การทดลองที่ 7 ยูนิกซ์เชลสคริปต์

จุดประสงค์

1. ศึกษาการใช้งานภาษาสคริปต์ในระบบปฏิบัติการยูนิกซ์
2. เพื่อให้นักศึกษาสามารถเขียนสคริปต์ในระบบปฏิบัติการยูนิกซ์อย่างง่ายได้

คำสั่ง

จงศึกษาทฤษฎีบทต่างๆ จากนั้นทดลอง และค้นคว้าเพื่อตอบคำถามท้ายการทดลอง

ทฤษฎี

"If anything can go wrong, it will"
- Murphy's Law

เชลสคริปต์ระบบปฏิบัติการยูนิกซ์เป็นต้นแบบของแบตช์ไฟล์ในระบบปฏิบัติการดอส โดยต้องการรวมคำสั่งเป็นลำดับไว้ให้สะดวกต่อการเรียกใช้ อีกทั้งปรับเปลี่ยนลักษณะการทำงานได้ตามเงื่อนไขสภาพแวดล้อม เชลสคริปต์จึงประกอบไปด้วยคำสั่งเรียกใช้โปรแกรมอื่นและคำสงวนสำหรับเชลสคริปต์เอง

ไวยากรณ์พื้นฐาน

บรรทัดแรกของเชลสคริปต์ไฟล์ต้องระบุถึงโปรแกรมที่ใช้ตีความตามรูปแบบนี้

```
#!/bin/sh
```

หลังจากเครื่องหมาย # ถือเป็นหมายเหตุ โปรแกรมไม่ตีความไปจนสุดบรรทัดนั้นๆ

```
# This is a remark.
```

เราสามารถออกคำสั่งเดี่ยวหรือกลุ่มคำสั่งในไฟล์เชลสคริปต์ได้เสมือนเรียกใช้จากเชลพรอมต์

```
cat file1.txt file2.txt > file3.txt
```

เราสามารใช้ ‘ หรือ “ เพื่อเริ่มต้นและจบสายอักษรได้ตามความเหมาะสม แต่ต้องขึ้นต้นและจบด้วยอักษรตัวเดียวกัน เช่น

```
echo “It’s my first time!”
echo ‘He said “Do you marry me?”.’
```

เมื่อต้องการจบการทำงานเชลสคริปต์ด้วยคำสั่ง

```
exit
```

โดยทั่วไปแล้ว หากโปรแกรมใดทำงานจบลงตามปรกติจะให้ค่าคืนกลับ (return value) เป็น 0 แต่ถ้าเกิดความผิดพลาดขึ้นและต้องการระบุค่าเพื่อจำแนกความผิดพลาดดังกล่าว สามารถระบุต่อท้ายคำสั่ง exit ได้ทันที เช่น

```
exit 13
```

การอ้างอิงตัวแปร

การอ้างอิงถึงตัวแปรสภาพแวดล้อมทำได้โดยใช้รูปแบบ ${variablename} เช่น

```
echo Current shell is “${SHELL}”
echo Your email address is ${LOGNAME}@ce.kmitl.ac.th
echo “${HOME}” == “$HOME”
```

ตัวแปรผ่านค่า (argument) อ้างอิงด้วยรูปแบบ $1 $2 ... $9 และสามารถเลื่อนค่า (shift) ได้ด้วย เช่น

```
echo $1 $2 $3
shift
echo $1 $2 $3
```

หากต้องการอ้างถึงตัวแปรผ่านค่าทั้งหมดสามารถใช้ $* โดยที่

```
"$*" = "$1 $2 $3 $4 ... $n"
```

สามารถอ้างอิงจำนวนตัวแปรผ่านค่าทั้งหมดด้วย $# เช่น

```
Shell Prompt$ cat sumpapi.sh
#!/bin/sh
echo "Number of argument(s) = $#"
Shell Prompt$ ./sumpapi.sh aa bb cc dd ee
Number of argument(s) = 5
```

การตั้งค่าตัวแปรใช้รูปแบบ variablename=value โดยที่ value นั้นมองเป็นสายอักษร เช่น

```
VAR1="123 456"
VAR2="${VAR1} 789"
```

ในบางกรณีเราอาจต้องการให้ค่าตัวแปรสภาพแวดล้อมจากเชลสคริปต์คงอยู่แม้จบการทำงานไปแล้ว สามารถทำได้โดยสั่ง export แล้วตามด้วยชื่อตัวแปรนั้นๆ เช่น

```
export VAR1 VAR2
```

ในบางกรณีเราอาจต้องการตั้งให้ค่าตัวแปรสภาพแวดล้อมเป็นไปตามผลลัพธ์ของคำสั่งที่ระบุ สามารถทำได้โดยใช้ ` คร่อมคำสั่ง เช่น

```
VAR3=`expr ${RANDOM} / 100`
```

ข้อควรทราบคืออักษรพิเศษใดๆ ไม่มีผลขณะอยู่ในสายอักษรที่คร่อมด้วย ' ฉะนั้น

```
echo '${HOME}' ' <> ' "${HOME}"
```

และตัวแปรต่อไปนี้เป็นตัวแปรพิเศษ

$?	หมายถึงค่าส่งกลับของคำสั่งที่เพิ่งกระทำล่าสุด
$$	หมายถึงหมายเลขประจำโพรเซสของโปรแกรมปัจจุบัน(ก็คือเชลสคริปต์นี้นั้นเอง)
$!	หมายถึงหมายเลขประจำโพรเซสของโปรแกรมฉากหลังล่าสุด

<u>ประโยค if</u>

รูปแบบคือ

```
if condition
then
     command(s)
elif condition
     command(s)
else
     command(s)
fi
```

โดย condition นั้นคือคำสั่งที่ต้องส่งค่ากลับ หากค่าที่ส่งกลับคืนเป็น 0 ถือว่าเงื่อนไขเป็นจริง นอกนั้นเป็นเท็จ ตามปรกติต้องอาศัย [(ตัวจริงคือคำสั่ง test ศึกษาข้อมูลเพิ่มเติมได้จาก "man test") เข้าช่วยเพื่อประมวลผลเงื่อนไข ส่วน elif และ else เป็นส่วนขยายเพิ่มเติมกรณีเพื่อความสะดวกในการเขียนโปรแกรม ไม่จำเป็นต้องระบุหากไม่ได้ใช้งาน ตัวอย่างเช่น

```
if [ "$1" != "" ]
then
     echo First argument = "$1".
else
     echo Please add an argument.
```

ประโยค for

รูปแบบคือ

```
for var in list
do
    command(s)
done
```

โดยที่ var เป็นตัวแปรชี้ลำดับซึ่งมีค่าไล่จากต้นรายการที่ระบุใน list ไปจนหมดรายการ ตัวอย่างเช่น

```
for j in /tmp/*.bak ~/*.bak
do
    rm ${j}
done
```

หรือ

```
for i in 0 1 2 3 4 5
do
    for j in 0 1 2 3 4 5 6 7 8 9
    do
        mkdir ~/tmp${i}${j}
    done
done
```

ประโยค while และ until

รูปแบบของ while คือ

```
while condition
do
    command(s)
done
```

รูปแบบของ until คือ

```
until condition
do
    command(s)
done
```

ประโยค case

รูปแบบคือ

```
case str in
pattern_a
    command(s)
    ;;
pattern_b
    command(s)
    ;;
pattern_c
    command(s)
    ;;
pattern_d
    command(s)
    ;;
*)
    command(s)
    ;;
esac
```

โดยที่ *) หมายถึงกรณีไม่เข้ากับรูปแบบอื่นใดเลย

การอ่านค่าเข้าด้วยคำสั่ง read

ในกรณีที่ต้องการรับค่าจากอินพุตมาตรฐาน (หรือคีย์บอร์ด) สามารถใช้คำสั่ง read ดังตัวอย่างต่อไปนี้

```
while read string   # Type ^D to end
do
     echo You input "${string}".
done
```

หรือ

```
ans=x
until [ ${ans} = "y" -o ${ans} = "Y" ]
do
     echo "Do you want to exit? [y/n] \c"
     read ans
done
```

นอกจากนี้ยังมีคำสั่งที่เป็นประโยชน์ต่อการเขียนเชลสคริปต์ให้อ่อนตัวคือ set expr และ eval เป็นต้น

การทดลอง

1. จงสร้างไฟล์ test-1.sh ซึ่งบรรจุคำสั่งต่อไปนี้

```
#!/bin/sh
ans=x
until [ ${ans} = "y" -o ${ans} = "Y" ]
do
     echo "Do you want to exit? [y/n] \c"
     read ans
done
exit 0    # Normal terminate
```

แล้วออกคำสั่งเปลี่ยนโหมดของไฟล์นี้ให้ปฏิบัติงานได้ด้วยคำสั่งต่อไปนี้

```
Shell Prompt$ chmod u+x test-1.sh
```

ทดลองเรียกใช้ด้วยคำสั่ง

```
Shell Prompt$ ./test-1.sh
```

หมายเหตุ การเปลี่ยนโหมดของไฟล์ต้องทำเสมอเพื่อให้เรียกใช้งานเชลสคริปต์ได้ ส่วนการเรียกใช้นั้นสามารถเรียกแบบเต็มรูปแบบ เช่น /home/adek/shellcodes/test-1.sh หรือระบุพาธเป็น . หากไดเรกทอรีปัจจุบันอยู่ตรงกับไฟล์ก็ได้ มิฉะนั้นต้องตั้งค่าตัวแปรสภาพแวดล้อม PATH ให้อ้างถึงไดเรกทอรีปัจจุบันหรือไดเรกทอรีที่ไฟล์เชลสคริปต์นั้นอยู่ด้วย เช่น

```
PATH=${PATH}:. ; export PATH
```

หรือ

```
PATH=${PATH}:/home/adek/shellcodes ; export PATH
```

2. จงสร้างไฟล์ test-2.sh ซึ่งบรรจุคำสั่งต่อไปนี้

```
#!/bin/sh
TIME=`date | (read u v w x y z; echo ${x})`
echo "Current time is ${TIME}"
exit 0    # Normal terminate
```

แล้วทดลองเรียกใช้ด้วยคำสั่ง

```
Shell Prompt$ date ; ./test-2.sh
```

3. จงสร้างไฟล์ test-3.sh ซึ่งบรรจุคำสั่งต่อไปนี้

```
#!/bin/sh
for i in 0 1 2 3
do
```

```
            for j in 0 1 2 3 4 5 6 7 8 9
            do
                > ./tmpfile${i}${j}
            done
        done
```

แล้วทดลองเรียกใช้ด้วยคำสั่ง

```
Shell Prompt$ ./test-3.sh ; (ls -al tmpfile* | more)
```

คำถามท้ายการทดลอง

จงพิจารณาเชลสคริปต์ต่อไปนี้

```
#!/bin/sh
### 1st area ###
cat > ftp.tmp.$$ << EOF
open ftp.nectec.or.th.
user ftp password@
cd /pub/linux-distributions/Debian
get README
bye
EOF
### 2nd area ###
ftp -n < ftp.tmp.$$ >/dev/null 2>&1
FTPRC=$?
if [ -f ./ftp.tmp.$$ ]
then
        rm ./ftp.tmp.$$ && echo "ftp.tmp.$$ has been removed."
fi
if [ ${FTPRC} != 0 ]
then
        echo "Transfer Failed!" ; exit 1
else
        echo "Transfer Completed!"
        if [ -s ./README ]
        then
                mv ./README "./README of ${LOGNAME}" && \
                echo "File has been renamed." || \
                echo "File has NOT been renamed."
        fi
fi
exit 0
```

แล้วระบุวัตถุประสงค์ของเชลสคริปต์นี้ รวมถึงอธิบายหลักการทำงานในพื้นที่ทั้งสองด้วย

ทิ้งท้าย

นักศึกษาสามารถค้นคว้าเพิ่มเติมได้ที่ http://steve-parker.org/sh/sh.shtml