Lecture 11

โครงสร้างควบคุมการทำงานแบบมีการเลือก

คำสั่งควบคุมโครงสร้างแบบมีการเลือกสำหรับการเขียน shell script มีหลายแบบเพื่อให้สอดคล้องกับงานที่มีหลายลักษณะ โครงสร้างแบบง่ายที่สุดคือ if.. then ซึ่งมีโครงสร้างทั่วไปดังนี้

```
if  test-command
    then
         commands
fi
```

เมื่อ test-command ประกอบด้วยคำสั่ง test และ argument ที่ต้องให้คำสั่ง test ทดสอบ ในกรณีที่ผลการทดสอบเป็นจริง คำสั่งtest จะคืนสถานะการเลิกทำงาน (exit status) เป็น 0 ในกรณีที่เป็นเท็จจะคืนค่าอื่นซึ่งไม่ใช่ 0 ให้แก่ shell script เพื่อนำไปใช้ในการกำหนดการทำงานของโครงสร้าง if ต่อไป

โปรแกรมต่อไปนี้ทำหน้าที่อ่านคำภาษาอังกฤษจากแป้นพิมพ์ 2 คำ และทำการเปรียบเทียบคำทั้งสอง ในกรณีที่เป็นคำเดียวกันจะแจ้งให้ผู้ใช้ทราบ

```
$ cat if1
echo -n "word1: "
read word1
echo -n "word2: "
read word2

if  test "$word1" = "$word2"        # หรือ["$word1" = "$word2" ]
    then
         echo Match
fi
```

หมายเหตุ

ก. โดยปกติเมื่อแสดงผลเสร็จ echo จะขึ้นบรรทัดใหม่ ในกรณีที่ไม่ต้องการให้มีการขึ้นบรรทัดใหม่ บังคับด้วย option -n

ข. read เป็นคำสั่งภายในของ bash สำหรับอ่านข้อมูล 1 บรรทัดจากผู้ใช้ เก็บในตัวแปรที่กำหนด

ค. เงื่อนไข test "$word1" = "$word2" โดยทั่วไปนิยมย่อเป็น ["$word1" = "$word2"] ในเอกสารต่อไปจะใช้ [] แทนคำสั่ง test

โครงสร้าง if จะทำงานตามคำสั่งที่กำหนดไว้หลังคำสำคัญ then หากคำสั่ง test คืนค่าสถานะการเลิกทำงานเป็นจริง (ค่าเป็นศูนย์) และออกจากโครงสร้าง if โดยทำงานตามคำสั่งที่กำหนดไว้หลังคำสำคัญ fi ในกรณีที่ค่าสถานะการเลิกทำงานของคำสั่ง test เป็นเท็จ (ค่าไม่เป็นศูนย์)

โปรแกรมต่อไปนี้ใช้ในการตรวจสอบว่าผู้ใช้กำหนด command line argument หรือไม่ โดยตรวจสอบจากค่าของตัวแปร $# ดังนี้

```
$ cat chkargs1
if   [ $# -eq 0 ]
    then
         echo "You must supply at least one argument."
         exit 1
fi
echo "Program running."
```

หมายเหตุ

ก. การเปรียบเทียบจำนวน argument กับ 0 ว่าเท่ากันหรือไม่ เป็นการเปรียบเทียบจำนวนเต็มต้องใช้ -eq เป็นเครื่องหมายเปรียบเทียบ

ข. exit เป็นคำสั่งสำหรับเลิกการทำงานของเชลล์ พร้อมคืน Return code ซึ่งเป็นจำนวนเต็มให้ผู้เรียก

ค. ตัวอย่างการทำงานเป็นดังนี้

```
$ chkargs1                      # เรียกใช้โดยไม่มี argument
You must supply at least one argument.

$ chkargs1 abc                  # เรียกใช้โดยมี "abc" เป็น argument
Program running.
```

โปรแกรม chkargs1 ทำหน้าที่ทดสอบว่าผู้ใช้กำหนด command line argument เมื่อมีการเรียกใช้โปรแกรมนี้หรือไม่ ในกรณีที่มีการกำหนด argument ตั้งแต่หนึ่งตัวขึ้นไป ค่าของตัวแปร $# จะเป็นจำนวนของ argument นั้น ในกรณีที่ผู้ใช้ไม่ได้กำหนด argument ค่าของตัวแปรนี้จะมีค่าเป็น 0 จึงสามารถใช้เงื่อนไข $# -eq 0 ในการทดสอบได้ การตรวจสอบจำนวน argumentเช่นที่ใช้ในโปรแกรม chkargs เป็นองค์ประกอบที่สำคัญของ shell script ทุกโปรแกรม เพื่อป้องกันไม่ให้โปรแกรมทำงานในกรณีที่ผู้ใช้ไม่กำหนด argument หรือกำหนดแต่รูปแบบหรือจำนวนของ argument ไม่ถูกต้อง นอกจากนี้แล้วในกรณีที่ผู้ใช้กำหนด argument เป็นชื่อแฟ้มที่มีข้อมูลจำเป็นที่ต้องใช้ในโปรแกรม ควรต้องตรวจสอบว่าแฟ้มที่ผู้ใช้ระบุมีอยู่จริง และผู้ใช้มีสิทธิในการอ่านแฟ้มนั้น รายละเอียดของคำสั่ง test ศึกษาได้จากแฟ้ม test.txt หรือจาก online manual page โดยใช้คำสั่ง man test

การตรวจสอบ argument โดยใช้โครงสร้าง if ที่ตอนต้นของ shell script เป็นการทำงานที่สำคัญมาก เพราะหากผู้ใช้เรียกโปรแกรมโดยไม่มี argument หรือมี argument ไม่ถูกต้องครบถ้วน ก็ควรที่จะต้องเตือนผู้ใช้และเลิกทำงาน ไม่ควรที่จะดำเนินการต่อไปซึ่งอาจทำให้เกิดความเสียหายขึ้นได้ อย่าลืมว่า "Garbage in, garbage out."

ของแถม

```
Garbage data ----> Perfect model ----> Garbage results
```

และ

```
Perfect data ----> Garbage model ----> Garbage results
```

หมายเหตุ

เนื่องจากคำสั่งทดสอบในระบบปฏิบัติการ Unix มีชื่อเฉพาะว่า test ดังนั้นจึงไม่ควรตั้งชื่อโปรแกรมที่เขียนขึ้นเป็น test เนื่องจากเวลาเรียกใช้งานจะหมายถึงการเรียกใช้โปรแกรม test ที่เป็นโปรแกรมอรรถประโยชน์ในระบบแทนที่จะเป็นโปรแกรมของผู้ใช้ โปรแกรมต่อไปนี้ใช้สำหรับทดสอบว่า argument ที่ผู้ใช้กำหนดเป็นแฟ้มปกติ (regular file) หรือไม่?

```
$ cat is_ordfile
if [ $# -eq 0 ]
   then
       echo "You must supply at least one argument."
       exit 1
fi

if [ -f "$1" ]
   then
       echo "$1 is a regular file."
   else
       echo "$1 is NOT a regular file."
       exit 2
fi
```

ข้อสังเกต

การทดสอบมีสองตอน ตอนแรกเป็นการทดสอบว่าผู้ใช้กำหนด argument หรือไม่ ส่วนตอนที่สองเป็นการทดสอบว่า argument ที่ผู้ใช้กำหนดเป็นแฟ้มปกติ (regular file) หรือไม่ เนื่องจากโปรแกรมมีการตรวจสอบสองตอน จึงกำหนดให้มีการคืน return code เป็นสองค่า คือ 1 - ไม่มี argument และ 2 - argument ไม่ใช่แฟ้มธรรมดา โปรแกรมนี้ใช้ โครงสร้าง if..then..else..fi ซึ่งรูปแบบโครงสร้างดังนี้

```
if  test-command
    then
        commands
    else
        commands
fi
```

การจัดการ error (ตอนที่ ๑)

โดยปกติทั่วไปหากผู้ใช้เรียกใช้ Utilities ไม่ถูกรูปแบบ วิธีการที่ในระบบปฏิบัติการ Unix รุ่นก่อนๆ ใช้คือแจ้งให้ผู้ใช้ทราบว่า รูปแบบการใช้งาน (synopsis) ที่ถูกต้องเป็นอย่างไรก่อนเลิกการทำงาน ในระบบปฏิบัติการรุ่นก่อนจะแสดงรูปแบบการใช้งานทั่วไป แล้วเลิกทำงาน สำหรับระบบปฏิบัติการในปัจจุบันมีการปรับปรุงให้แสดงความช่วยเหลือให้ดียิ่งขึ้นและตรงกับการทำงานมากขึ้น เช่นการใช้คำสั่ง cp โดยไม่มีชื่อแฟ้มใด เป็น argument เลย ปรากฏผลดังนี้

```
$ cp                        # ไม่มีชื่อแฟ้มใด เป็น argument
cp: missing file operand
Try `cp --help' for more information.

$ cp xx.txt                 # มีชื่อแฟ้มต้นทางแต่ไม่มีชื่อแฟ้มปลายทาง
```

```
cp: missing destination file operand after `xx.txt'
Try `cp --help' for more information.

$ cp -z xx.txt yy.txt          # เรียกใช้ option ที่ไม่มี
cp: invalid option -- 'z'
Try `cย่าp --help' for more information.

$ cp --help                    # ขอความช่วยเหลือ
Usage: cp [OPTION]... [-T] SOURCE DEST
   or:  cp [OPTION]... SOURCE... DIRECTORY
   or:  cp [OPTION]... -t DIRECTORY SOURCE...
Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

     Mandatory arguments to long options are mandatory for short options
too.
       -a, --archive                same as -dR --preserve=all
       ...                            ...
```

แบบฝึกหัด

กำหนดให้ในไดเรกทอรีปัจจุบันมีเฉพาะแฟ้ม `xx.txt` และ `aa.txt` คำสั่งต่อไปนี้ให้ `error message` หรือไม่ อย่างไร

```
$ cp  zz.txt       # มีเฉพาะชื่อแฟ้มต้นทาง และเป็นแฟ้มที่ไม่มีอยู่จริง
$ cp  zz.txt  yy.txt     # มีชื่อแฟ้มต้นทางและปลายทาง แต่แฟ้มต้นทางไม่มีอยู่จริง
$ cp  xx.txt  aa.txt     # มีชื่อแฟ้มต้นทางและปลายทาง แฟ้มต้นทางมีอยู่จริง
                         # แฟ้มปลายทางมีอยู่แล้ว
```

คำถาม

---

1. จากการทดลอง สามารถเขียนขั้นตอนวิธีการตรวจสอบ `argument` ได้หรือไม่? การแสดงความช่วยเหลือผู้ใช้ในกรณีเช่นนี้มีจุดเด่นจุดด้อยอย่างไร จงอิปราย และ ในกรณีที่แฟ้มต้นทางมีจริง และแฟ้มปลายทางมีอยู่แล้ว (คัดลอกแฟ้มต้นทางไปทับข้อมูลเดิมในแฟ้มปลายทาง) การจัดการ เหมาะสมหรือไม่ ในกรณีที่ไม่เหมาะสมควรปรับปรุงอย่างไร

2. คำสั่ง `cp` มีรูปแบบการใช้งานอย่างง่ายเป็น

```
cp    [option]   <ชื่อแฟ้มต้นทาง(ต้นฉบับ)>   <ชื่อแฟ้มปลายทาง(สำเนา)>
```

การทดสอบคำสั่ง `cp` ที่ทำมาแล้วทั้งหมด ครบถ้วนทุกกรณีแล้วหรือไม่? หากยังไม่ครบขาดกรณีในบ้าง? และควรทดสอบอย่างไร?

การจัดการ `error` (ตอนที่ ๒)

เนื่องจากโพรเซสมีการเชื่อมต่อกับแฟ้ม `standard input, standard output,` และ `standard error` จึงสามารถแยก `output` และ`error message` ออกจากกันได้ โปรแกรมภาษา C

```
fprintf(stdout, "result output\n");          /* ส่งผลลัพธ์ออกทาง standard output */
fprintf(stderr, "error message\n");          /* ส่งผลลัพธ์ออกทาง standard error  */
```

และ C++

```
cout << "result output\n";          // ส่งผลลัพธ์ออกทาง standard output
cerr << "error message\n";          // ส่งผลลัพธ์ออกทาง standard error
```

เนื่องจาก echo เป็นโปรแกรมที่ทำการส่งข้อความที่กำหนดออกทาง standard output เสมอ เมื่อนำมาใช้ในการแสดง error message จึงต้องเปลี่ยนทิศทางจาก standard output ไปยัง standard error

โปรแกรม chkargs2 ต่อไปนี้ หากผู้ใช้ไม่กำหนด argument จะแสดงรูปแบบการใช้งานด้วยคำสั่ง echo และเปลี่ยนทิศทางไปยัง standard error

```
$ cat chkargs2
echo "check arguments version 2"          # standard output
if [ $# -eq 0 ]
   then
        echo "Usage: chkarg2  argument ..."  1>&2
                                  # redirect output to standard error
        exit 1
fi
echo "Program running."           # standard output
```

การทดสอบโปรแกรมทำดังนี้

```
----------------------
$ chkargs2  test                  # (1). มี argument
check arguments version 2
Program running.

$ chkargs2                        #  (2). ไม่มี argument -- เกิด error
check arguments version 2
Usage: chkarg2  argument ...      # error แสดงวิธีการใช้งาน

$ chkargs2   2>errlog             # (3). error และเปลี่ยนทิศทางไปยัง standard error
check arguments version 2         # standard output
$ cat errlog                      # error ในแฟ้ม errlog
Usage: chkarg2  argument ...

$ chkargs2  1>oplog               #  (4). error และเปลี่ยนทิศทางไปยัง standard output
Usage: chkarg2  argument ...      # error
$ cat oplog                       # output ในแฟ้ม oplog
check arguments version 2
```

คำถาม

---

1. การทดลองตาม (2) ผู้ใช้ไม่ได้กำหนด argument และไม่มีการเปลี่ยนทิศทาง เพราะเหตุใดจึงได้ error message ทาง standard output
2. และเช่นเดียวกัน เพราะเหตุใดการทดลองตาม (4) จึงได้ error message ทาง standard output

การดำเนินการกับ option

โปรแกรม out ต่อไปนี้ทำการรับชื่อแฟ้มที่ต้องการให้ดำเนินการแสดงผลจาก command line argument โดยหากผู้ใช้กำหนด option -v จะทำการเรียกใช้คำสั่ง more หากไม่กำหนดจะเรียกใช้คำสั่ง cat ดังต่อไปนี้

```
$ cat out
if  [ $# -eq 0 ]
    then
         echo "Usage: out -v filenmaes"  1>&2
         exit 1
fi

if  [ "$1"  !=  "-v"  ]
    then
         cat -- "$@"
    else
         shift
         more -- "$@"
fi
```

ข้อสังเกต

---

1. ตัวแปร $@ เก็บ command line argument ทั้งหมดไว้ เช่นหากเรียกใช้โปรแกรม out ดังนี้ คือ

   $ out -v a.dat b.dat

   ค่าของตัวแปร $@ มีค่าเป็น -v a.dat b.dat
2. เครื่องหมาย -- หลังคำสั่ง cat และ more ใช้กำหนดให้คำสั่งรู้ว่าจบส่วน option ของคำสั่งนั้นแล้ว ทั้งนี้ เพื่อป้องกันคำสั่งนำargument ที่อยู่ในตัวแปร $@ เช่น -v ไปกำหนดเป็น option ของตัวเอง ซึ่งจะทำให้ความหมายของโปรแกรมผิดไป
3. คำสั่ง shift ใช้สำหรับเลื่อนค่าของ argument ในตัวแปร $@ ไปทางซ้ายมือ 1 ตำแหน่ง มีผลให้ค่าแรกของ argument หายไป ซึ่งค่านี้ได้แก่ option -v ดังนั้นในตัวแปร $@ จึงเหลืออยู่เฉพาะชื่อแฟ้ม

โครงสร้าง if..then..elif..fi

```
if  test-command
    then
        commands
    elif test-command
    then
         commands
    else
         commands
fi
```

elif ที่เพิ่มขึ้นมาจากโครงสร้าง if..then..else เป็นการรวมโครงสร้าง else และ if..then เข้าด้วยกันเพื่ออำนวยความสะดวกในกรณีที่มีทางเลือกหลายทาง ซึ่งสามารถเขียนแทนได้ด้วยโครงสร้างเดิมดังนี้

```
if  test-command
    then
         commands
    else
        if  test-command
            then
                  commands
             else
                  commands
         fi
 fi
```

ตัวอย่างต่อไปนี้เป็นโปรแกรมที่ทำหน้าที่รับคำภาษาอังกฤษจากแป้นพิมพ์จำนวน 3 คำ จากนั้นทำการเปรียบเทียบคำทั้งสามเพื่อตรวจสอบว่ามีคำใดเหมือนกันบ้าง เนื่องจากมีจำนวนคำที่ต้องเปรียบเทียบ 3 คำ ดังนั้นจึงอาจมีกรณีที่เป็นไปได้ทั้งหมด 5 กรณีคือ

1. คำทั้งสามเป็นคำเดียวกัน
2. คำแรกเหมือนกับคำที่สอง
3. คำแรกเหมือนกับคำที่สาม
4. คำที่สองเหมือนกับคำที่สาม
5. คำทั้งสามแตกต่างกัน

เมื่อเขียนเป็นโปรแกรม (if3) แล้ว เป็นดังนี้

```
$ cat if3
echo -n "word 1: "
read word1
echo -n "word 2: "
read word2
echo -n "word 3: "
read word3



if [ "$word1" = "$word2" -a "$word2" = "$word3" ]
   then
       echo "Match: words 1, 2, & 3"
```

```
   elif [ "$word1" = "$word2" ]
   then
       echo "Match: words 1 & 2"
   elif [ "$word1" = "$word3" ]
   then
       echo "Match: words 1 & 3"
   elif [ "$word2" = "$word3" ]
   then
       echo "Match: words 2 & 3"
   else
       echo "No match"
fi
```

ข้อความที่ต้องการให้คำสั่ง echo พิมพ์ออก ต้องกำหนดไว้ในเครื่องหมายคำพูด เนื่องจากมีเครื่องหมาย & (แทนคำว่า and) เพื่อป้องกันไม่ให้ shell ตีความเป็นเครื่องหมายพิเศษ

การประยุกต์โครงสร้าง if ในโปรแกรมที่ใช้งานได้จริง

ความต้องการ: โปรแกรมสำหรับหา hard link ของแฟ้ม ใน directory ที่กำหนด ในกรณีที่ผู้ใช้ไม่ได้กำหนดชื่อ directory ให้ดำเนินการค้นหาใน directory ปัจจุบัน

รูปแบบของคำสั่ง: links <ชื่อแฟ้ม> [<ชื่อ directory>]

แนวความคิดทั่วไป:
แฟ้มที่มี hard link เมื่อใช้คำสั่ง ls -l จะมีการแสดงจำนวน link ใน column ที่ 2 โดยมีค่าตั้งแต่ 2 ขึ้นไป และการแสดงผลของ hard link จะมีรูปแบบเดียวกับแฟ้มจริงทำให้แยกความแตกต่างได้ยาก แต่เนื่องจากแฟ้มที่เป็น hard link ซึ่งกันและกันจะมี i-node number เป็นค่าเดียวกัน เมื่อได้ค่า i-node ของแฟ้มที่กำหนดได้แล้ว สามารถใช้คำสั่ง find ในการหาแฟ้มอื่นที่มีค่า i-node เดียวกันได้

การออกแบบโปรแกรม (ระดับที่ 0)

```
begin
  รับชื่อแฟ้ม และ directory (ถ้ามี) จาก command line argument
  ตรวจสอบความถูกต้องของชื่อแฟ้ม และชื่อ directory
      if ผู้ใช้ไม่กำหนดชื่อ directory then
    กำหนดให้เป็น directory ปัจจุบัน
      endif
  หาจำนวน link ของแฟ้มโดยใช้คำสั่ง ls -l
      if  จำนวน link = 1 then
    แฟ้มที่กำหนดไม่มี link
      else
    หาค่า i-node ของแฟ้มที่กำหนด
    ใช้คำสั่ง find หาแฟ้มอื่นๆที่มี i-node ตรงกัน
```

```
      endif
end
```

การออกแบบโปรแกรม (ระดับที่ 1)

```
---------------------------
begin
      { ตรวจสอบความถูกต้องของจำนวน argument }
      if  จำนวน argument ไม่อยู่ในช่วงที่เหมาะสม (1 หรือ 2) then
    แสดงรูปแบบการใช้งานของคำสั่ง links
    เลิกการทำงาน
      endif

      { ถ้าทำงานมาถึงจุดนี้แสดงว่า argument ที่ผู้ใช้ป้อนเข้ามามีจำนวนอย่างน้อย 1 ตัว }
      { ซึ่งอาจเป็นชื่อแฟ้ม หรือชื่อ directory ก็ได้ จึงต้องทำการทดสอบต่อไป }
      if  argument ตัวแรกเป็น directory และมีอยู่จริง then
          { ผู้ใช้กำหนด argument ตัวแรกผิด เนื่องจาก argument ตัวแรกต้องเป็นชื่อแฟ้มเสมอ }
    แสดงรูปแบบการใช้งานของคำสั่ง links
    เลิกการทำงาน
      else
           { สันนิษฐานในเบื้องต้นว่าargument ตัวแรกเป็นชื่อแฟ้ม }
    เก็บ argument ตัวแรกไว้เป็นชื่อแฟ้ม
      endif

      { ที่จุดนี้ทำการจัดเก็บ argument ตัวแรกเป็นชื่อแฟ้ม แต่เนื่องจากชื่อ directory ซึ่งเป็น }
      { argument ตัวที่ 2 อาจมีหรือไม่ก็ได้ ในกรณีที่ไม่มี กำหนดให้เป็น directory ปัจจุบัน }
      if  มี argument เพียงตัวเดียว then
    เก็บ directory ปัจจุบันเป็นชื่อ directory
      elif  argument ตัวที่สองเป็นชื่อ directory ที่มีอยู่จริง then
    เก็บ argument ตัวที่สองไว้เป็นชื่อ directory
      else
    แสดงรูปแบบการใช้งานของคำสั่ง links
    เลิกการทำงาน
      endif

      { ถ้าทำงานมาถึงจุดนี้แสดงว่าผู้ใช้กำหนด argument ได้ถูกต้องตามรูปแบบที่กำหนด }
      { แต่ยังไม่ได้ทดสอบชื่อแฟ้มว่าถูกต้องหรือไม่ จึงต้องดำเนินการ }
      if  not (ชื่อแฟ้มเป็นชื่อที่ถูกต้องและเป็นแฟ้มที่มีอยู่จริง) then
    แจ้งให้ผู้ใช้ทราบว่าชื่อแฟ้มที่ผู้ใช้ป้อนเข้ามาไม่มีอยู่จริง
    เลิกการทำงาน
      endif
```

```
      { หาจำนวน link ของแฟ้มที่กำหนด โดยเรียกใช้คำสั่ง ls }
  เรียกใช้คำสั่ง ls -l <ชื่อแฟ้ม> และนำผลลัพธ์ที่ได้กำหนดให้ตัวแปรระบบ
  เก็บค่าจำนวน link ซึ่งอยู่ในตัวแปรลำดับที่สองไว้สำหรับดำเนินการต่อไป

      { ในกรณีที่จำนวน link ที่อ่านได้มีค่าเป็น 1 แสดงว่าแฟ้มนั้นไม่มี link แสดงข้อความ }
      { แจ้งให้ผู้ใช้ทราบและเลิกการทำงานตามปกติ }
      if  จำนวน link = 1  then
    แจ้งให้ผู้ใช้ทราบว่าแฟ้มที่ผู้ใช้กำหนดไม่มี link
    เลิกการทำงานตามปกติ
      endif

      { หา i-node ของแฟ้มที่กำหนดและหาแฟ้มอื่นที่มี inode ตรงกันโดยเรียกใช้คำสั่ง find }
  หาค่า i-node number ของแฟ้มที่กำหนด โดยเรียกใช้คำสั่ง ls -i <ชื่อแฟ้ม>
  นำค่า i-node number ซึ่งอยู่ในตัวแปรลำดับที่หนึ่งกำหนดให้แก่ตัวแปร inode
  ค้นหาแฟ้มอื่นที่มี i-node number ค่าเดียวกับแฟ้มที่กำหนดและแสดงผลโดยใช้คำสั่ง find -inum
end
```

ตัวแปรที่ต้องใช้

```
------------
     $file        ตัวแปรเก็บชื่อแฟ้มที่ต้องการตรวจสอบ link
     $directory   directory ที่ต้องการค้นหาแฟ้ม
     $linkcnt     จำนวน link ของแฟ้มที่กำหนด
     $inode หมายเลข inode ของแฟ้มที่กำหนด
```

Utilities ของระบบปฏิบัติการ Unix ที่ต้องใช้

```
-------------------------------------
      ls -l <ชื่อแฟ้ม>            แสดงค่าจำนวน link ของแฟ้มที่กำหนด
      ls -i <ชื่อแฟ้ม>            แสดงค่า i-node no. ของแฟ้มที่กำหนด
      find <ชื่อ directory> -inum <หมายเลข i-node> -print
```

ค้นหาและแสดงรายชื่อแฟ้มที่มี inode ที่กำหนดโดยเริ่มจาก directory ที่กำหนด และทำการแสดงผล

```
#!/bin/bash
# Identify links to a file
# Usage: lnks  file  [directory]

if [ $# -eq 0 -o $# -gt 2 ]; then
   echo "Usage: lnks  file  [directory]" 1>&2
   exit 1
fi

if [ -d "$1" ]; then
   echo "First argument cannot be a directory." 1>&2
```

```
   echo "Usage: lnks  file  [directory]" 1>&2
   exit 1
else
   file="$1"
fi

if [ $# -eq 1 ]; then
   directory="."
   elif [ -d "$2" ]; then
        directory="$2"
   else
        echo "Optional second argument must be a directoty." 1>&2
        echo "Usage: lnks  file  [directory]" 1>&2
        exit 1
fi

# Check that file exists and is an ordinary file
if [ ! -f "$file" ]; then
   echo "lnks: $file not found or special file" 1>&2
   exit 1
fi

# Check link count on file
set -- $(ls -l "$file")

linkcnt=$2

if [ "$linkcnt" -eq 1 ]; then
   echo "lnks: no other hard link to $file" 1>&2
   exit 0
fi

# Get the i-node of the given file
set -- $(ls -i "$file")
inode=$1

# Find and print the files with that i-node number
echo "lnks: using find to search for links ... " 1>&2
find "$directory" -xdev -inum "$inode" -print
```

การทดสอบการทำงานของโปรแกรม Links

---

1. สร้างและเข้าสู่ directory tmp
2. เรียกใช้โปรแกรม vi เพื่อสร้างแฟ้มชื่อ aa.txt โดยมีข้อความดังนี้

   ```
   a quick brown fox jumps over a lazy dog.
   ```

3. จัดเก็บแฟ้ม และตรวจสอบรายละเอียดของแฟ้มโดยใช้คำสั่ง ls -l

   ```
   -rw-r--r--    1 jira     staff          41 Jul  1 15:18 aa.txt
   ```

ข้อมูลที่น่าสนใจอยู่ใน column ที่ 2 ข้อมูลดังกล่าวนี้คือจำนวน link ของแฟ้ม ซึ่งในตัวอย่างนี้แสดงว่าแฟ้ม aa.txt มีจำนวน link เป็น 1 หรือเป็นแฟ้มที่ยังไม่มีการสร้าง link

4. ทำการสร้าง hard link ชื่อ bb.txt ไปยังแฟ้ม aa.txt โดยใช้คำสั่ง ln ดังนี้

   ```
   $ ln  aa.txt  bb.txt
   ```

5. ทำการตรวจสอบรายชื่อแฟ้มโดยใช้คำสั่ง ls -l

```
-rw-r--r--    2 jira     staff          41 Jul  1 15:18 aa.txt
-rw-r--r--    2 jira     staff          41 Jul  1 15:18 bb.txt
```

จะเห็นว่ามีการสร้าง bb.txt ขึ้นเป็นแฟ้มที่มีคุณลักษณะเหมือนกับแฟ้ม aa.txt และแฟ้มทั้งสองมีจำนวน link เป็น 2

6. ใช้โปรแกรม vi แก้ไขข้อมูลในแฟ้ม bb.txt และตรวจสอบการดำเนินการโดยเรียกดูข้อมูลจากแฟ้ม aa.txt จะเห็นว่าข้อมูลในแฟ้มทั้งสองเป็นชุดเดียวกัน

7. นอกจากแฟ้มทั้งสองจะมีข้อมูลเหมือนกันแล้ว ยังมี i-node number (ดัชนีของแฟ้มในระบบปฏิบัติการ Unix) ตรงกัน ค่า inode number นี้สามารถตรวจสอบได้โดยใช้คำสั่ง ls -il ดังนี้

```
11862143 -rw-r--r--    2 jira     staff          41 Jul  1 15:18
aa.txt
11862143 -rw-r--r--    2 jira     staff          41 Jul  1 15:18
bb.txt
```

8. ทดลองใช้คำสั่ง touch สร้างแฟ้มว่างอีกสองสามแฟ้ม และตรวจสอบ inode number

9. ทดลองเรียก shell script ดังนี้

```
$ links  aa.txt
```

ผลที่ได้เป็นอย่างไร เข้าใจการทำงานของโปรแกรมหรือไม่

หมายเหตุ

คำสั่ง set -- $(ls -l "$file")

คำสั่ง set เมื่อนำมาใช้ร่วมกับ $() เป็นการกำหนดให้สร้างเชลล์ย่อยและนำผลลัพธ์จากเชลล์ย่อยมากำหนดให้กับ positional parameter (ตัวแปร $1 … $9) สำหรับเครื่องหมาย -- หลังคำสั่ง set ใช้แสดงให้ทราบว่าคำสั่ง set ไม่มี option

สามารถทดสอบการทำงานได้ที่ prompt ของ shell ดังนี้

```
$ set -- $(ls -l aa.txt)
```

จากผลลัพธ์ของคำสั่ง ls จะมีการจัดเก็บในตัวแปรตามตำแหน่งดังนี้

```
-rw-r--r--    1     jira     staff          41   Jul   1   15:18
aa.txt
$1             $2    $3       $4             $5   $6  $7   $8
$9
```

จะเห็นได้ว่าจำนวน link ของแฟ้ม อยู่ในตัวแปรลำดับที่สอง ($2) สามารถทดสอบค่าของตัวแปรเหล่านี้ได้โดยใช้คำสั่ง echo ดังนี้

```
$ echo $1
$ echo $2
```

เมื่อทราบว่าจำนวน link อยู่ในตัวแปรลำดับที่ 2 ($2) แล้ว สามารถนำค่าในตัวแปรนี้มาทดสอบได้ หากค่าเป็น 1 แสดงว่าไม่มี link อื่น หากค่ามากกว่า 1 แสดงว่ามี hard link

ในทำนองเดียวกัน คำสั่ง

```
set -- $(ls -i "$file")
```

เป็นการหา i-node no ของแฟ้ม ซึ่งแฟ้มที่เป็น hard link ซึ่งกันและกัน มีหมายเลข i-node เดียวกัน โดย option -i ใช้ในการหาหมายเลข inode ของแฟ้มที่ระบุ เช่น

```
$ ls -i aa.txt
11862143   aa.txt
```

11862143 เป็นหมายเลข inode ของแฟ้ม aa.txt ซึ่งหมายเลขนี้เป็นข้อมูลชุดแรก เมื่อใช้คำสั่ง

```
$ set -- `ls -i aa.txt`
```

หมายเลข inode จะอยู่ในตัวแปรลำดับที่ 1 ซึ่งสามารถนำไปใช้ร่วมกับคำสั่ง find

เพื่อหาแฟ้มอื่นที่มี inode ตรงกันได้ คำสั่ง

```
$ find <directory>  -inum <inode_no> -print
```

เป็นการกำหนดให้มีการค้นหาแฟ้มที่มี inode เป็น <inode_no> ตามที่กำหนด (-inum) โดยเริ่มค้นจาก <directory> ที่กำหนดและเมื่อพบแฟ้มแล้ว ให้แสดงชื่อแฟ้มนั้น (-print)

option -xdev ของคำสั่ง find มีไว้เพื่อกำหนดให้ทำการค้นหาแฟ้มเฉพาะในระบบแฟ้มเดียวกัน ไม่มีการข้ามไปค้นหาแฟ้มในระบบแฟ้มอื่น

โครงสร้างทำซ้ำชนิด for

โครงสร้างทำซ้ำชนิด for มีรูปแบบการใช้งาน 2 แบบ

แบบที่ 1 for...in

```
for <ตัวแปรดัชนี>  in  <รายการค่าที่ต้องดำเนินการ>
do
  รายการคำสั่ง
done
```

ตัวอย่าง กำหนดรายการที่ต้องการดำเนินการ เป็น รายชื่อผลไม้

รายการ =(apples, oranges, pears, bananas)

ตัวแปรดัชนี - fruit

```
$ cat  fruit
```

```
for fruit in apples oranges pears bananas
do
     echo "$fruit"
done
echo "Task complete."

$ fruit
apples            # รอบแรก  ตัวแปร fruit = apples
oranges           # รอบที่สอง ตัวแปร fruit = oranges
pears             # ...
bananas
```

คำอธิบาย

```
--------
       รายการที่ต้องดำเนินการ
                      |
for  fruit  in  apples  oranges  pears  bananas
      |
  ตัวแปรดัชนี

do
  คำสั่งการประมวลผลใน loop - การแสดงค่าของตัวแปร fruit
done
```

คำถาม

```
------
```

หากทำการเปลี่ยนคำสั่งแสดงผลใน `loop` เป็น `echo "fruit"` โปรแกรมทำงานเหมือนเดิมหรือไม่? เพราะเหตุใด?

เนื่องจากโครงสร้าง `for` เป็นส่วนหนึ่งของเชลล์จึงสามารถใช้กับอักขระพิเศษเช่น * (หรือ `ambiguous file reference`) ซึ่งใช้แทนแฟ้มทุกแฟ้ม (ยกเว้นแฟ้มซ่อน) ในไดเรกทอรีปัจจุบัน ตัวอย่างต่อไปนี้เป็นการแสดงรายชื่อไดเรกทอรีย่อยทั้งหมดที่มีอยู่ในไดเรกทอรีปัจจุบัน

```
$ cat dirfiles
for i in *
do
     if [ -d "$i" ]; then
          echo "$i"
     fi
done
```

ก่อนจะทำงานใน `loop` เชลล์จะขยาย * เป็นรายการของชื่อแฟ้ม และนำมากำหนดให้ตัวแปร `i` ครั้งละชื่อ จนกว่าครบทุกชื่อ การดำเนินกับชื่อแฟ้มในรายการคือตรวจสอบ และแสดงเฉพาะชื่อที่เป็นไดเรกทอรี

ตัวอย่าง for รูปแบบที่ 1

```
#!/bin/bash

for i in 1 2 3 4 5
do
    echo $i
done
echo ========================

for i in {1..5}
do
    echo $i
done
echo ========================

for i in {1..10..2}
do
    echo $i
done
echo ========================

for i in $(seq 1 2 10)
do
    echo $i
done
echo ========================

for ((i=1; i<=5; i++))
do
    echo $i
done
```

โครงสร้างทำซ้ำชนิด for รูปแบบที่ 2

รูปแบบนี้ทำงานกับ command line argument (ไม่มีคำสำคัญ in)

```
for  loop-index              # in command line argument หรือตัวแปร $@
do
     commands
done
```

ตัวอย่าง

```
$ cat for_test
for  arg
do
     echo "$arg"
done
```

```
$ for_test  notebook netbook ultrabook tablet
notebook
netbook
ultrabook
tablet
```

สรุป คำสั่ง `for`

- รูปแบบแรก ผู้ใช้กำหนด "รายการค่าที่ต้องการดำเนินการ" หลังคำสำคัญ `in`
- รูปแบบที่สอง ผู้ใช้กำหนด "รายการค่าที่ต้องการดำเนินการ" ผ่านทาง `command line argument` หรือรายการที่ต้องการดำเนินการอยู่ในตัวแปร `$@` -- การดำเนินการในแต่ละรอบ ตัวแปรดัชนี `loop` จะมีค่าตาม `$1, $2, ...` จนกว่าจะหมดรายการ

Workshop1

- ให้เขียน Shell Script วน Loop แสดง Menu การคำนวณ ด้านล่าง

```
Select the operation
1: Add
2: Subtract
3: Multiply
4. Divide
5. Exit
```

- หลังจากผู้ใช้กด 5 ให้โปรแกรมแสดง "Bye Bye. Thanks for using me." แล้ว exit
- หากผู้ใช้กด 1-4 ให้โปรแกรมรับค่าสองตัว และแสดงผลลัพธ์ ดังตัวอย่างด้านล่าง (เป็นตัวอย่างการแสดงข้อความ และ รับค่า หลังจากที่ผู้ใช้กด 1)

```
Input the first operand: 5  (ผู้ใช้ป้อนข้อมูล 5)
Input the second operand: 6 (ผู้ใช้ป้อนข้อมูล 6)
Add result is: 11
```

Workshop2

ให้เขียน Shell Script ชื่อ findEven ที่ช่วยคำนวณจำนวนเลขคู่จากเลขทั้งหมดที่ผู้ใช้ป้อนข้อมูลเข้า ดังตัวอย่าง

```
$ ./findEven
"Usage: findEven  num  [num ...]"      # แจ้งวิธีการใช้งาน

$ ./findEven 1 2 3 4 5 6 7 8 9 10
The number of even numbers is : 5      # แสดงจำนวนเลขคู่
```