที่ต้องการ

who –u ใช้ดูว่าในขณะนั้นมีใคร login อยู่ในระบบ (เวลาเริ่มต้น เวลาสิ้นสุดบ้าง)

who ดูว่าใคร login อยู่บ้าง จะเห็น IP และ terminal (/dev/pts) ของผู้ login

alias คำสั่งย่อ= คำสั่งเต็ม เป็นการย่อคำสั่งให้สั้นลง

type คำสั่ง ดูว่าคำสั่งนี้มีอยู่หรือไม่ มีอยู่ที่ path ไหน

which คำสั่ง ดูว่าคำสั่งนี้มีอยู่หรือไม่ มีอยู่ที่ path ไหน

stty erase key_set เป็นการเซ็ตปุ่มในการ ลบ (เหมือน BackSpace)

grep ข้อความ /path เป็นการค้นหาข้อความที่ต้องการ หรือ pkg หรือ deamon ที่ต้องการ

ls |grep สิ่งที่ต้องการหา เป็นการหาสิ่งที่ต้องการจากผลของคำสั่ง ls

ll ดู list ไฟล์แบบเต็มโดยจะแสดง properties ต่างๆด้วย

cd /path : เปลี่ยน path

cp file1 file2 : copy file1 to file 2

rm file : ลบไฟล์

rm –r dir : ลบ dir

mv file1 file2 : เปลี่ยนชื่อ file1 เป็น file2

touch filename สร้าง file เปล่า

expr numA + numB บวกค่า numA และ numB

A=$(ls) หรือ A=`ls` เป็นการเก็บผลลัพธ์ของคำสั่ง ls ไปเก็บไว้ที่ตัวแปร A

VI Editor

VI (Virtual Interface) เป็น Editor ที่นิยมใช้มากที่สุดในระบบ Unix และบางท่านอาจคิดว่าเป็น Editor ที่ใช้ยากที่สุด แต่หากได้ลองใช้อย่างต่อเนื่องจนเกิดความเคยชิน แล้ว จะเห็นได้ว่า Vi Editor นั้นเป็น Editor ที่ใช้งานสะดวกและที Short key ให้ใช้อย่างสะดวก เป็นเป็น Editor ที่น่าหลงไหลอย่างนึงเลยทีเดียว

ทั้งนี้ผมจะขอนำเสนอ Short key ที่ได้งานบ่อยๆ ในการเขียน Shell Script ดังต่อไปนี้ครับ
ก่อนอื่น vi เข้าไปยัง File ที่ต้องการแก้ไขด้วยคำสั่ง

```
# vi  filename
```

จากนั้นทดลองใช้งาน Short key ต่างๆ

** Vi Editor นั้น ใช้ปุ่ม Esc ในการเปลี่ยน mode การทำงานโดยโหมดการทำงานสามารถแยกได้ดังนี้

Insert Mode : เหล่านี้ได้แก่

i	คือ การบอกกับ Editor ว่าจะเริ่มทำการ insert ข้อมูล จากนั้นทำการ insert ข้อมูลลงไป หรือหาก i อยู่ระหว่าง word ก็จะเป็นการ แทรกอักขระลงไปก่อนตำแหน่ง current cursor
a	คือการ append ข้อมูลหลัง current cursor
o	ขึ้นบรรทัดใหม่ต่อจาก บรรทัดปัจจุบัน
O	ขึ้นบรรทัดใหม่ เหนือบรรทัดปัจจุบัน
J	นำบรรทัดล่างมาต่อบรรทัดปัจจุบัน

Delete Mode : ได้แก่

x	เป็นการ delete ทีละอักขระ จาก current cursor ถอยหลังไปเรื่อยๆ
dw	เป็นการ delete word คือ delete ทีละ 1 ข้อความ
dd	เป็นการ delete ทีละบรรทัด
d$	Delete จาก current cursor จนถึงท้ายบรรทัด
d^	Delete จาก current cursor จนถึงต้นบรรทัด
dG	Delete จาก current line จนถึงท้ายไฟล์
d1G	Delete จาก current line จนถึงต้นไฟล์

Move Mode : ได้แก่

j	เลื่อน cursor จากแถวบนลงแถวล่าง
k	เลื่อน cursor จากแถวล่างขึ้นแถวบน
h	เลื่อน cursor ไปด้านซ้ายทีละอักขระ
l	เลื่อน cursor ไปด้านขวาทีละอักขระ
$	เลื่อน cursor ไปยังท้ายบรรทัด
^ , 0	เลื่อน cursor ไปยังต้นบรรทัด
w	เลื่อน cursor ไปทีละ word
b	เลื่อน cursor กลับทีละ word
1G	เลื่อน cursor ไปยังต้นไฟล์
G	เลือน cursor ไปยังท้ายไฟล์
15G	เลื่อน cursor ไปยังบรรทัดที่ 15

Copy Mode : ได้แก่

yy	กำหนดบรรทัดที่จะทำการ copy (1 บรรทัด current cursor)
5y	กำหนดบรรทัดที่จะทำการ copy 6 บรรทัดรวมบรรทัด current cursor ด้วย
p	วางบรรทัดที่ได้กำหนดต่อจากบรรทัด current cursor

การค้นหา word โดยกดเครื่องหมาย / ตามด้วย word ที่ต้องการค้นหา

เป็นการค้นหา word จากตำแหน่ง cursor ไปจนถึงท้ายไฟล์หาก word ที่ค้นหามีอยู่หลายตำแหน่ง สามารถ กดปุ่ม n เพื่อค้นหา word ในตำแหน่งถัดไป

การค้นหา word โดยกดเครื่องหมาย ? ตามด้วย word ที่ต้องการค้นหา

เป็นการค้นหา word จากตำแหน่ง cursor ไปจนถึงต้นไฟล์หาก word ที่ค้นหามีอยู่หลายตำแหน่ง สามารถ กดปุ่ม n เพื่อค้นหา word ในตำแหน่งถัดไป

นอกจากนี้ Vi Editor ยังสามารถใช้ร่วม Stream Editor (sed) โดยการ กดปุ่ม : (colon) เพื่อ insert คำสั่งเข้า

การ save file

:w	save file แบบไม่ออกจาก vi
:wq!	save file และออกจาก vi

Header of script

การเขียน Script ที่ดีนั้นควรจะต้องการ รายละเอียดพอสังเขป ของ Script เช่น Script name , Description,Author,Modify date เป็นต้น

#!/bin/bash : บรรทัดนี้เป็นตัวระบุว่า Script นี้จะถูก compile ด้วย Shell ชนิดใด

```
#-----------------------------------------------------------------------------------------
# Script name :  TransferFile.bash
# Description :   Transfer file from /data  -> /backup/data/
# Modify date :   10/10/2007
# Author         :  Siwanat  Ponkhun , Programmer Analyst
# Contact        :   Mobile 087-1028290
#-----------------------------------------------------------------------------------------
```

Variable

การตั้งค่าตัวแปรก็คล้ายกับ programming ทั่วไปโดยต้องไม่ขึ้นต้นด้วยอักขระพิเศษ หรือตัวเลข โดยต้องขึ้นต้นด้วยตัวอักษรแล้ว จากนั้นจึงสามารถตามด้วยตัวเลขหรือตัวอักษรได้ และขอแนะนำเพิ่มเติมว่าควรตั้งชื่อตัวแปรให้สื่อ ต่อการใช้งาน และควรให้เป็นที่สังเกตง่าย โดยผมเอง มักตั้งชื่อตัวแปรเป็นตัวพิมพ์ใหญ่ เพื่อให้ง่ายต่อการสังเกตและใช้งานครับ เช่น

```
SCRIPT_PATH=/home/siwanat
DATA_PATH=${SCRIPT_PATH}/data
TEMP_PATH=${SCRIPT_PATH}/temp
TEMP_FILE=${TEMP_PATH/file.tmp
OUTPUT_FILE=${DATA_PATH}/file.output
TOTAL=100
```

สำหรับการนำตัวแปรไปใช้งานนั้น สามารถทำได้โดยการใส่ ตัว $ ไว้หน้าชื่อตัวแปรส่วนเครื่องหมาย {} ที่ครอบอยู่ที่ตัวแปรที่ผมได้ใส่ข้างต้นนั้น อาจจะใส่หรือไม่ใส่ก็ได้ แต่ผมแนะนำให้ใส่ ให้เป็นนิสัยเลยครับ เพราะการใส่เครื่องหมาย {} นี้เป็นการบอกถึงขอบเขตการสิ้นสุดตัวแปร เพราะเนื่องจาก การเขียน Script บางครั้ง มีการนำค่าของตัวแปรมาต่อกับอักขระ ซึ่งหากไม่มีเครื่องหมาย {} มาระบุขอบเขตชื่อตัวแปรแล้ว จะทำให้ Shell ไม่รู้จักชื่อตัวแปรนั้นๆ ตัวอย่างเช่น

```
FILE_NAME=Data.csv
OUTPUT_FILE=$FILE_NAME_output
```

จากการตั้งชื่อตัวแปรดังกล่าว เมื่อเราอ้างตัวแปร $OUTPUT_FILE ไปใช้งาน ตัวแปรจะไม่มีค่าเนื่องจาก Shell ไม่รู้จักตัวแปร $FILE_NAME_output ที่จะส่งให้ตัวแปร OUTPUT_FILE แต่เมื่อเราทำการใส่เครื่องหมาย {} ให้กับตัวแปร FILE_NAME ดังนี้

```
OUTPUT_FILE=${FILE_NAME}_out
```

ตัวแปร OUTPUT_FILE จะมีค่าเท่ากับ Data.csv_output ซึ่งเป็นค่าที่ Transfer มาจากตัวแปร FILE_NAME บวกกับ อักขระ _output นั่นเอง

Operator

- Numberic Operator

  การเปรียบเทียบค่าของตัวเลข สามารถใช้เครื่องหมายดังต่อไปนี้

  -eq : equal หมายถึงหากค่าของตัวแปรมีค่า เท่ากัน

  -ne : not equal หมายถึงหากค่าของตัวแปรมีค่า ไม่เท่ากัน

  -lt : less than หมายถึงหากค่าของตัวแปรมีค่า น้อยกว่า

  -le : less than or equal หมายถึงหากค่าของตัวแปรมีค่า น้อยกว่า หรือ เท่ากัน

  -gt : grater than หมายถึงหากค่าของตัวแปรมีค่า มากกว่า

  -ge : grater than or equal หมายถึงหากค่าของตัวแปรมีค่า มากว่า หรือ เท่ากัน

  +,-,*, / , % : บวก , ลบ , คูณ , หารแบบไม่เอาเศษ , หารแบบเอาเฉพาะเศษ(mod)

- Character Operator

  = หรือ = = หมายถึง หากตัวแปรทั้งสองมีค่า เหมือนกัน

  != หมายถึง หากตัวแปรทั้งสองมีค่า ไม่เหมือนกัน

- Logic Operator

  -z zero หมายถึงถ้าหากตัวแปรมีค่า ว่าง

  -n non zero หมายถึงถ้าหากตัวแปร มีค่าอยู่จริง (ไม่เป็นค่าว่าง)

- File and Directory Operator

  -ot older than หมายถึงหากตัวแปร file ด้านซ้าย เก่ากว่า file ด้านขวา

  -nt newer than หมายถึงหากตัวแปร file ด้านซ้าย ใหม่กว่า file ด้านขวา

  -f file หมายถึงหากตัวแปรเป็น file และมีอยู่จริง

  -d directory หมายถึงหากตัวแปรเป็น directory และมีอยู่จริง

  -r read หมายถึงหากตัวแปร file สามารถ read ได้จริง

  -w write หมายถึงหากตัวแปร file สามารถ write ได้จริง

  -x execute หมายถึงหากตัวแปร file สามารถ execute ได้จริง

  -s size หมายถึงหากตัวแปร file มีขนาดมากกว่า 0 byte จริง

- Meta character

  คืออักขระ ที่มีความหมายพิเศษในตัวเอง ที่ใช้บ่อยมีดังนี้

  * แทนอักขระ อะไรก็ได้มากว่า 1 อักขระ เช่น *.log หมายถึงชื่ออะไรก็ได้ที่ลงท้ายด้วย .log

  ? แทนอักขระอะไรก็ได้ 1 อักขระ เช่น Da??.log หมายถึงขึ้นต้นด้วย Da ตามด้วยอักขระอะไรก็ได้อีก 2 อักขระแล้วลงท้ายด้วย .log

  “ “ double qout เป็นตัวบอกขอบเขตของกลุ่มอักขระแต่ยังคงแสดงค่าในตัวแปรออกมาได้

  ‘ ‘ single qout เป็นตัวบอกขอบเขตของกลุ่มอักขระแต่จะไม่แสดงค่าในตัวแปรออกมา

  ^ แทนการขึ้นต้น เช่น ^Shell คือขึ้นต้นด้วยคำว่า Shell , $ แทนการลงท้ายเช่น Shell$

- Comment

  การ Comment คือการละเว้นไม่ให้ Shell compile คำสั่งในบรรทัดนั้นๆ ทำได้โดยการใส่เครื่องหมาย **“ # ”** ที่หน้าบรรทัดนั้น Shell จะข้ามบรรทัดนั้นไปโดยไม่มีการ Compile

Condition

การตรวจสอบเงื่อนไข ใน Shell Script นั้นมีความสำคัญอย่างมากเนื่องจากเป็น ตัวกำหนดการกระทำถัดไปของ Program สำหรับ Syntax ที่มักใช้คู่กับเงื่อนไขอยู่เสมอนั่นก็คือ

- if , else
- case
- While , Do while

IF , ELSE

เป็น Statement ที่พบบ่อยใน Programming ซึ่งใน Shell Script เองก็เช่นเดียวกันครับ มี Concept เหมือนกับ Programming ทั่วไป แต่อาจจะต่างกันที่ Syntax เล็กน้อยดังนี้

```
if [  condition1 ]
  then
       statment1
       ……
elif [ confition2 ]
  then
       statment2
       ……
  else
       statment3
       ……
fi
```

ตัวอย่าง

```
INPUT_NUM=100
REF_NUM=10

if [ ${INPUT_NUM} –gt ${REF_NUM} ]
   then
     echo “Input number grater than reference number.”
elif [ ${INPUT_NUM} -lt ${REF_NUM} ]
   then
     echo “Input number less than reference number.”
else
     echo “Input number and reference number is equal.”
fi
```

```
PW_FILE=/etc/passwd

if [ -f  “${PW_FILE}” ]
   then
      echo “${PW_FILE} is a file.”
   else
      echo “${PW_FILE} is not file.”
fi
```

Multi condition

หากเงื่อนไขมีมากกว่า 1 ชุดเราก็สามารถเชื่อม เงื่อนไขระหว่างแต่ละชุดได้โดย

-a and หมายถึง หากเงื่อนไขทั้งสองชุดเป็นจริงทั้งคู่

-o or หมายถึง หากเงื่อนไขใด เงื่อนไข หนึ่งในสองชุด เป็นจริง

ตัวอย่าง

```
NAME=Shell
LAST_NAME=Script
SEX=male

if [ "${NAME}" = "Shell" –a  "${SEX}" = "male" ]
   then
      echo "You success authorized."
   else
      echo "You failed."
fi


NAME=Shell
LAST_NAME=Script
SEX=female
NULL_VAL=""

if [ "${NAME}" = "Shell" –o  "${SEX}" = "female" ]
   then
      echo "You success authorized."
   else
      echo "You failed."
fi


if [ -z  "${NULL_VAL}" ]
    then
      echo "Value is null"
    else
      echo "Value is not null"
fi
```

สำหรับการเปรียบเทียบค่าที่เป็นอักขระนั้นให้ ให้ใส่เครื่องหมาย " "คร่อมตัวแปรและตัวเปรียบเทียบเสมอครับ เนื่องจากบางครั้งตัวแปร เก็บค่าที่มี space คั่น เช่น NAME="Shell Script" หากเราไม่ใส่เครื่องหมาย " " แล้ว Shell จะมองเห็นค่าตัวแปรเป็น 2 คำซึ่งจะทำให้ผิด syntax ของการใช้ if ครับ ดังนั้นให้ใส่เครื่องหมาย " " คร่อมตัวแปรที่เป็นอักขระเสมอเมื่อมีการใช้เปรียบเทียบเงื่อนไขครับ

Case

สำหรับท่านที่เคย ทำงาน Programming มาแล้วคงจะคุ้นเคยกับ switch case นะครับและแน่นอนครับว่า Concept นั้นเหมือนกันคือจะรับตัวแปรมา 1 ตัวแปรจากนั้น case จะตรวจสอบว่าค่าตัวแปรที่ได้ตรงกับเงื่อนไขใด แล้วจึงเข้าไปทำงานยัง statement ในเงื่อนไขนั้นๆ โดย มี syntax ดังนี้

```
case  ${VARIABLE} in
       CONDITON_A)       statement A
                         ……
                         ;;
      CONDITON_B)        statement B
                         ……
                         ;;
       CONDITON_C)       statement C
                         ……
                         ;;
        *)               other statement
                         ;;
esac
```

ทั้งนี้ case ยังสามารถที่จะใส่ Condition ได้มากกว่า 1 Condition และสามรถใช้ร่วมกับ Meta character ได้อีกด้วย ตัวอย่างเช่น

```
OS_NAME=redhat

case ${OS_NAME} in
       *hat|fedora)    echo “Your operation system is redhat or fedora”   ;;
          Solaris)     echo “Your operation system is Solaris”   ;;
        “Shell os”)    echo “Your operation is Shell os”  ;;
                *)     echo “Your operation system is not not  redhat  fedora or
                       solaris”   ;;
esac
```

Loop

เป็นอีกหนึ่ง Syntax ที่ต้องรู้ครับเพราะใช้บ่อยครั้งมาก Concept และ Syntax นั้นไม่ยากเย็นครับ ลองมาดูกันเลยครับ

FOR LOOP

```
for  i  in ${ARRAY}
  do
     statement
     ……
done
```

สำหรับตัวแปร ${ARRAY) จะเป็นข้อมูลที่เป็น Array จริงๆ หรือ เป็นข้อมูลจากตัวแปรที่คั่นด้วย space เช่น ARRAY=”1 2 3 4 5” ก็ได้เช่นกันครับ

สำหรับท่านที่เคยใช้ for loop ที่มีการ initial ค่า เพิ่มค่า และ ใส่เงื่อนไข ได้ในคราวเดียวนั้นแนะนำให้ใช้ คำสั่ง seq เข้ามาช่วยครับ เนื่องจาก shell script ไม่ได้มี syntax รองรับการ initial ดังกล่าว วิธีการดังตัวอย่างต่อไปนี้ครับ

```
for i in $(seq 0 1 10)
   do
      statement
      ……
done
```

หรือ

```
for (( i=1 ; i<=10; i++))
  do
    echo ${i}
done
```

ความหมายของ script ด้านบนนี้คือ ให้ i มีค่าเริ่มต้นที่ 1 เพิ่มค่าไปทีล่ะ 1 จนกว่า i มีค่าเท่ากับ 0 อย่างไรก็ตาม script ข้างต้นอ้างอิงกับ bash shell บน Redhat Linux ครับ

```
for (( i=1 ; i<=10; i=i+2))
  do
    echo ${i}
done
```

break เป็นคำสั่งที่ใช้ใน loop ใช้ได้ทั้ง loop for และ while loop เมื่อต้องการหยุดการทำงานของ loop นั้นๆ ครับเมื่อเจอคำสั่ง break script จะหลุดออกจาก loop ทันทีครับ

continue เป็นคำสั่งที่ใช้ใน loop เช่นกัน ใช้เมื่อต้องการให้ loop กลับขึ้นไปทำการวนรอบครั้งต่อไปโดยไม่สนใจคำสั่งที่อยู่หลังคำสั่ง continue ครับ

While loop

เป็น loop ที่จะทำจนกว่าเงื่อนไขจะเป็นเท็จจากนั้นจึงจะหลุดออกจาก loop หรือเจอคำสั่ง break นอกจากนี้ while loop ยังมีความสามารถที่จะอ่าน file เพื่อนำข้อมูลมาวนรอบได้ รวมทั้งยังยังสามารถที่จะทำเป็น loop อนันท์ เพื่อทำงานต่อเนื่องไปเรื่อยๆ ได้ด้วย ตัวอย่างของ while loop มีดังต่อไปนี้

```
ANUM=10
BNUM=1
while [ ${BNUM} –lt ${ANUM} ]
      do
        echo “${BNUM} < ${ANUM}”
        ((BNUM+=1))
done
```

จาก script while loop จะทำงานต่อเมื่อ BNUM มีค่าน้อยกว่า ANUM เมื่อ BNUM เพิ่มขึ้นมีค่าเท่ากับ ANUM loop ก็จะหยุดการทำงานทันทีโดยไม่เข้าไปทำงานใน loop อีก

```
 while  true
     do
        PSID=$(ps –ef  |grep stop.bash)
         if [ -n “${PSID}” ]
           then
              exit
         fi
 done
```

อธิบายได้ดังนี้คือ loop จะทำงานไปจนกว่า จะมี process ที่ชื่อว่า stop.bash เกิดขึ้นมาเมื่อมี process stop.bash เกิดขึ้นจะเข้าเงื่อนไข และทำคำสั่งในเงื่อนไข if นั่นคือ exit ออกจาก program นั่นเอง

```
 INPUT_FILE=Shell.log
 while read line
    do
       echo “Test : ${line}”
done < ${INPUT_FILE}
```

เป็นการ ใช้ while loop ในการ อ่าน file โดยการใช้ การ redirect input (<) เข้ามาช่วย script นี้จะทำการอ่านข้อมูลจาก file Shell.log มาทีละบรรทัดจนกว่าจะหมดถึงบรรทัดท้ายสุดของ file และแน่นอนว่า while loop นั้น สามารถใช้ break และ continue ได้เช่นเดียวกับ for loop ครับ

Select loop

เป็น loop ที่เรามักไม่ค่อยได้ใช้บ่อยนักแต่เมื่อใดที่เราต้องสร้าง script ที่ต้องมี choice ให้เลือกโดยที่ choice เหล่านั้นเป็น file หรือ path ที่มีอยู่แล้ว select loop จะช่วยย่นระยะเวลาให้เราได้เยอะเลยครับ โดย select loop จะสร้าง choice ให้เราโดยอัตโนมัติครับจากนั้นเราก็เพียงนำผลจาก choice ไปทำตามแต่ที่เราต้องการครับ ตัวอย่าง

```
select i in $(ls)
  do
    echo -n "Are you sure to select ${i} : "
    read CHOICE
    case ${CHOICE} in
        y) cat ${i} ;;
        n) break ;;
        q) exit ;;
      *) echo "Please select choice." ;;
    esac
done
```

จาก script เป็นการ ls ชื่อ file ที่อยู่ ณ path ปัจจุบันเข้ามาเป็น choice โดยใช้ select loop ครับ ผลที่ได้จะเป็นดังนี้ครับ

```
[root@server1 ~]# bash select.bash
1) anaconda-ks.cfg      5) install.log.syslog   9) testx
2) awk.bash             6) select.bash          10) while.bash
3) Desktop              7) test                 11) x.bash
4) install.log          8) test.bash
#? 10
Are you sure to select while.bash : y
#/bin/bash

ANUM=5
BNUM=1
while [ ${BNUM} -lt ${ANUM} ]
  do
          echo "${BNUM} < ${ANUM}"
    ((BNUM+=1))
done
#?
```

Shell script with file access.

ในหัวข้อนี้จะกล่าวถึงเรื่องหลัก ๆ คือ read , write และ execute file ครับ การที่เราจะสามารถ read , write หรือ execute file ได้หรือไม่นั้นขึ้นอยู่กับ ความเป็นเจ้าของ (owner) และก็สิทธ์ในการเข้าถึง (permission) ของ file นั้นๆ ด้วยครับ โดยเราสามารถดูข้อมูลเหล่านี้ได้โดยให้คำสั่ง ls –l หรือ ll ครับ

การ read เราสามารถใช้ คำสั่ง cat เพื่ออ่านข้อมูลจาก file หรือใช้การ redirect input (<) ดังที่เราได้ลองใช้ในตัวอย่าง while loop ก็ได้ครับ

นอกจากนี้แล้วเรายังใช้คำสั่ง head –n (n คือจำนวนบรรทัดที่ต้องการอ่านจากต้น file) เพื่ออ่านข้อมูลจากต้น file ตามจำนวนบรรทัดที่ต้องการ และใช้ tail –n (n คือจำนวนบรรทัดที่ต้องการอ่านจากท้าย file) เพื่ออ่านข้อมูลจากท้าย file ตามจำนวนบรรทัดที่ต้องการ

การ write เราสามารถใช้คำสั่ง echo “messages” ร่วมกับ การ redirect output (> , >>) เพื่อ write file ได้โดย > หมายถึงเขียนทับข้อมูลเดิม ส่วน >> เป็นการเขียนข้อมูลต่อจากข้อมูลเดิมที่มี ตัวอย่างเช่น echo “test messages” >> temp.log เป็นต้น

การ execute file นั้น ๆ จะต้องมี permission ในการ execute ได้ด้วยโดยการ execute file ที่เป็น shell script file นั้น สามารถทำได้โดย ระบุ shell ที่จะใช้ในการ run script ตามด้วย ชื่อ script หรืออีกวิธีหนึ่งคือ ./ชื่อscript วิธีนี้ script จะใช้ shell ที่ ระบุอยู่ที่ต้น script ในการ run script นั้น

ตัวอย่าง script ในการ read , write และ execute

```
INPUT_FILE=input.log
OUTPUT_FILE=output.bash

echo "echo This is output file."  >> ${INPUT_FILE}

while read line
  do
    echo "${line}" > ${OUTPUT_FILE}
    chmod 777 ${OUTPUT_FILE}
done < ${INPUT_FILE}
bash ${OUTPUT_FILE}
```

จาก script เราทำการ write message ‘echo This is output file.’ ลงที่ INPUT_FILE จากนั้น ใช้ while ในการ read INPUT_FILE แล้ว write ข้อมูลที่ได้ ลงที่ OUTPUT_FILE แล้วใช้คำสั่ง chmod เปลี่ยน permission ให้กับ OUTPUT_FILE เพื่อให้ execute ได้จากนั้น ทำการ execute OUTPUT_FILE ซึ่งในขณะนี้ ภายใน OUTPUT_FILE มี ข้อความดังนี้

echo This is output file.

ซึ่งข้อความข้างต้น เป็น command ในการแสดงข้อความ This is output file. ดังนั้นเมื่อทำการ execute OUTPUT_FILE แล้วจึงได้ผลคือข้อความ This is output file. ปรากฏขึ้นนั่นเอง

Keyboard input (การรับข้อมูลจาก keyboard)

การรับข้อมูลจาก keyboard ใช้คำสั่ง read ในการรับค่า เพื่อรอให้ผู้ใช้ป้อนค่าผ่าน keyboard จากนั้นจึงนำข้อมูลที่ได้ไปใช้ในการตัดสินใจต่อไป เช่น

```
echo –n “Please enter your number : “
read number
echo “Your data is : ${number}”
```

ความหมายคือ script จะแสดงข้อความ Please enter your number : จากนั้นเมื่อผู้ใช้ป้อนค่าและ enter ข้อมูลที่ป้อนเข้าไปจะถูกเก็บไว้ที่ตัวแปร number จากนั้นเราสามารถใช้ตัวแปร number นี้เพื่อเป็นข้อมูลของ script ต่อไป

การรับ Argument

Shell script นั้นสามารถที่จะรับ argument ได้โดย นับจาก argument ที่ 0 – n โดย argument ที่ 0 นั้นหมายถึงชื่อของ script ที่กำลัง run อยู่นั่นเอง ส่วน argument ที่ 1 – n เป็นสิ่งที่เราพิมพ์ตามหลังชื่อ script ตามลำดับ โดยการอ้างถึง argument นั้น อ้างเช่นเดียวกันกับการอ้างตัวแปร โดยใช้เครื่องหมาย $ นำหน้า ดังตัวอย่างต่อไปนี้

Script ชื่อ GetArgument.bash

```
echo “Argument 0 : ${0}”
echo “Argument 1 : ${1}”
echo “Argument 2 : ${2}”
```

ทดสอบ run script

```
bash GetArgument.bash Shell Script
Argument 0 : GetArgument.bash
Argument 1 : Shell
Argument 2 : Script
```

***

- ค่า **$0** คือชื่อ ของ Script ที่ run
- ค่า **$1...$n** คือค่าที่พิมพ์ต่อจากชื่อ Script
- ค่า **$#** คือ จำนวนของ Argunent list
- ค่า **$$** คือค่า PID ของ Process ปัจจุบัน
- ค่า **$?** คือค่า exit status ที่ได้หลังจาก execute คำสั่งโดยหาก Execute สำเร็จ ค่าที่ได้จะเป็น 0

Function of shell script

การประกาศ function ใน shell script นั้นมีประโยชน์อย่างยิ่งเนื่องจากคำสั่งบางชุดที่เราต้องการเรียกใช้บ่อยๆ นั้นเราไม่จำเป็นต้องเขียนหลายครั้ง หลายที่ เราสามารถสร้างเป็น function เพื่อเรียกใช้ทุกเวลาที่ต้องการ แต่ มีข้อแม้ว่า function นั้นต้องอยู่ส่วนบนก่อนที่จะเรียกใช้ function โดยรูปแบบที่นิยมใช้มีดังนี้

1.
```
function hello
{
    echo “Hello”
 }

 hello
```
2.
```
hello()
{
    echo “Hello”
}

hello
```

ทั้งสองแบบสามารถใช้ได้เหมือนกันครับโดยการเรียกใช้ function นั้นเพียงแต่ระบุชื่อ function ก็สามารถเรียกใช้ function ได้แล้วครับ

Shell script with FTP

บางองค์กรอาจใช้การ FTP file ในการ transfer file จาก server หนึ่งไปยัง server หนึ่งครับ ส่วนนี้จึงอยากให้ทำความเข้าใจดัง script ครับ

```
FILENAME=test.log

ftp -i -n -v 192.168.7.128 << PROC
   user youruser yourpasswd
   cd
   put ${FILENAME}
bye
PROC
```

เป็นการเข้าไปสู่การใช้งาน ftp แต่สามารถที่จะใช้ Shell command ในการอ้างตัวแปรต่างๆ ร่วมกับการ ftp ได้ครับแต่ข้อเสียคือเรา ต้อง HardCode username และ password ลงไปใน script ครับ ทำให้เรื่อง Security นั้นก็อ่อนลงไปด้วย

Shell Script with SSH.(Secure Shell)

นอกจากการใช้ Ftp ในการ Transfer file ระหว่างเครื่องแล้ว ที่นิยมมากอีกอย่างหนึ่ง คงหนีไม่พ้น ssh ครับ ต่อไปนี้เป็น command ในการ copy และ execute command ผ่าน ssh ครับ

scp localfile user@hostname:/destination_path : copy file ขึ้นไปยัง server นั้นๆ
scp user@hostname:/remote_file /local_path : copy file จาก server ลงมาที่เครื่อง

*หากเป็นการ copy dir ให้ใช้ option –r เช่น
scp –r testdir user@hostname:/destination_path
ssh [ip,hostname] “remote command” : เป็นการ run คำสั่งผ่านเครื่องอีกเครื่องหนึ่งเช่น
ssh 192.168.0.100 “ls”

แต่เนื่องจากการ copy หรือ การ execute command ผ่าน ssh นั้นต้องป้อน password เพื่อทำการ login เสียก่อนจึงจะเข้าไป access เครื่องนั้นๆ ได้ทำให้เป็นปัญหาในการ เขียนเป็น script ดังนั้นหากต้องการให้ script สามารถทำการ copy หรือ execute command ผ่าน ssh ได้นั้นจำเป็นต้องทำการ generate key.ขึ้นมาเพื่อ authenticate เข้าไปยังโดยที่ไม่ต้องถาม password ทั้งนี้จะต้องได้รับอนุญาตจากทางด้าน Security ขององค์กรก่อนนะครับ โดยวิธี Generate key ทำได้ดังนี้ครับ

- Login เครื่องด้วย User ใดๆ ที่ต้องการสร้างไฟล์ SSH Key
- ใช้คำสั่ง ssh-keygen –t dsa –b 1024 แล้ว enter 3 ครั้ง
- ขณะนี้จะมี file $HOME_DIR_PATH/.ssh/id_dsa.pub เกิดขึ้น
- เปลี่ยนชื่อไฟล์ที่ $HOME_DIR_PATH/.ssh/id_dsa.pub เป็นชื่ออื่นที่ไม่ใช่ชื่อ authorized_keys
- นำไฟล์ที่ได้เปลี่ยนชื่อแล้วไปวางไว้ที่ เครื่องอื่นๆ (ปลายทาง) ที่ PATH เดียวกัน
- Cat file ดังกล่าวไปต่อ file authorized_keys ด้วยคำสั่ง cat filex >> authorized_keys
- ลองทำการ Login เข้าไปยังเครื่องนั้นๆจะเห็นว่าสามารถเข้าไปยังเครื่องนั้นๆ โดยไม่ต้องป้อนรหัสผ่านแล้ว

วิธีการ Login ด้วย ssh สามารถทำได้ดังนี้

- ใช้คำสั่ง ssh ตามด้วย IP เช่น

  ssh 192.168.171.128

- ใช้คำสั่ง ssh ตามด้วย username@IP เช่น

  ssh srsc@192.168.141.129

  หมายความว่าต้องการ Login เข้าเครื่อง 192.168.171.128 ด้วย User srsc นั่นเอง

Trap interrupt protection (การป้องกันการ Interrupt (Trap))

ในการเขียน Shell script เพื่อให้ user ใช้งานเพียงอย่างเดียว เช่น ให้เลือกป้อนเลข menu ตามที่มีให้ป้อน โดยจะไม่ยอมให้ user ได้ Prompt ของ Shell ไปนั้นเราจำเป็นต้องมีการ ป้องกันการ Interrupt ด้วยการกดปุ่มต่างๆ เช่น Ctrl+c , Ctrl+d เป็นต้น เราสามารถป้องกันได้ดังนี้ครับ

Signal	Meaning
INT	ctrl+c
QUIT	ctrl+\
TSTP	ctrl+z

Command ที่จะทำการป้องกันการกดปุ่ม signal เหล่านี้คือ command trap [signal] โดยการใช้งาน command นี้จะต้อง นำไปวางไว้ในส่วนต้น ของ script ดังนี้

```
#/bin/bash
trap “” INT      #กันการกดปุ่ม ctrl+c
trap “” TSTP    #กันการกดปุ่ม ctrl+z

echo –n “Test no interrupt : “
read  test
exit
```

COMMAND TIPs

- ถ้ามี file /etc/nologin อยู่ user ใดๆ จะไม่สามารถ Login เข้าเครื่องได้ นอกจาก root เท่านั้น
- nc –c –z –w 3 [ip,hostname] [portnumber] : เช็คว่า port นั้นๆ เปิดอยู่หรือไม่
- ping –c 1 : เป็นการ ping โดยระบุว่าจะ ping กี่ package
- su –l root –c “command” : เป็นการ run command ด้วย root (ต้องแก้ไข /etc/pam.d/su ก่อน บรรทัดที่ 3 ที่ลงท้ายด้วย trust used_uid ให้เอา # หน้าบรรทัดออก แล้ว user ที่จะ run command นี้ได้ต้องอยู่ใน group wheel ด้วย
- ls |xargs –n1 command เช่น

  ls |xargs –n1 rm : คือการนำผลลัพธ์จากคำสั่งข้างหน้ามาเป็น argument ของคำสั่งหลัง
- hastatus : ดู status ของ cluster บน solaris
- vxtask list : ดู status ของการทำ mirror (solaris)
- hagrp –switch $service_groups –to hostname : สลับให้ cluster ไป active ที่ hostname (solaris)
- hastatus –sum : ดู service group ทั้งหมด (solaris)
- ใต้ path /usr/lib/osa/bin/

  ./lad : check device A1000

  ./healtchk –a : check status A1000

  ./raidutil –c c0d0t0s0 -B : check battery A1000
- lspci : ดู pci card
- lsusb : ดู usb
- lsmod : ดู module
- rpm –qi kernel : ดูรายละเอียด kernel
- awk “NF > 0” $filename : ตัดบรรทัดว่างออกจาก filename
- awk “NR == 5” $filename : เลือกเอาบรรทัดที่ 5 ของ filename
- awk “NF == 3” $filename : เลือกเอาเฉพาะบรรทัดที่มี 3 field
- cat $filename |awk ‘{length($1) == “4”}’ : เลือก field ที่มีจำนวนอักระเท่าที่ต้องการ
- telnet [ip,hostname] [portnumber] : telnet port ใดๆ
- ถ้า a=2,b=3,c=4

  echo $((a+b+c)) : บวกค่าได้เลย

  echo $((a*b+c)) : คำนวณค่าได้
- ใน awk command

  $NF = ‘${value}’ : คือการให้ค่ากับตัวแปร NF

  substr(abc,2,2) : เป็นการตัด อักขระหมายถึงเริ่มนับจากตัวที่ 2 ไปอีก 2 ตัวเริ่มจากตัวมันเอง จะได้ bc

tolower(A) : จะได้ a

toupper(a) : จะได้ A

- route add –net networkip : เป็นการ add route ด้วย network ip
- route add –host ip : add route ด้วย ip
- route del –net networkip :del network
- route del –host ip : del ip
- ipcs : ดูการใช้ hold memory resource
- ipcrm [shmid,semid] : คืน resource ให้ระบบ
- tcpdump ip src host $IP : ดู connection ที่มาจาก $IP
- tcpdump ip dst host $IP : ดู connection ที่วิ่งไปที่ $IP
- tcpdump ip host $IP : ดู connection ที่วิ่งไป – มา ระหว่าง $IP
- tcpdump port 22 : ดู connection ที่วิ่งผ่าน port 22
- tcpdump src port 22 : ดู connection ที่วิ่งเข้ามาด้วย port 22
- tcpdump dst port 22 : ดู connection ที่วิ่งออกไปผ่าน port 22
- tcpdump –I eth0 : ดู connection ที่วิ่งผ่าน interface eth0
- tcpdump –c 10 : ทำ 10 ครั้ง
- lsattr : ดูว่า file หรือ dir ใดๆ มีการ set attribute หรือไม่
- chattr +i $filename : เปลี่ยน ให้ $filename อ่านได้อย่างเดียว root ก็ลบไม่ได้ถ้าไม่เปลี่ยนคืน
- chattr –i $filename : เปลี่ยน attribute คืน
- chattr +a $filename : ให้ file นี้เพิ่มข้อความได้อย่างเดียวห้ามลบ
- chattr –a $filename : กลับสู่ attribute เดิม
- ps –ef |grep $processname |awk –F" " '{print $2}' |kill -9 : ค้นหาและ kill proces
- echo "abc" |sed s/a/X/g เปลี่ยน ตัว a เป็น X
- echo "abc" |sed '/a/d' : ลบบรรทัดที่มีตัว a
- cat $file |sed –n "10,20 p" : ตัดเอาบรรทัดที่ 10-20 ของ $file
- cd ~/ : คือ ไปอยู่ที่ home directory path
- cd - : คือ cd ไป path ก่อนหน้าที่จะเปลี่ยนมา path ปัจจุบัน