INTRODUCTION TO UNIX SHELL SCRIPT PROGRAMING

1. SUMMARY OF COURSE 4

2. AGENDA 4

2.1. SHELL CHARACTERISTIC 4
2.2. UNIX BASIC COMMAND 4
2.3. SHELL COMMAND LANGUAGE 4
2.4. SHELL SCRIPT PROGRAMMING 4
2.5. LOGIN ENVIRONMENT 4

3. SHELL CHARACTERISTIC 4

3.1. PROGRAM EXECTION 4
3.2. FILENAME SUBSTITUTION 4
3.3. INPUT/OUTPUT REDIRECTION (>, <, >>, | , TEE) 4
3.4. PIPELINE HOOKUP 5
3.5. ENVIRONMENT CONTROL 5
3.6. INTERPRETIVE PROGRAMING LANGUAGE 5

4. SHELL PROGRAMMING 5

5. SHELL COMMAND LANGUAGE 5

5.1. METACHARECTORS 5
5.2. SPECIAL CHARACTER 6
5.3. INPUT/OUTPUT REDIRECTION 7
5.4. LIFT CYCLE OF PROCESS(การเริ่มต้นและการสิ้นสุดการ EXECUTE ของ PRECESSES) 8

6. SHELL SCRIPT PROGRAMMING 10

6.1. SHELL PROGRAMMING OR SHELL SCRIPT 10
6.2. VARIABLE 10
6.3. SHELL PROGRAMING CONSTRUCTS 16

7. LOGIN ENVIRONMENT 25

8. EXERCISES 25

9. WHERE TO GET MORE INFORMATION 27

9.1. MORE INFORMATION ON INTERNET FOR STARTER 27

10. APPENDIX คำสั่งทั่วไปที่ใช้ในการทำงาน 27

10.1. UNIX BASIC COMMAND 27

1. Summary of Course

Create your own shell script by bourne shell for easy to use your unix system

2. Agenda

2.1. Shell Characteristic

2.2. Unix basic command

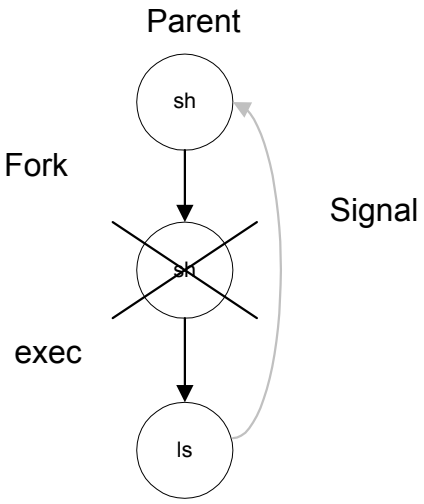
2.3. Shell Command Language

2.4. Shell Script Programming

2.5. Login Environment

3. Shell Characteristic

3.1. Program exection



รูปแบบทั่วไปของ command line คือ

command (argument) < CR >

Shell จะรับเอา argument ที่ใส่เข้าไปส่งไปยังโปรแกรมที่ต้องการ execute ตาม command ที่สั่งและหลังจากเสร็จ

สิ้นการทำงานตามคำสั่งแล้ว การควบคุมการติดต่อจะกลับมาอยู่ที่ shell อีกครั้ง เพื่อรอรับคำสั่งต่อไป ซึ่งจะแสดงให้ผู้ทราบโดยขึ้น command prompt

3.2. filename substitution

shell จะทำการแปลความหมายสัญลักษณ์พิเศษ ที่สามารถใช้อ้างอิงถึงชื่อ file ที่เรียกว่า metacharacters ก่อนที่จะ execute โปรแกรม ช่วยให้โปรแกรมไม่จำเป็นต้องรับรู้การแทนความหมายด้วยสัญลักษณ์เหล่านั้น

3.3. Input/Output Redirection (>, <, >>, | , tee)

shell สามารถกำหนดให้โปรแกรมรับ input หรือส่ง output ไปยังที่อื่นที่ไม่ใช่ standard input หรือ output (Keyboard และจอภาพที่ผู้ใช้ติดต่อกับระบบ) ได้กรณีของ input redirection นั้น shell จะทำการเปิด file

ที่ต้องการอ่านแล้วเชื่อมไปยัง standard input ให้โปรแกรมมาอ่านตามปกติจาก standard input ส่วนกรณี output redirection ก็ทำนองเดียวกัน คือ ส่ง output ของโปรแกรมไปยัง standard output ที่ เชื่อมโยงไปยัง file ที่ต้องการ โดยลักษณะนี้โปรแกรมจะรับส่งข้อมูลจาก standard input / output เท่านั้น

3.4. Pipeline hookup

เป็นการเชื่อมโยงโปรแกรม 2 โปรแกรม เข้าด้วยกันโดย output ของโปรแกรมแรกจะถูกส่งไปเป็น input ของโปรแกรมถัดมา

3.5. Environment Control

เราสามารถเปลี่ยนแปลง environment ต่าง ๆ ในการทำงานให้เหมาะสมกับความต้องการได้โดยผ่าน shell

3.6. Interpretive Programing Language

shell มี programming language ทีมีประสิทธิภาพ เรียกว่า shell script Bourne shell เป็น shell ที่ถูกพัฒนาขึ้นมาโดย Stephen Bourne

4. Shell Programming

รายละเอียดจะกล่าวเป็น 3 หัวข้อใหญ่คือ shell command languages, shell programming และการแก้ไข login environment

5. Shell Command Language

5.1. Metacharectors

หรือเรียกอีกอย่างว่า wild cards ใช้ในการอ้างอิงถึงกลุ่มของ file โดยระบุเพียงสั้น ๆ

5.1.1. asterisk (*) **ใช้แทนกลุ่มของตัวอักษร รวมทั้ง** null string **ด้วย**

5.1.2. questionmark (?) **ใช้แทนตัวอักขระใด ๆ** 1 **ตัว**

5.1.3. brackets ([ ]) **ใช้แทนอักขระตัวหนึ่งตัวใดที่ระบุไว้ หรือ ตัวเลข / ตัวอักษร ใน** range **ที่กำหนด**

<u>Example 1</u>

```
$ ls <CR>
. aab        a.ab     bat      cat 1    cat 3
a            aaaa     bye      cat 2
$ ls * <CR>
a            aaaa     bye      cat 2
a.ab         bat      cat 1    cat 3
$ ls a* <CR>
a            a.ab     aaaa
$ ls *b* <CR>
a.aab        bat      bye
```

```
$ ls b?? <CR>
bat          bye
$ ls ?a? * <CR>
aaaa         bat          cat 1        cat 2        cat 3
$ ls [ b - c ] at * <CR>
bat          cat 1        cat 2        cat 3
$ ls cat [ 1 - 2 ] <CR>
cat 1        cat 2
```

5.2. Special Character

นอกจาก metacharacters ยังมีอักขระอื่นที่มีความหมายพิเศษ ดังนี

5.2.1. space, tab, newline เป็นตัวแยก parameters

5.2.2. < , > , >> ใช้สำหรับ input / output redirection โดย < เป็น I / p redirection, > เป็น o / p redirection เป็นการ overwrite ลงไปใน file , >> เป็น o / p redirection ที่เป็นการเขียนต่อ ท้าย file เดิมที่มีอยู่

5.2.3. pipe (|) ใช้เป็น pipe เชื่อมโยงคำสั่ง 2 คำสั่งต่อกัน

5.2.4. semicolon (;) ใช้คั่นคำสั่งที่ต้องการให้ทำงานเป็นลำดับ ในการเขียนคำสั่งหลายคำสั่งใน บรรทัดเดียวกัน

5.2.5. ampersand (&) ใช้ส่งงานให้ไป run เป็น background เพื่อให้สามารถใช้จอภาพทำงานอย่างอื่นได้ขณะที่โปรแกรม กำลัง execute

5.2.6. single quotes (‘ ‘) ใช้ยกเลิกความหมายพิเศษของ special characters ทั้งหมด

5.2.7. double quotes (“ “) ใช้ยกเลิกความหมายพิเศษของ special characters ยกเว้น $ และ ‘

5.2.8. backquotes หรือ grave accent marks (‘ ‘) ใช้ในการแทน argument ของคำสั่งหนึ่งด้วย อีกคำสั่งหนึ่ง คือ ทำคำสั่ง ซ้อนคำสั่ง

5.2.9. dollar sign ($) ใช้ในการแสดงค่าตัวแปร

5.2.10. back slach (\ ) ใช้ยกเลิกความหมายพิเศษของ special characters

<u>Example 2</u>

```
$ banner happy birthday “to you” <CR>
HAPPY
BIRTHDAY
TO YOU
```

$ echo ‘ $ $ What is your name ? $$’ <CR>

$$ What is your name ? $$

$ echo \ \ \ ‘ \” \ ? \ * \ $ \ ‘ <CR>

\ ‘ “ ? * $ ‘

$ cd ; pwd ; ls <CR>

เป็นการเปลี่ยน directory ไปที่ home directory แล้วดูว่าอยู่ที่ directory ใดแล้ว แสดง file ต่างๆ ใน directory นี้ตามลำดับ

5.3. Input/Output redirection

Symbol	meaning
<	รับ input จาก file แทน keyboard
>	ส่ง output ไปที่ file แทน terminal
>>	ส่ง output ไปต่อท้าย file ที่มีอยู่แล้ว
|	ส่ง output ไปเป็น input ของอีกคำสั่ง
tee	ส่ง output แยกจากเดิมไปที่อีก file หนึ่ง
` `	ส่ง output ไปเป็น argument ของอีกคำสั่งหนึ่ง

<u>Example 3</u>

$ spell memo > misspell <CR>

เป็นการหาคำสะกดที่ไม่มีในภาษาอังกฤษจาก file ชื่อ memo ไปใส่ไว้ที่ file ชื่อ misspell ระวังอย่าใช้ sort list > list <CR> เพราะเวลาทำคำสั่ง sort shell จะ clear file ที่จะ เก็บ output ก่อน แล้วจึงทำการจัดเรียง แล้วใส่ output ลงไปใน file ที่ถูก clear นั้น ดังนั้น ถ้าทำการ sort แล้วใส่ file ชื่อเดิม จะเป็นการ sort file ว่าง ๆ

<u>Example 4</u>

$ cat trial1<CR>

This is the first line of trial1.

Hello.

This is the last line of trial1.

$ cat trial2<CR>

This is the beginning of trial2.

Hello.

This is the end of trial2.

$ cat trial2 >> trial1<CR>

$ cat trial1<CR>

This is the first line of trial1.

Hello.

This is the last line of trial1.

This is the beginning of trial2.

Hello.

This is the end of trial2.

Example 5

$ date | cut -c12-19<CR>

10:23:34

Example 6

$ date | tee temp | cut -c12-19<cr>

10:23:35

$ cat temp<CR>

Tue Nov 27 10:23:35 1990

Example 7

$ banner 'date|cut -c12-19'<CR>

15 : 23 : 35

5.4. Lift cycle of process(การเริ่มต้นและการสิ้นสุดการ execute ของ Precesses)

ปกติ shell จะเริ่มต้น process เมื่อรับ command เข้าไป และสิ้นสุดเมื่อทำงานตาม command นั้นเสร็จลงแล้ว แต่ยังมีการสั่งให้ทำงานในลักษณะอื่นดังนี้

5.4.1. การสั่งให้ทำงานในเวลาอื่นหลังเวลาในปัจจุบัน

ใช้คำสั่ง batch หรือ at

batch เป็นการสั่งให้ run process ในเวลาที่ system เป็นผู้กำหนด ในกรณีที่เป็น process ที่ต้องใช้เวลาของระบบมาก โดยงานต่าง ๆ นั้นจะถูกจัดคิวไป run เมื่อมี load ลดลงถึงระดับที่ยอมรับได้ เพื่อให้ขณะที่ run ระบบยัง respond ต่อผู้ใช้อื่นได้เร็ว

format : batch<CR> หรือ batch *command_line*<CR>

first-command<CR> <^d>

:

last_command<CR>

<^d>

Example 8

$ batch grep dollar * > dol_file<CR>

<^d>

job 155223141.b at Sun Dec 7 11:14:54 1990

at เป็นการสั่งให้ run process ตามเวลาที่ผู้ใช้กำหนด

format : at time<CR>

…commands <CR> …

<^d>

โดย ถ้าเวลาที่ต้องการไม่ใช่วันนี้ใน time ต้องระบุวันที่ด้วย

Example 9

$ at 8:15am Feb 27 <CR>

banner happy birthday | mail emily<CR>

< ^d >

job 453400603.a at Thurs Feb 27 08: 15: 00 1990

5.4.2. **การดู status ของ process**

ใช้คำสั่ง ps

options :

-e แสดงทุก process ที่กำลัง run อยู่

-f แสดงข้อมูลที่ประกอบด้วย

หมายเหตุ: ps -ef ใน unix ที่เป็น BSD ให้ใช้ ps -aux

UID user id ของเจ้าของ process	PID process Id	PPID process id ของ parent process	C การใช้ CPU ในการจัดลำดับ	STIME เวลาที่เริ่ม run	TTY Terminal ที่สั่ง Run	TIME เวลาที่ใช้ในการทำงาน	CMD command

ถ้าไม่ใส่ option จะแสดงเฉพาะ process ของ user นั้น และมีข้อมูลเพียง 4 columns คือ PID, TTY, TIME และ COMMAND ถ้าต้องการดู process ใน background ให้ใส่ & ไว้ข้างหลังด้วย

Example 10

$ grep word * > temp &<CR>

28223

$ ps<CR>

PID	TTY	TIME	CMD
28124	tty1	00:00	sh

28223 tty1 00:04 grep

28224 tty1 00:04 ps

- การหยุด process ที่ run อยู่

ใช้คำสั่ง kill

format : kill [-signo] PID<CR>

signo : 9 sure kill

Example 11

$ kill -9 473<CR>

เป็นการ break การทำงานของ process No. 473

- การสั่ง run ใน background

ทำโดยใส่ & ไว้ท้าย command line วิธีนี้ process จะ run ได้ขณะที่ผู้ใช้ยังไม่ log out เท่านั้น แต่ถ้าต้องการให้ run ต่อไป แม้จะ logout ไปแล้วให้ใช้คำสั่ง nohup เพิ่มเข้ามา

format : nohup *command* &<CR>

Example 12

$ nohup grep word * > word.list &<CR>

เป็นการหาบรรทัดที่มีคำว่า "word" จากทุก file แล้วเก็บไว้ใน file ชื่อ word.list โดยเป็นการทำงานใน background

6. Shell Script Programming

การทำงานของ shell นอกจากรับคำสั่งเป็น command mode
แล้วยังสามารถให้ผู้ใช้สร้างโปรแกรมคำสั่งขึ้นเองได้โดยใช้ commands, variables และ control structures
โปรแกรมเหล่านี้จะเรียกว่า shell scripts ซึ่งจะเขียนขึ้นโดยใช้ ed และ vi editor

6.1. Shell Programming or Shell Script

6.1.1. - **การตั้งชื่อ ไม่ควรตั้งซ้ำกับ** system command

6.1.2. - **การ** execute **หลังการสร้างโปรแกรมโดย** editor **เสร็จ** permission **ของ** file **จะไม่ให้ทำการ** execute **เราสามารถ** execute **ได้โดยใช้คำสั่ง** sh **ตามด้วยชื่อโปรแกรมนั้น หรืออาจทำการแก้** permission **ด้วยคำสั่ง** chmod **เพิ่ม** execute permission **เข้าไป หลังจากนั้นโปรแกรมจะ** run **ได้โดยสั่งชื่อโปรแกรมนั้นโดยตรง**

6.2. Variable

6.2.1. **การตั้งชื่อ**

ชื่อตัวแปร จะต้องประกอบด้วยตัวอักษร ตัวเลข หรือ underscore (_) โดยไม่มี blank space หรืออักขระอื่นๆ ปนอยู่เช่น ., !, ? เป็นต้น และต้องขึ้นต้นชื่อด้วยตัวอักษรหรือ _ เท่านั้น

Example 13

Valid variables :	Invalid variables :
i5	5i
length	.length
input_file	input file
HOME	
_cflag	

6.2.2. **การกำหนดค่า**

วิธีแรก ใช้ = เป็น assignment operator โดยที่ต้องไม่มี space อยู่หน้าหรือหลังเครื่องหมาย = ค่าที่ assign อาจเป็นค่าคงที่ ตัวอักขระ (ถ้าเป็น space bar ต้องใช้ quotes) , ค่าของตัวแปรอื่น หรือ output ของ command ที่แทนด้วย backquotes ก็ได้

Example 14

```
$ length=80
$ file_name="memo"
$ time='date | cut -c12-19'
```

วิธีที่สอง ใช้คำสั่ง read ให้ผู้สั่ง execute โปรแกรมเป็นผู้กำหนดค่า

format : read *variable*<CR>

Example 15

```
$ cat mknum<CR>
echo Type in name
read name
echo Type in number
read num
echo $name $num >> list
$ chmod u+x mknum<CR>
```

6.2.3. **การแสดงค่า**

การนำค่าตัวแปรไปใช้ให้นำหน้าชื่อตัวแปรด้วย $

Example 16

```
$ value=0
$ echo $value
0
3 value='expr $value + 1'
3 echo $value
1
```

6.2.4. ชนิดของตัวแปร

สามารถแบ่งได้เป็น 3 ชนิด ได้แก่ user defined variables , shell variables และ read-only shell variables

6.2.4.1. User variables

เป็นตัวแปรที่ผู้ใช้กำหนดขึ้นเอง สามารถอ่านคำและเปลี่ยนแปลงคำได้

เมื่อ assign ค่าตัวแปรแล้วต้องการลบทิ้งใช้ unset variable<CR>

การแสดงค่าตัวแปรทั้งหมดที่มีอยู่ ใช้ set<CR>

เราเปลี่ยนตัวแปรที่กำหนดขึ้นให้แก้ไขไม่ได้ได้โดยใช้ readonly variable<CR>

การแสดงค่าตัวแปร read-only ทั้งหมด ใช้ readonly<CR>

Example 17

```
$ name='john smith'
$ age=20
$ echo "$name $age"
john smith 20
$ unset age
$ echo $age

$ readonly name
$ name=mike
name: is read only
$ readonly
readonly name
```

6.2.4.2. Shell variables

เป็นตัวแปรที่ประกอบกันเป็น environment ของระบบ ซึ่งจะถูก set ค่าขึ้นทันทีที่มีการ login เข้าระบบ shell เป็นผู้กำหนด แต่ผู้ใช้อ่านและสามารถเปลี่ยนแปลงได้

HOME	จาก file ชื่อ เป็นค่า home directory ของ user โดยเริ่มแรกที่ login เข้าระบบ จะถูกอ่านค่ามา/etc/passwd ขณะที่ทำงานอยู่ในระบบ ถ้าต้องการไปที่ home directory ให้ใช้คำสั่ง cd<CR> จะเปลี่ยนที่อยู่จาก directory ปัจจุบันเป็น home directory ได้ทันที
LOGNAME	ชื่อ user
PATH	เป็นตัวแปรที่บอก shell ว่า command ที่ต้องการ execute จะหาได้จาก directory ใด โดยแต่ละ directory จะแยกกันโดยใช้ colon (:) คั่น shell จะทำการหา command จาก directory ทางซ้ายไปขวา ถ้า : อยู่ข้างหน้า หมายถึงให้หาจาก current drectory ก่อน
CDPATH	เป็นที่เก็บ path สำหรับคำสั่ง cd ถ้าไม่กำหนดไว้ path สำหรับ cd จะเป็น home directory
PS1	เครื่องหมาย prompt หลักของ shell default คือ $
PS2	เครื่องหมาย prompt รอง เมื่อมีการต่อบรรทัด default คือ >
TERM	บอก terminal type ซึ่งจำเป็นสำหรับ vi editor ถ้าระบุไม่ตรงกับความเป็นจริง จะทำให้การควบคุม cursor movement ผิดไป
MAIL	บอกชื่อ file ที่ใช้เก็บ mail
TZ	ค่าที่ใช้คำนวณหา local time (เทียบจาก Greenwish Mean Time)

<u>Example 18</u>

```
$ set
HOME= / usr / sam
LOGNAME=sam
PATH= : / usr / ucb : / bin : / usr / bin
PS1 = $
PS2 = >
TERM = pt2000
```

<u>Example 19</u>

```
$ PS1= ' Next command? = = = > '
Next command?  = = = > echo Hi
Hi
Next command? = = = >
```

<u>Example 20</u>

```
$ echo 'This message is
> on two lines.'
```

```
This message is
on two lines.
$ PS2 = ‘ More input? = = = > ‘
$ echo ‘This message is
More input? = = = > on two lines. ‘
This message is
on two lines.
```

6.2.4.3. Read-only shell variables

เป็นตัวแปรที่ shell กำหนดขึ้นโดยอัตโนมัติ ผู้ใช้อ่านได้แต่เปลี่ยนแปลงไม่ได้

$0	ชื่อของโปรแกรมปัจจุบัน
$1,..,$9	เก็บ argument ที่ 1 ถึง 9 บน command line เรียกว่า positional parameters
$*	แทน arguments ทั้งหมดบน command line
$#	เก็บจำนวน arguments บน command line
$$	PID ของ process ปัจจุบัน
$!	PID ของ background process ล่าสุด
$?	Exit status ของงานสุดท้ายที่ถูก executed

<u>Example 21</u>

```
$ cat myscript
set sue mary jane
echo $# ‘new arguments : ‘ $*
$ myscript john mike bill
3 new arguments : sue mary jane
```

คำสั่ง shift ใช้เลื่อน argument ตัวหลังมาแทนตัวที่อยู่ก่อนหน้า 1 ตัว

<u>Example 22</u>

```
$ cat myscript
echo ‘argl = ‘$1   ‘arg2 = ‘$2   ‘arg3 = ‘$3
shift
echo ‘arg1 = ‘$1   ‘arg2 = ’$2   ‘arg3 = ‘$3
shift
echo ‘arg1 = ‘$1   ‘arg2 = ‘$2   ‘arg3 = ‘$3
shift
echo ‘arg1 = ‘$1   ‘arg2 = ‘$2   ‘arg3 = ‘$3
```

```
shift
$ myscript john mike bill
arg1 = john   arg2 = mike   arg3 = bill
arg1 = mike   arg2 = bill   arg3 =
arg1 = bill   arg2 =        arg3 =
arg1 =        arg2 =        arg3 =
myscript : cannot shift
```

ตัวแปรของแต่ละ process จะเป็น local variables ของ process นั้นๆ แม้ว่าต่าง process กัน จะมีชื่อตัวแปรเหมือนกันแต่ไม่มีความเกี่ยวข้องกัน

Example 23

```
$ cat myscript
project = payroll
echo $0  $$  $project
subscript
echo $0  $$  $project
$ cat subscript
echo $0  $$  $project
project = marketing
echo $0  $$  $project
$ myscript
myscript 541 payroll
subscript 542
subscript 542 marketing
myscript 541 payroll
```

parent process สามารถส่งค่าตัวแปรไปให้ child process ของมันได้โดยคำสั่ง export

Example 24

```
$ cat myscript
export project
project = payroll
echo $0 $$ $project
subscript
echo $0 $$ $project
$ cat subscript
echo $0 $$ $project
```

```
project = marketing
echo $0 $$ $project
$ myscript
myscript 541 payroll
subscript 542 payroll
subscript 542 marketing
myscript 541 payroll
```

ถ้าต้องการให้มีการ share ตัวแปรร่วมกันระหว่าง script ให้ใช้ ในการเรียก subscript

Example 25

```
$ cat myscript
project = payroll
echo $0 $$ $project
. subscript
echo $0 $$ $project
$ cat subscript
echo $0 $$ $project
project = marketing
echo $0 $$ $project
$ myscript
myscript 541 payroll
subscript 541 payroll
subscript 541 marketing
myscript 54 1 marketing
```

6.3. Shell Programing Constructs

6.3.1. comment

shell จะถือว่าข้อความที่ตามหลัง pound sign (#) ในแต่ละบรรทัดเป็น comment “

6.3.2. here document

คือข้อความที่ถูก redirect ไปเป็น input ของ command ในโปรแกรม

format : command << delimiter <CR>

… input - lines <CR> …

delimiter <CR>

โดยที่ delimiter เป็นอักขระ หรือกลุ่มอักขระก็ได้

Example 26

```
$ cat ch.text<CR>
echo Type in the file name.
read file1
echo Type in exact text to be changed.
read old_text
echo Type in the exact new text to replace the above.
read new_text
ed $file1 <<!
  g / $old_text / s / / $new_text / g
  w
  q
!
```

6.3.3. return code

command ส่วนใหญ่จะมีการ set return code เมื่อ execute command เสร็จโดยจะ return ค่า 0 ถ้า execute ได้ถูกต้องปกติ ถ้าเป็นค่าอื่นแสดงว่าผิดปกติ ค่านี้จะเก็บอยู่ที่ $?

Example 27

```
$ cat hi<CR>
This is file hi.
$ echo $?<CR>
0
$ cat hello<CR>
cat: cannot open hello
$ echo $?<CR>
2
```

return code สามารถใช้คำสั่ง exit เป็นตัวกำหนดค่า $? นี้ได้

6.3.4. control structure

6.3.4.1. Conditional constructs

6.3.4.1.1. test command

เป็นคำสั่งที่ตรวจสอบว่าเงื่อนไขเป็นจริงหรือเท็จ ถ้าจริงจะให้ exit status เป็น 0

format : test_expression<CR>

expression ได้แก่

- r file	เป็นจริงถ้ามี file นี้อยู่และสามารถอ่านได้
- w file	เป็นจริงถ้ามี file นี้อยู่และสามารถเขียนได้
- x file	เป็นจริงถ้ามี file นี้อยู่และสามารถ execute ได้
- s file	เป็นจริงถ้ามี file นี้อยู่และมีขนาดมากกว่า 0
- z s1	เป็นจริงถ้าความยาวของ string s1 เป็น 0
- n s1	เป็นจริงถ้าความยาวของ string s1 ไม่เป็น 0
s1 = s2	เป็นจริงถ้า string s1 เท่ากับ s2
s1! = s2	เป็นจริงถ้า string s1 ไม่เท่ากับ s2
var1 -eq var2	เป็นจริงถ้า var1 มีค่าเป็นตัวเลขเท่ากับ var2
var1 -ne var2	เป็นจริงถ้า var1 มีค่าเป็นตัวเลขไม่เท่ากับ var2
var1 -gt var2	เป็นจริงถ้า var1 มีค่าเป็นตัวเลขมากกว่า var2
var1 -ge var2	เป็นจริงถ้า var1 มีค่าเป็นตัวเลขมากกว่าหรือเท่ากับ var2
var1 -lt var2	เป็นจริงถ้า var1 มีค่าเป็นตัวเลขน้อยกว่า var2
var1 -le var2	เป็นจริงถ้า var1 มีค่าเป็นตัวเลขน้อยกว่าหรือเท่ากับ var2

expression ข้างต้น สามารถใช้ร่วมกับ operators ต่อไปนี้

!	not
-a	binary and
-o	binary or (-a มี precedence สูงกว่า -o)
(expr)	สำหรับแบ่งกลุ่ม

<u>Example 28</u>

```
$ a = 1<CR>
$ b =  true<CR>
$ test  $a  gt  0  -a  $b "true"<CR>
$ echo $?<CR>
0
```

6.3.4.1.2. if-then-else

```
Format: if<CR>
command1<CR>
:
last command<CR>
then<CR>
command1<CR>
:
last_command<CR>
else<CR>
command1<CR>
:
last_command<CR>
:
last_command<CR>
fi<CR>
```

```
if(test-command) <CR>
then<CR>
:
:
else<CR>
:
:
fi<CR>
```

แบบแรก โปรแกรมจะ execute คำสั่งภายใต้ then ถ้าสามารถ execute คำสั่งสุดท้ายในเงื่อนไข if ได้สำเร็จ (return status เป็น true) แต่ถ้าไม่สำเร็จจะไป execute คำสั่งภายใต้ else ซึ่งสามารถละส่วน else นี้ได้ในกรณีที่ ไม่ต้องทำอะไร เป็น if-then เท่านั้น

แบบที่สอง จะเอา test command เข้ามาเป็นเงื่อนไข ถ้าเป็นจริงจะทำคำสั่งใต้ then ถ้าเป็นเท็จจะทำคำสั่งใต้ else

<u>Example 29</u>

```
$ cat list<CR>
if (test $# = 0)
    then
        echo 'Error, no arguments entered.'
    else
        ls  -al  $1
fi
$ list
Error, no arguments entered.
$ list /usr/john
```

```
total 496
drwxr-xr-x 12 root   root   320   Jan   13   13:59
        :
        :
```

6.3.4.1.3. case

```
format :  case test_string
in<CR>
pattern1) <CR>
        command1<CR>
                :
      last_command<CR>
; ; <CR>
pattern2)<CR>
        command1<CR>
                :
      last_command<CR>
; ;
:
*) <CR>
        command1<CR>
                :
      last_command<CR>
; ;
esac
```

case เป็น multiple branch decision mechanism จะเลือกทำกลุ่มคำสั่งของ pattern ที่ตรงกับ test string โดยใน pattern สามารถใช้ *, ?, [. .] หรือ | ประกอบได้ (| ใช้แยกตัวเลือกหลายตัวใน 1 pattern)

<u>Example 30</u>

```
$ cat digit<CR>
case "$1" in
0)    echo zero ; ;
```

```
                    1)   echo one ; ;
                    2)   echo two ; ;
                    3)   echo three ; ;
                    4)   echo four ; ;
                    5)   echo five ; ;
                    *)   echo "over five" ; ;
esac
$ digit 2<CR>
two
$ digit 7<CR>
over five
```

6.3.4.1.4. Looping

6.3.4.1.4.1.for

format : for loop_index [in argument_list]<cR>

do< CR >

loop-index จะถูก set ให้เป็นค่าตาม argument ใน argument_list ในแต่ละรอบเรียงไปจนครบทุกตัวจึงจบ loop กรณีที่ไม่ระบุ "in argument_list" จะหมายถึง "in $*"

<u>Example 31</u>

```
$ cat myscript
for day
   do
        echo 'Today is '$day
done
echo 'script finished.'
$ myscript Mon Tue Wed
Today is Mon
Today is Tue
Today is Wed
script finished.
```

6.3.4.1.4.2.while

format : while<CR>

command1<CR>

```
        :
    last_command<CR>

    do<CR>
        command1<CR>
            :
        last_command<CR>
    done<CR>
```

หรือ

```
while (test_command)<CR>
  do<CR>
    … commands<CR> …
done<CR>
```

โปรแกรมจะทำคำสั่งภายใน do ถึง done ไปเรื่อยๆ จนกระทั่งเงื่อนไขของ while เป็น false status

<u>Example 32</u>

```
$ cat myscript
counter = 0
while (test $counter -lt 5)
    do
        echo "$counter\c"
        counter = 'expr $counter + 1'
done
echo
echo 'Script finished.'
$ myscript
01234
Script finished.
```

6.3.4.1.4.3.until

```
format : until<CR>
    command<CR>
        :
    last_command<CR>
  do<CR>
    command1<CR>
```

หรือ

```
until (test_command)<CR>
do<CR>
  … commands<CR> …
done<CR>
```

```
            :
        last_command<CR>
    done<CR>
```

Example 33

```
$ cat myscript
counter = 0
until (test $counter -eq 5)
   do
        echo "$counter\c"
        counter = 'expr $counter + 1'
done
echo
echo 'Script finished.'
$ myscript
01234
Script finished.
```

6.3.4.2. Unconditional control statement

6.3.4.2.1. break

จะไปทำคำสั่งหลัง done, fi หรือ esac

6.3.4.2.2. continue

จะทำรอบต่อไปของ loop ทันที

- shell's garbage can : /dev/null

/dev/null เป็น file สำหรับให้ shell นำ output ที่ไม่ต้องการไปทิ้ง

Example 34

```
$ cat name<CR>
john
$ cat add<CR>
echo -n "Enter name : "
read x
while true ; do
        if test $x  ! = "q" ;  then
```

```
                echo $x >> name
                echo -n "Enter name : "
                read x
        else
                echo quit
                break
        fi
done
$add<CR>
Enter name : jack<CR>
Enter name : jame<CR>
Enter name : q<CR>
quit
$ cat name<CR>
john
jack
jame
```

6.3.5. Program execution

มี 5 วิธีในการ execute โปรแกรม

- by name ต้องมี execute permission ปกติหลังจากใช้ editor สร้างโปรแกรมเสร็จจะมี permission แค่ read กับ write เราทำให้ executable ได้โดยเปลี่ยน mode โดยคำสั่ง chmod : chmod u+x program<CR>
- ใช้ sh : sh program ทำได้โดยไม่ต้องมี execute permission
- ใช้ . ไม่ต้องการ execute permission
- exec ต้องการ execute permission ทำแล้วจะไม่กลับไปที่ script เดิม
- command substitution ใช้ backquotes (‘ ‘) เป็นการซ้อน command ด้วย command

<u>Example 35</u>

```
$ cat script1<CR>
echo 'script1 PID is ' $$
$ cat script2<CR>
echo $0 ' PID is ' $$
. script1
echo 'This is last line of script2.'
$ script2<CR>
```

```
script2 PID is 514
script1 PID is 514
This is last line of script2.
```

Example 36

```
$ chmod u|x script1<CR>
$ cat script1<CR>
echo 'script1 PID is ' $$
$ cat script2<CR>
echo $0 ' PID is ' $$
exec script1
echo 'This is last line of script2.'
$ script2<CR>
script2 PID is 514
script1 PID is 514
```

6.3.6. Debugging Programs

sh -v program จะพิมพ์ input lines ออกตามที่ระบบอ่านเข้าไป

sh -x program พิมพ์คำสั่ง และ arguments ตามที่ถูก execute

7. Login Environment

เมื่อ login เข้าระบบ shell จะไปอ่าน file ชื่อ .profile ใน login directory ซึ่งมีหน้าที่ควบคุม shell environment ซึ่งเราสามารถเพิ่มหรือเปลี่ยนแปลงคำสั่งใน .profile ได้ โดยที่หลังจากเปลี่ยนแปลงเรียบร้อยแล้ว ให้ทำการ initialize ใหม่โดยใช้ . .profile<CR>

8. Exercises

1. สมมุติว่าที่ current directory มีไฟล์ดังนี้

   Abel, Cain, George, Gorth, Greg, Sam, Ted, Trod

   ให้ใช้คำสั่ง ls เพื่อแสดงชื่อไฟล์ที่ตรงกับเงื่อนไขต่อไปนี้

   a) มี 3 ตัวอักษร

   b) มี 4 ตัวอักษร

   c) ขึ้นต้นด้วย G แล้วตามด้วย e หรือ o

   d) ขึ้นต้นด้วย T และลงท้ายด้วย d

2. set prompt (ใน Bourne Shell) ให้เป็น "ชื่อ User:"

3. list user-di และ user-name ในไฟล์ /etc/passwd ที่สั่งให้ run คำสั่ง /bin /csh โดยแสดงเรียงตามลำดับ user-id

4. เขียน IF-THEN-ELSE statement เพื่อทดสอบตัวแปรตัวหนึ่งว่ามีคำเป็น “data”, “source”, “comments” หรือค่าอื่นๆ
5. เขียน CASE statement ทดสอบตัวแปรตัวหนึ่งว่ามีค่าเป็น “data”, “source”, “comments” หรือค่าอื่นๆ
6. เขียน shell script ชื่อ whois โดยมีรูปแบบดังนี้

   $ whois <user-name>

   โดยให้แสดงชื่อเต็มของ user-name ที่รับเข้ามา (ชื่อเต็มคือ comment field ใน /et/passwd)
7. เขียน shell script ชื่อ showfile โดยมีรูปแบบดังนี้

   $ showfile [filename….]

   โดยให้แสดงชนิดของ file ที่รับเข้ามา ถ้าไม่ระบุ หมายถึง ทุก file ใน current directory

   <u>หมายเหตุ</u> script ตัวนี้จะทำหน้าที่เหมือนคำสั่ง file แต่คำสั่ง file ถ้าไม่มี argument จะเกิด error
8. เขียน shell script ชื่อ lssize โดยทำงานเหมือน ls-al ทุกประการ แต่ให้แสดงเรียงตามลำดับของขนาดของ file จากมากไปหาน้อย
9. เขียน shell ชื่อ proglist โดยมีรูปแบบดังนี้

   $ proglist <progname> [<#copies>]

   เพื่อสั่งพิมพ์ไฟล์ที่ระบุใน progname ตามจำนวนชุดที่สั่ง

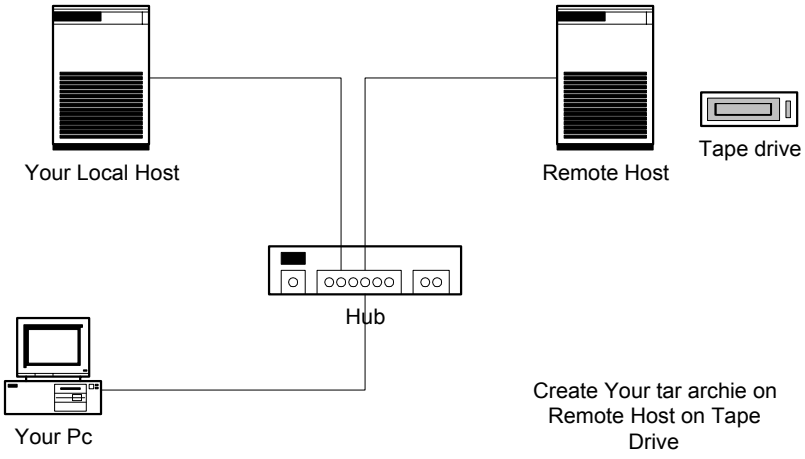
   - ถ้าไม่ได้ใส่จำนวนชุดถือว่าสั่งพิมพ์ 1 ชุด
   - check จำนวน argument ที่รับ
   - check <#copies> เป็นตัวเลขหรือไม่
   - check <programe> เป็นไฟล์หรือไม่
   - การสั่งพิมพ์ จะมีการจัดหน้าให้ด้วย
10. เขียน shell script ให้ run คำสั่งต่าง ๆ ในmenu ดังนี้

    <u>Main Menu</u>

    1. who
    2. date
    3. tty
    4. set
    5. exit.

    Please enter choice = >

11. เขียน Shell Script เพื่อทำการ Remote backup ข้อมูลทั้งหมดที่ Home Directoryไปเก็บเป็น tar format โดยขึ้น Tape ที่มี Tape Drive อยู่ที่อีกเครื่องใน Network เดียวกัน



9. Where to get more information

9.1. More Information on Internet for starter

http://uwsg.ucs.indiana.edu/uhelp/tutorials/begin.html

10. Appendix คำสั่งทั่วไปที่ใช้ในการทำงาน

10.1. Unix basic command

backup Command

compress command

Network command

jobs control command

Performance command

10.1.1. backup Command

10.1.1.1. tar

10.1.1.2. cpio

10.1.1.3. dd

10.1.2. Compress Command

10.1.2.1. compress

10.1.2.2. gzip

10.1.3. Network command

10.1.3.1. ftp

10.1.3.2. rsh

10.1.3.3. telnet

10.1.3.4. netstat

10.1.3.5. ifconfig

10.1.3.6. mail

10.1.3.7. arp

10.1.4. jobs control command

10.1.4.1. kill

10.1.4.2. cron

10.1.4.3. jobs

10.1.4.4. ps

10.1.5. Performance command

10.1.5.1. sar

10.1.5.2. vmstat

10.1.5.3. top