Lecture 13

โครงสร้างควบคุมการทำงานแบบมีทางเลือกหลายทาง

โครงสร้างควบคุมการทำงานแบบมีทางเลือกหลายทาง (case) มีรูปแบบการใช้งานทั่วไปดังนี้

```
case   ตัวแปรเก็บสายอักขระที่ต้องการทดสอบ in
       สายอักขระแบบที่ 1)
              คำสั่งสำหรับสายอักขระแบบที่ 1
              ;;
       สายอักขระแบบที่ 2)
              คำสั่งสำหรับสายอักขระแบบที่ 2
              ;;
       ... ... .. )
              … … …
              ;;
esac
```

หมายเหตุ

(1) เครื่องหมาย ;; ที่ใช้ปิดท้ายของแต่ละกรณี เป็นไวยากรณ์บังคับของโครงสร้างไม่สามารถตัดออกเพื่อให้ผลการทำงานเป็นแบบผ่านไปสู่กรณีถัดไปหรือที่นิยมเรียกว่า fall through เช่นในโครงสร้าง switch..case ในภาษา C ได้

(2) โปรดระลึกไว้เสมอว่า โครงสร้าง case เป็นเพียงกรณีพิเศษของโครงสร้าง if เท่านั้น มีไว้เพื่ออำนวยความสะดวกในการเขียนโปรแกรม และทำให้ "เนื้อหา" ของโปรแกรมชัดเจนขึ้น สามารถเขียนแทนด้วยโครงสร้าง if เสมอ

ตัวอย่างที่ 1

โปรแกรมตัวอย่างต่อไปนี้ ทำหน้าที่ตรวจสอบอักขระที่ผู้ใช้ป้อนเข้าทางแป้นพิมพ์และแสดงผลอักขระนั้น เช่นต้องการตรวจสอบเฉพาะ A, B หรือ C กรณีที่เป็นอักษรอื่นจะแจ้งให้ผู้ใช้ทราบว่าไม่ใช้ตัวอักษรในกลุ่ม

```
$ cat log
#!/bin/bash
echo -n "Enter A, B, or C : "
read letter

case "$letter" in
     A)
          echo "You entered A"
          ;;
     B)
          echo "You entered B"
          ;;
     C)
          echo "You entered C"
          ;;
     *)
          echo "You did not enter A, B, or C"
          ;;
esac
```

สายอักขระที่ใช้ในโครงสร้าง case นอกจากเป็นข้อความธรรมดาแล้ว ยังสามารถใช้อักขระพิเศษของเชลล์ได้ดังนี้

`*` ใช้แทนสายอักขระใด ๆ ที่มีความยาวตั้งแต่ 0 ตัวขึ้นไป โดยปกตินิยมใช้ในการกำหนดกรณีที่ต้องการให้เป็น default ในโครงสร้าง

`?` ใช้แทนอักขระใด ๆ จำนวนหนึ่งตัว

`|` ใช้คั่นระหว่างสายอักขระที่ต้องการใช้เป็นตัวเลือก ทำหน้าที่เป็น Logical OR

`[…]` ใช้กำหนดกลุ่มของอักขระ (character class) ในข้อความซึ่งตรงกับอักขระตัวใดตัวหนึ่งในสายอักขระที่กำหนด หากต้องการดัดแปลงให้โปรแกรม case1 สามารถรับอักษร A ได้ทั้งอักษรตัวใหญ่และตัวเล็ก ดำเนินการได้โดยการดัดแปลงโปรแกรมโดยใช้ | หรือ [...] ดังนี้

```
A|a)                        # หรือ[Aa]
     echo "You entered A"
     ;;
```

กรณีของอักษรอื่นก็ดำเนินการเช่นเดียวกัน

โครงสร้าง case เป็นโครงสร้างที่เหมาะสมในการสร้างรายการเลือก หรือ Menu อย่างง่าย ดังโปรแกรมตัวอย่างต่อไปนี้

ตัวอย่างที่ 2

```
---------
     $ cat menu.prn
     #!/bin/bash

     echo -e "\n            COMMAND MENU\n"
     echo    "     a.  Current date and time"
     echo    "     b.  Users currently logged in"
     echo    "     c.  Name of the working directory"
     echo -e "     d.  Contents of the working directory\n"
     echo -n "     Enter a, b, c, or d : "

     read answer
     echo

     case "$answer" in
          a)
               date
               ;;
          b)
               who
               ;;
          c)
               pwd
               ;;
          d)
               ls
               ;;
          *)
               echo "There is no selection: $answer"
               ;;
     esac
```

เมื่อให้ script ทำงานปรากฏผลดังนี้

```
$ menu

            COMMAND MENU

      a.  Current date and time
      b.  Users currently logged in
      c.  Name of the working directory
      d.  Contents of the working directory

      Enter a, b, c, or d : _
```

ของใหม่ที่นำมาใช้ในโปรแกรมนี้คือ echo -e, ซึ่ง option -e ของคำสั่ง echo ใช้สำหรับกำหนดการใช้งาน escape character รูปแบบเดียวกับการใช้งานในภาษา C ได้แก่

```
\b - backspace          \f - formfeed
\n - newline            \r - carriage return
\t - horizontal tab     \v - vertical tab
\\ - backslash          \nnn - เมื่อ nnn เป็นรหัส ascii ฐานแปด
```

โปรแกรมนี้เป็นเพียงการสาธิตการใช้โครงสร้าง case สำหรับใช้กับเมนูเท่านั้น คำสั่งที่ใช้จึงเป็นเพียง Utilities พื้นฐานเท่านั้น สำหรับโปรแกรมใช้งานจริงสามารถแทนคำสั่งด้วยโปรแกรมย่อย หรือ script ที่เขียนขึ้น ซึ่งจะช่วยให้ shell script สามารถทำงานที่ซับซ้อนได้อีกมากมาย

แบบฝึกหัด

จงเปรียบเทียบโครงสร้าง case ของ shell script และโครงสร้าง switch..case ในภาษา C ต่อไปนี้ในแบบ Compare and Contrast

```
shell script                                ภาษา C
------------                                ------
case  ตัวแปรเก็บสายอักขระ  in               switch ( นิพจน์ที่ให้ค่าเป็นจำนวนเต็ม ) {
      สายอักขระแบบที่ 1)                   case ค่าแรก:
            คำสั่งสำหรับสายอักขระแบบที่ 1          คำสั่งสำหรับค่าแรก;
            ;;                                     break;
      สายอักขระแบบที่ 2)                   case ค่าที่สอง:
            คำสั่งสำหรับสายอักขระแบบที่ 2          คำสั่งสำหรับค่าที่ 2;
            ;;                                     break;
      สายอักขระแบบที่3)                    case ค่าที่สาม:
            คำสั่งสำหรับสายอักขระแบบที่ 3          คำสั่งสำหรับค่าที่ 3;
            ;;                                     break;
      ... ... ...                           ... ... ...
            ... ... ...                            ... ... ...
      *)                                    default:
            คำสั่งสำหรับสายอักขระอื่นๆ             คำสั่งสำหรับค่าอื่นๆ;
 esac                                       }
```

"Compare" คือการเปรียบเทียบของสองสิ่งว่ามีความเหมือนหรือความคล้ายคลึงกันอย่างไร ส่วน "Contrast" คือการเปรียบเทียบว่าของสองสิ่งนั้นมีความแตกต่างกันอย่างไร ดังนั้น "Compare and Contrast" จึงเป็นการเปรียบเทียบของสองสิ่งให้เห็นความคล้ายและความแตกต่าง ก่อนจะลงมือเขียนคำตอบต้องรวบรวมและจัดความคิดให้ดี และควรทำการเปรียบเทียบไปทีละคุณลักษณะ

ตัวอย่างการประยุกต์ใช้งานโครงสร้าง case

โปรแกรม safedit ซึ่งเป็น script ในการเรียกใช้โปรแกรม vim (vi improved) โดยจะทำการสร้างแฟ้มสำรองให้โดยอัตโนมัติ เมื่อมีการนำแฟ้มที่มีอยู่มาแก้ไข และในกรณีที่ไม่สามารถดำเนินการได้สำเร็จ เช่น สายสื่อสารหรือเครือข่ายไม่ทำงาน หรือ โพรเซสถูกทำลายไป จะรักษาแฟ้มเดิมก่อนการแก้ไขไว้

```
$ cat  safedit
#!/bin/bash

PATH=/bin:/usr/bin
script=$(basename $0)                  # basename

case $# in                        # case ของจำนวน command line argument

      # ไม่มี argument เลย แสดงว่าผู้ใช้ต้องการพิมพ์โดยยังไม่ได้ตัดสินใจตั้งชื่อแฟ้ม
      0)
            vim
            exit 0
            ;;

      # มี argument จำนวน 1 ตัว สันนิษฐานว่าเป็นชื่อแฟ้มที่ผู้ใช้ต้องการดำเนินการ
      1)
```

```
            # เป็นชื่อแฟ้มใหม่
            if [ ! -f "$1" ]
                 then
                      vim "$1"
                      exit 0
            fi

            # แฟ้มที่มีอยู่แล้ว ตรวจสอบสิทธิในการ read และ write
            if [ ! -r "$1" -o ! -w "$1" ]
                 then
                      echo "$script: check permissions on $1" 1>&2
                      exit 1
                 else
                      editfile=$1
            fi

            # เนื่องจากต้องมีการสร้างแฟ้มสำรองในระหว่างการทำงาน จึงต้องตรวจสอบว่า
            # มีสิทธิ์สร้างแฟ้มใหม่ในไดเรกทอรีปัจจุบันหรือไม่
            if [ ! -w "." ]
                 then
                      echo "$script: backup cannot be " \
                              "created in the working directory" 1>&2
                      exit 1
            fi

            ;;

      # มี argument มากกว่า 1 ตัว
      *)
          vi echo "Usage: $script [file-to-edit]" 1>&2
           exit 1
           ;;
esac

# กำหนดชื่อแฟ้มสำรองเป็น 'หมายเลขโพรเซส.script' แฟ้มนี้สร้าง /tmp (temporary - ชั่วคราว)
# ซึ่งเป็นไดเรกทอรีสำหรับสร้างแฟ้มชั่วคราว ผู้ใช้ทุกคนในระบบมีสิทธิในการใช้งาน เมื่อกำหนดชื่อแล้ว
# จึงคัดลอกแฟ้มที่ต้องการแก้ไขไปสำรองไว้

tempfile=/tmp/$$.script     # $$ คือตัวแปรเก็บ process id (PID) ของโพรเซสปัจจุบัน
cp $editfile $tempfile

# เรียก vim สำหรับผู้ใช้แก้ไขแฟ้มที่ระบุ -- vim เมื่อ vim ทำงานเสร็จจะ return exit status
# หากเป็น 0 จะทำให้เงื่อนไขของ if เป็นจริง หากเป็นค่าอื่นจะเป็นเท็จ
if vim $editfile
     then
          # แจ้งให้ผู้ใช้ทราบว่ามีการสำหรองข้อมูลของแฟ้มเดิมไว้
          mv $tempfile bak.$(basename $editfile)
          echo "$script: backup file created"
```

```
        else
                # เปลี่ยนชื่อแฟ้มที่สำรองข้อมูลไว้เป็น editerr และแจ้งผู้ใช้ทราบ
                mv $tempfile editerr
                echo "$script: edit error -- copy of " \
    fi
```

*** หมายเหตุ: หาก comment ในโปรแกรมไม่เพียงพอ มีคำอธิบายเพิ่มเติมดังนี้ ***

ตัวแปรที่ใช้

$0 - เส้นทางสัมบูรณ์ (absolute path) ของชื่อ shell script - ตัดชื่อเส้นทางออกเหลือเฉพาะชื่อเก็บในตัวแปร script

$1 - command line argument ตัวแรก สันนิษฐานว่าเป็นชื่อแฟ้มที่ต้องการแก้ไข

script - ตัวแปรเก็บชื่อ shell script

editfile - ชื่อแฟ้มที่ต้องการแก้ไข

tempfile - ชื่อแฟ้มชั่วคราว - bak. ชื่อแฟ้มที่ต้องการแก้ไข

เหตุผลที่เป็น bak. ชื่อแฟ้มเดิม แทนที่จะเป็น ชื่อแฟ้มเดิม.bak เนื่องจาก unix บางระบบจำกัดความยาวของชื่อแฟ้มไว้เพียง 14 ตัวอักษร

อธิบายคำสั่ง

```
PATH=/bin:/usr/bin
```

กำหนดเส้นทางการค้นแฟ้มในตัวแปร PATH ในกรณีที่ผู้ใช้อาจไม่ได้กำหนดไว้ ซึ่งจะทำให้เกิด error ในแบบ command not found

ค่านี้เป็นของระบบปฏิบัติการ Linux สำหรับในระบบปฏิบัติการอื่น อาจต้องกำหนด directory เพิ่มเติม เช่น /usr/ucb และ /usr/5bin สำหรับระบบปฏิบัติการ Sun Solaris ของบริษัท Sun Microsystem เป็นต้น

```
script=$(basename $0)
```

นำผลลัพธ์ของคำสั่ง basename ซึ่งใช้ในการตัดเส้นทางออกจากชื่อแฟ้ม ตัดชื่อเส้นทางออกจากชื่อโปรแกรม เนื่องจากต้องการนำชื่อของโปรแกรมไปเป็นส่วนหนึ่งในชื่อแฟ้มสำรอง และการที่ต้องเรียกใช้คำสั่ง basename กับชื่อโปรแกรม ไม่สามารถกำหนดชื่อโปรแกรมตายตัวได้ เนื่องจากผู้ใช้อาจเปลี่ยนชื่อโปรแกรมได้

```
if [ ! -f "$1" ]
```

ทดสอบว่าชื่อแฟ้มที่ผู้ใช้ระบุเป็นแฟ้มใหม่ซึ่งยังไม่มีในไดเรกทอรีปัจจุบัน จะได้สร้างแฟ้มใหม่โดยไม่ต้องดำเนินการเรื่องการสำรองข้อมูล ในทางปฏิบัติเป็นการทดสอบว่าค่าในตัวแปร $1 เป็นแฟ้มธรรมดา (regular file) และมีอยู่จริงหรือไม่ ในกรณีที่ไม่มีอยู่ในระบบคำสั่ง test จะคืน exit status เป็น "เท็จ" จึงต้องนิเสธโดยใช้ ! (not)

```
if [ ! -r "$1" -o ! -w "$1" ]
```

ทดสอบว่าผู้ใช้มีสิทธิในการอ่าน read/write หรือไม่ หาก "ไม่มีสิทธิในการอ่าน" หรือ "ไม่มีสิทธิในการเขียน" ให้แจ้งให้ผู้ใช้เปลี่ยนแปลงสิทธิให้เหมาะสมก่อนเรียกใช้งานใหม่

"ไม่มีสิทธิในการอ่าน" คือ !-r "$1", "ไม่มีสิทธิในการเขียน" คือ !-w "$1" เชื่อมด้วย OR คือ -o

```
if [ ! -w "." ]
```

ทดสอบว่าผู้ใช้มีสิทธิในการสร้างแฟ้มในไดเรกทอรีปัจจุบัน (".") หรือไม่ หรือมีสิทธิ write ในไดเรกทอรีปัจจุบันหรือไม่

```
tempfile=/tmp/$$.$script
```

กำหนดชื่อแฟ้มชั่วคราวเป็นหมายเลขโพรเซส (ตัวแปร $$) ตามด้วยชื่อของโปรแกรม ในไดเรกทอรี /tmp ซึ่งผู้ใช้ทุกคนมีสิทธิสร้างแฟ้ม การกำหนดชื่อแฟ้มแบบนี้จะจำเป็นมาก เนื่องจากผู้ใช้แต่ละคนอาจตั้งชื่อแฟ้มซ้ำกันและอาจบังเอิญแก้ไขในเวลาเดียวกันได้

```
drwxrwxrwt   5 root root  4096 Jan 31 16:42 tmp
```

t - สำหรับไดเรกทอรี หมายความว่าถึงแม้ผู้ใช้จะมีสิทธิในการ read/write ในไดเรกทอรี แต่ระบบจะอนุญาตให้อ่าน, แก้ไข, และลบแฟ้มได้เฉพาะเจ้าของแฟ้มเท่านั้น

```
if vi $editfile
```

การใช้งานโครงสร้าง if ที่กำหนดให้มีการเรียกใช้ vim แก้ไขแฟ้มที่กำหนดในตัวแปร editfile เมื่อ vim ทำงานเสร็จจะคืนค่าสถานะการทำงาน (exit status หรือ return status) ผ่านตัวแปร $? หากทำงานได้เสร็จสมบูรณ์ จะมีค่าเป็น 0 ทำให้เงื่อนไขของ if เป็นจริง หากมีค่าไม่เป็น 0 เงื่อนไขของ if จะเป็นเท็จ

แบบฝึกหัด

---

(1) script มีเงื่อนไขการทำงานหลายแบบ จงออกแบบการทดลองเพื่อให้สามารถทดสอบการทำงานได้ทุกกรณี และเขียนรายงานการทดสอบ ซึ่งประกอบด้วย

- โครงสร้างและไดเรกทอรีของแฟ้มที่จำเป็นต้องใช้ในการทดสอบ
- คำสั่งที่ใช้ในการทดสอบ พร้อมผลการทดสอบ (ผ่าน/ไม่ผ่าน)

แบบฝึกหัดนี้เป็นพื้นฐานที่สำคัญสำหรับบทที่ 4 การทดลองและการอภิปรายผล สำหรับโครงงานในระดับปริญญาตรี

(2) โปรแกรมนี้มี "จุดด้อย" ที่สำคัญอย่างไร? และควรต้องแก้ไขอย่างไร?

Here Document

---

here document (เอกสารอยู่ที่นี่) เป็นกลุ่มของข้อความหรือคำสั่งที่มีจุดมุ่งหมายพิเศษ ที่ใช้รูปแบบของการเปลี่ยนทิศทาง input/output เพื่อป้อนข้อมูลหรือคำสั่งให้แก่โปรแกรมหรือคำสั่ง เช่น ftp, cat และโปรแกรมในกลุ่ม editor เป็นต้น

```
คำสั่ง <<ข้อมูลหรือคำสั่ง          COMMAND <<InputComesFromHere
...                                ...
...                                ...
ข้อมูลหรือคำสั่ง                   InputComesFromHere
```

สัญลักษณ์หรือข้อความที่ใช้กำหนด 'ขอบเขต' ของกลุ่มข้อมูลหรือคำสั่งเรียกว่า limit string ซึ่งต้องใช้เหมือนกันสำหรับกำหนดจุดเริ่มต้นและจุดสิ้นสุด limit string ที่ใช้เป็นจุดเริ่มต้นอยู่หลัง << ซึ่งใช้แทนการ redirect ข้อมูลหรือคำสั่งในกลุ่มเข้าทาง standard input ของโปรแกรม - limit string เลือกจากสัญลักษณ์หรือข้อความที่แน่ใจว่าไม่มีในกลุ่มข้อมูลและคำสั่งที่ต้องการใช้งาน กล่าวอีกนัยหนึ่งคือ here document เป็นกลไกที่ช่วยให้ผู้เขียนโปรแกรมสามารถทำการเปลี่ยนทิศทางข้อมูลเข้าจากแฟ้มข้อมูลภายนอก เป็นข้อความที่กำหนดไว้ภายในโปรแกรม ขอให้ศึกษาจากโปรแกรม birthday ซึ่งใช้สำหรับหาวันเกิดของบุคคลที่มีรายชื่อกำหนดไว้ในแฟ้ม ดังต่อไปนี้

```
$ cat birthday
grep -i "$1" <<+
Alex        June 22
Babara      February 3
Darlene     May 8
Helen       March 13
Jenny       January 23
Nancy       June 26
+
```

เครื่องหมาย << แสดงเปลี่ยนทิศทางข้อมูลเข้าของ grep เป็นข้อมูลที่อยู่หลัง << ซึ่งเริ่มต้นด้วย limit string และขึ้นบรรทัดใหม่ ข้อมูลจริงจะอยู่ในบรรทัดต่อไป เมื่อหมดข้อมูลแล้ว บรรทัดสุดท้ายเป็น limit string ชุดเดียวกัน ซึ่งต้องเริ่มต้นที่ตำแหน่งแรกของบรรทัด

โปรแกรม birthday ตามตัวอย่างนี้เลือกใช้เครื่องหมาย + เพียงตัวเดียวเป็น limit string สำหรับกำหนดจุดเริ่มต้นของ here document ซึ่งเป็นรายชื่อบุคคลพร้อมวันเกิด เพื่อใช้เป็นข้อมูลนำเข้าของคำสั่ง grep เครื่องหมาย + ซึ่งอยู่บรรทัดสุดท้ายเป็น limit string ซึ่งใช้เป็นเครื่องหมายกำกับจุดสิ้นสุดของข้อมูล และเป็นเครื่องหมายเดียวกับเครื่องหมายกำกับจุดเริ่มต้น

ตัวอย่างการทำงานของโปรแกรมเป็นดังนี้

```
$ birthday Jenny
Jenny       January 23

$ birthday June
Alex        June 22
Nancy       June 26
```

การประยุกต์ใช้งาน Here document

โปรแกรมที่มีการประยุกต์ใช้งาน Here document ได้ดีเยี่ยมโปรแกรมหนึ่งคือ bundle ซึ่งเขียนขึ้นโดย Brian Kernighan และ Rob Pike สำหรับใช้ในการรวมแฟ้มข้อมูลแบบ text file หลายแฟ้มเข้าด้วยกันเป็นแฟ้มเดียว (package) เพื่อสะดวกในการนำไปติดตั้งในระบบอื่น การทำงานของโปรแกรมคล้ายกับการทำงานของโปรแกรม tar (tape archive) ซึ่งใช้ในการรวมแฟ้มเพื่อใช้ในการสำรองข้อมูล

โปรแกรม bundle เป็น Bourne shell script ที่ทำหน้าที่สร้าง Bourne shell script อีกตัวหนึ่ง โดยนำข้อมูลจากแฟ้มที่ต้องการรวมสร้างเป็น Here Document เมื่อนำแฟ้มนี้ไปทำงานในระบบอื่นที่ต้องการติดตั้งแฟ้ม จะทำการแยกแฟ้มที่รวมมาออกเป็นแฟ้มย่อยเช่นเดิม รายละเอียดของโปรแกรมเป็นดังนี้

```
$ cat bundle
#!/bin/sh
# bundle: group files into distribution package
echo '#!/bin/sh'
echo '# To unbundle, sh this file'
for i
do
       echo "echo $i 1>&2"
       echo "cat > $i << 'End of $i'"
       cat $i
       echo "End of $i"
done
```

หมายเหตุ

การกำหนด bundle ทำงานในสิ่งแวดล้อมของ Bourne shell (/bin/sh) สามารถทำได้ เนื่องจาก Linux กำหนดให้ /bin/sh เป็น symbolic link ไปยังโปรแกรม /bin/dash ดังนี้

```
$ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 Mar 30  2012 /bin/sh -> dash
```

dash ทำหน้าที่เป็น command interpreter หรือเป็น shell ที่มีขนาดเล็กว่า bash และเป็นไปตามมาตรฐาน POSIX ชื่อของ dash มาจาก Debian Almquist shell (dash)

การทดลองใช้งานโปรแกรม bundle

เมื่อพิจารณาโปรแกรม bundle จะเห็นว่าเป็นโปรแกรมที่สั้นใช้เฉพาะคำสั่งพื้นฐาน แต่เป็นโปรแกรม ที่ทำความเข้าใจจาก source code ได้ยากโปรแกรมหนึ่ง การเริ่มต้นทำความเข้าใจจึงควรต้องทดลองใช้งานโปรแกรมและทำความเข้าใจเป็นลำดับไป

การเตรียมก่าร

การทดลองใช้โปรแกรม bundle ต้องมีแฟ้มข้อมูลอย่างน้อยสองแฟ้ม ในที่นี้กำหนดให้มีชื่อแฟ้มเป็น file1 และ file2 ซึ่งมีข้อมูลดังนี้

```
$ cat file1
Readonly Shell Variables:
$0 - Name of the calling program
$n - Value of the nth command line argument
$* - All of the command line arguments
$@ - All of the command line arguments
```

```
$ cat file2
$# - Count of the command line arguments
$$ - PID number of the current process
$! - PID number of the most recent background
$? - Exit status of the last task that was executed
```

ทำการเรียกใช้โปรแกรม bundle โดยมี file1 และ file2 เป็น argument และทำการเปลี่ยนทิศทางผลลัพธ์ที่ได้ไปยังแฟ้ม package ดังนี้

```
$ bundle file1 file2 > package
```

จะได้แฟ้ม package ซึ่งเป็น shell script ที่มีข้อมูลในแฟ้ม file1 และ fille2 เป็น Here Document จากนั้นจึงทำการลบแฟ้ม file1 และ file2 และทำการตรวจสอบสิทธิในการใช้งานแฟ้ม combine โดยใช้คำสั่ง ls ดังนี้

```
$ rm file1 file2
$ ls -l package
-rw-------    1 jira     staff         505  Jul  10  09:30  package
```

จะเห็นว่าเป็นแฟ้มที่เจ้าของแฟ้มยังไม่มีสิทธิในการ execute จึงไม่สามารถเรียกให้ทำงานได้โดยตรง แต่สามารถเรียกให้ทำงานได้โดยกำหนดให้ shell ปัจจุบันสร้าง shell ใหม่เป็น Bourne shell (คำสั่ง sh) และให้ทำการ execute แฟ้ม package ซึ่งสามารถทำได้ไม่ว่าผู้ใช้จะมีสิทธิในการ execute แฟ้มนั้นหรือไม่ก็ตาม

```
$ sh package
```

ซึ่งเป็นการกำหนดให้ shell ปัจจุบันสร้าง shell ใหม่เป็น Bourne shell (คำสั่ง sh) และให้ทำการ execute แฟ้ม package ไม่ว่าจะมีสิทธิในการ execute หรือไม่ก็ตาม

คำอธิบายโปรแกรม

```
--------------
echo '#!/bin/sh'
echo '# To unbundle, sh this file'
```

เป็นการส่งข้อความที่กำหนดออกไปยัง standard output แต่เนื่องจากมีการทำ output redirection ไว้จาก command line ดังนั้นข้อความนี้จึงเป็นข้อมูลในแฟ้ม package และเพื่อป้องกันเชลล์ดำเนินก่ารกับอักขระพิเศษที่อาจมีในข้อความจึงต้องกำกับด้วย single quote

```
for i
```

เป็นการกำหนดโครงสร้างการทำซ้ำแบบ for เนื่องจากไม่มีคำสำคัญ in จึงเป็นการกำหนดให้นำ command line argument มาเป็นรายการที่ต้องดำเนินการ เมื่อมีการเรียกใช้ bundle ดังนี้

```
$ bundle file1 file2 > package
```

รายการที่ต้องดำเนินการของโครงสร้าง for คือ file1 file2 สำหรับแฟ้ม package ไม่อยู่ในรายการนี้ เพราะเป็นการเปลี่ยนทิศทางของโปรแกรม bundle

```
echo "echo $i 1>&2"
```

เป็นการส่งข้อความ "echo $i 1>&2" ซึ่งในการทำงานรอบแรกของการทำงานจะเป็น "echo file1 1&2" ไปยังแฟ้ม package

```
echo "cat > $i << 'End of $i'"
```

เป็นการส่งข้อความ cat > file1 << End of file1 ไปยังแฟ้ม package ซึ่งเมื่อทำงาน คำสั่ง cat > $i << 'End of $i' เป็นการกำหนดให้ cat ทำการ redirect ข้อมูลจาก here document ที่มีสายอักขระ End of <ชื่อแฟ้มในตัวแปร $i> เป็น limit string เริ่มต้น

```
cat $i
```

เป็นการส่งข้อมูลในแฟ้มที่กำหนดด้วยตัวแปร $i ทาง standard output และจะถูกเปลี่ยนทิศทางต่อไปยังแฟ้ม package เพื่อทำหน้าที่เป็น Here document

```
echo "End of $i"
```

เป็นการส่งข้อความ End of file1 เป็นเครื่องหมายปิดท้าย Here document ในแฟ้ม package

แฟ้มที่เกิดขึ้นจากการทำงานของโปรแกรม bundle

แฟ้มที่เกิดขึ้นจากการทำงานของโปรแกรม bundle ตามตัวอย่างในหัวข้อ “การทดลองใช้งานโปรแกรม bundle” มีรายละเอียดดังนี้

```
$ cat  package
#!/bin/sh
# To unbundle, sh this file
echo file1 1>&2                         # แสดงชื่อแฟ้ม file1 ออกทางจอภาพ
cat > file1 << 'End of file1'           # cat รับข้อมูลจาก Here document ส่งผลลัพธ์
Readonly Shell Variables:               # ไปยังแฟ้ม file1 โดยมีข้อความ End of file1
$0 - Name of the calling program        # เป็นเครื่องหมายกำหนดจุดเริ่มต้น
$n - Value of the nth command line argument
$* - All of the command line arguments
$@ - All of the command line arguments
End of file1                            # จุดสิ้นสุดของ Here document ชุดที่ 1
echo file2 1>&2
cat > file2 << 'End of file2'           # cat รับข้อมูลจาก Here document ส่งผลลัพธ์
$# - Count of the command line arguments # ไปยังแฟ้ม file2 โดยมีข้อความ End of file2
$$ - PID number of the current process  # เป็นเครื่องหมายกำหนดจุดเริ่มต้น
$! - PID number of the most recent background
$? - Exit status of the last task that was executed
End of file2                                #จุดสิ้นสุดของ Here document ชุดที่ 2
```

แฟ้ม package เมื่อทำงานสามารถสร้างแฟ้ม file1 และ file2 ที่มีข้อมูลเหมือนเดิมได้อย่างไร เป็นคำถามที่ทิ้งไว้ให้ผู้อ่านต้องศึกษาหาคำอธิบายด้วยตนเอง