บทที่ 7 การเขียนเชลล์สคริพต์

จุดประสงค์เชิงพฤติกรรม

1 เพื่อให้สามารถบอกหน้าที่ของเชลล์ได้
2 เพื่อให้สามารถอธิบายโครงสร้างของเชลล์สคริพต์ได้
3 เพื่อให้สามารถเขียนเชลล์สคริพต์ได้
4 เพื่อให้สามารถเขียนฟังก์ชันได้
5 เพื่อให้สามารถอธิบายการใช้ตัวแปรได้
6 เพื่อให้สามารถใช้รหัสเอสกีควบคุมการแสดงผลได้

เชลล์เป็นองค์ประกอบหลักหนึ่งของโครงสร้างระบบลินุกซ์ มีหน้าที่สำคัญคือการรับคำสั่งจาก ผู้ใช้งานเพื่อสั่งให้ระบบปฏิบัติการได้ดำเนินการประมวลผลตามคำสั่ง นอกจากการรับคำสั่งทีละคำสั่ง แล้ว เรายังสามารถที่จะเขียนคำสั่งเป็นกลุ่มหรือไฟล์แบตช์ (Batch) ในลักษณะของสคริ

พต์ (Script) เพื่อให้ทำงานบนเชลล์ได้

7.1 เชลล์สคริพต์

โดยทั่วไปแล้วเชลล์ที่สามารถนำมาใช้งานบนระบบยูนิกซ์ได้แก่ บอร์นเชลล์ (sh) , คอร์น เชลล์ (ksh), หรือ ซีเชลล์ (csh) แต่เนื่องจากเชลล์ดังกล่าวต้องมีมูลค่าในการจัดซื้อจัดหา ไม่เหมาะที่จะ นำมาใช้กับระบบลินุกซ์ซึ่งเป็นซอฟต์แวร์ที่ไม่คิดมูลค่า จึงได้มีการสร้างเชลล์ที่มีคุณสมบัติในการ ทำงานเทียบเท่าหรือดีกว่าบอร์นเชลล์และเชลล์อื่นๆ และเป็นเชลล์ที่ไม่คิดมูลค่าของตัวโปรแกรม เรียก เชลล์นี้ว่า บอร์นอะเกนเชลล์ หรือ bash

รูปแบบคำสั่งของ bash ครอบคลุมคำสั่งทั้งหมดของบอร์นเชลล์ ดังนั้นสคริปต์ที่เขียนสำหรับ บอร์นเชลล์ส่วนใหญ่จึงสามารถเรียกใช้งานใน bash ได้โดยไม่ต้องแก้ไข จะมียกเว้นก็เช่น สคริปต์ที่เรียกใช้ตัวแปรพิเศษในบอร์นเชลล์ หรือใช้คำสั่งภายในของบอร์นเชลล์

รูปแบบคำสั่งใน bash ยังได้รับแนวความคิดจาก คอร์นเชลล์และซีเชลล์ เช่น การแก้ไขคำสั่ง การจดจำคำสั่งเก่า ตัวแปร $RANDOM และ $PPID เป็นต้น เวลาใช้เป็นเชลล์ทางคอมมานด์ไลน์ bash จะเติมชื่อโปรแกรม ชื่อไฟล์ ชื่อตัวแปร ให้ครบโดยอัตโนมัติเมื่อผู้ใช้กดปุ่ม TAB

7.1.1. เชลล์สคริพต์คืออะไร

ในการใช้คำสั่งบนเชลล์ของลินุกซ์นั้น เราสามารถที่จะใช้คำสั่งหลาย ๆคำสั่งบนบรรทัด เดียวกันได้หรือการใช้คำสั่งแบบซีเควนซ์หรือแบบตามลำดับโดยมีเครื่องหมายเซมิคอลอน (;) คั่น ระหว่างคำสั่งแต่ละคำสั่ง

นอกจากการใช้คำสั่งแบบซีเควนซ์สำหรับการเรียกคำสั่งหลาย ๆคำสั่งแล้ว เรายังสามารถใช้คำสั่งเป็นกลุ่มด้วยการบันทึกคำสั่งนั้นในไฟล์ข้อความหรือสคริพต์ ดังนั้นเชลล์สคริพต์ก็คือ ไฟล์ๆหนึ่ง ที่ได้บรรจุคำสั่งหลายๆคำสั่งไว้ในไฟล์ โดยในไฟล์ที่เป็นเชลล์สคริพต์นั้น จะต้องประกอบด้วย คำสั่ง ต่างๆ ตัวแปร สภาพแวดล้อม และส่วนของการควบคุมทิศทางการทำตามคำสั่ง (Flow control)

การเขียนสคริพต์จะทำเช่นเดียวกับการเขียนโปรแกรมด้วยภาษาคอมพิวเตอร์ โดยจะมีหลักเกณฑ์และไวยากรณ์เฉพาะในการเขียนคำสั่ง ในการเรียกใช้งานสคริพต์ก็สามารถทำได้เช่นเดียวกับการเรียกโปรแกรม

7.1.2. โครงสร้างของเชลล์สคริพต์

เชลล์สคริพต์จะเป็นมากกว่าการเอาคำสั่งในเชลล์หลายๆคำสั่งมาวางต่อๆกันแล้วนำไปใส่ไฟล์ไว้เพื่อให้เวลาเรียกใช้งานจะได้ประมวลผลตามลำดับที่เรียงไว้แต่จะรวมไปถึงการกำหนดคุณสมบัติเพื่อการตัดสินใจ การทำคำสั่งซ้ำ การติดต่อกับผู้ใช้การประมวลผลตัวแปร

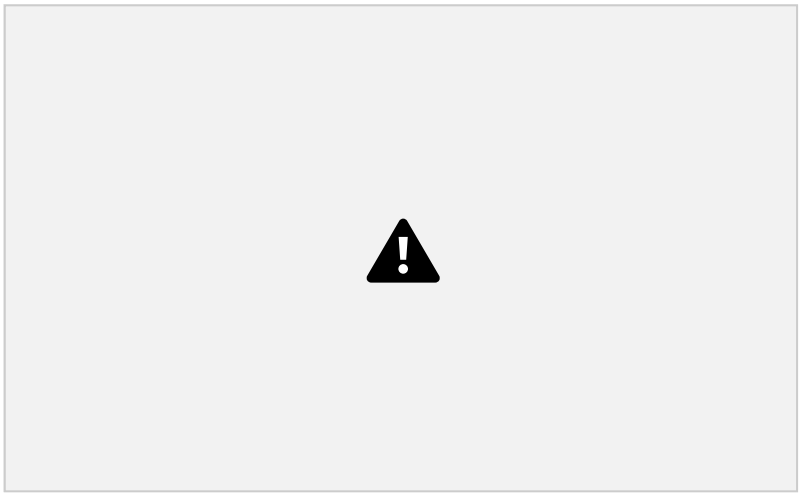
ในการเขียนเชลล์สคริพต์นั้นจะไม่มีส่วนของการประกาศตัวแปร

เหมือนกับภาษาคอมพิวเตอร ์บางภาษา นั่นคือเราสามารถที่จะกำหนดค่าให้กับตัวแปรที่ตรงส่วนไหนของเชลล์สคริพต์ก็ได ้โดยไม ่ต้องมีการกำหนดตัวแปรและชนิดของตัวแปรขึ้นมาก่อน โครงสร้างของเชลล์สคริพต์จะมี 2 ส่วน คือ

1. **ส่วนกำหนดตัวแปลสคริพต์** ส่วนนี้จะเป็นส่วนที่บอกให้รู้ว่าเชลล์สคริพต์นี้เป็นเชลล์สคริพต์ ที่ใช้เชลล์ประเภทอะไรเป็นตัวแปลสคริพต์ หรือกล่าวง่ายๆว่า จะใช้เชลล์อะไรมาคอมไพล์สคริพต์นี นั่นเอง การกำหนดจะให้กำหนดไว้ที่บรรทัดบนสุดของสคริพต์
2. **ส่วนของสคริพต์** ส่วนนี้จะประกอบด้วยคำสั่งต่างๆ ฟังก์ชั่น ตัวแปรและการกำหนดค่า ให้กับตัวแปร และหมายเหตุ

กำหนดตัวแปลสคริพต์

ตัวอย่างเชลล์สคริพต์



จากตัวอย่างข้างบนจะอธิบายได้ดังนี้

บรรทัดที่ 1 จะเห็นว่าเชลล์สคริพต์จะมีการเริ่มต้นด้วยบรรทัด "#!bin/bash" ซึ่งถ้าเป็นการใช้งานตามปกตินั้นเครื่องหมาย "#" จะแสดงถึงการเริ่มต้นของหมายเหตุ (comment) แต่เมื่อกำหนดเป็น #! จะเป็นเครื่องหมายพิเศษซึ่งบอกให้ลินุกซ์ใช้บอร์นอะเกนเชลล์ (bash) ในการแปลสคริพต์ดังนั้น เครื่องหมาย #! จึงต้องอยู่ในบรรทัดแรกของสคริพต์เสมอ

บรรทัดที่ 2, 4, 5 และ 9 เป็นคำสั่งใช้ในการนำข้อความที่อยู่ภายในเครื่องหมาย " " มา แสดงออกทางหน้าจอ มีข้อสังเกตในคำสั่งคือ มีการใช้ค่าในตัวแปรสภาพแวดล้อม (environment variable) หรือตัวแปรระบบมาใช้ในการแสดงผล คือ ตัวแปร USER สำหรับแสดงชื่อของผู้เข้าใช้ระบบ ตัวแปร PWD สำหรับแสดงไดเร็กตอรีปัจจุบันที่ผู้ใช้งานกำลังใช้งานอยู่ และใช้ตัวแปร time เพื่อแสดงวันที่โดยที่การเรียกค่าในตัวแปรสำหรับเชลล์สคริพต์นั้น จะต้องใช้คู่กับเครื่องหมาย "$"

บรรทัดที่ 3 และ 7 จะเป็นการเขียนหมายเหตุหรือคอมเม็นต์สำหรับใช้

ในการอธิบายคำสั่งหรือ การทำงาน ซึ่งจะใช้เครื่องหมาย “#” ไว้หน้าบรรทัด หมายเหตุเสมอ

บรรทัดที่ 6 เป็นบรรทัดที่มีการเรียกใช้คำสั่ง ls และมีหมายเหตุอยู่ในบรรทัดเดียวกัน ซึ่ง ตัวแปลสคริพต์จะเรียกใช้แต่เฉพาะคำสั่ง ls เท่านั้น โดยไม่นำเอาข้อความหลังเครื่องหมาย # มาแปล ร่วมด้วย

บรรทัดที่ 8 จะเป็นการกำหนดค่าให้กับตัวแปร โดยค่าที่กำหนดให้ตัวแปร time จะเป็นค่าที่ได้จากการเรียกใช้คำสั่ง date

หลังจากที่เราเขียนสคริพต์ตามต้องการแล้วจะต้องทำการจัดเก็บบันทึกเชลล์สคริพต์นั้นไว้ วิธีการในการบันทึกก็ขึ้นอยู่กับโปรแกรมที่เราใช้พิมพ์สคริพต์ โดยจะต้องมีการตั้งชื่อให้กับไฟล์สคริพต์ด้วยหลักการเดียวกับการตั้งชื่อไฟล์และไม่ควรที่จะตั้งชื่อให้ซ้ำกับคำสั่งของระบบ

7.1.3. การทำงานของเชลล์สคริพต์

เมื่อเขียนเชลล์สคริพต์และจัดเก็บบันทึกไว้แล้ว ตามปกติแล้วไฟล์เชลล์สคริพต์ที่เราเขียนนั้น จะยังคงไม่สามารถเรียกใช้งานหรือเอ็กซิคิวได้ ถ้าเราเรียกให้เชลล์สคริพต์นั้นเอ็กซิคิว โดยการพิมพ์ชื่อ เชลล์สคริพต์นั้น ที่บรรทัดคำสั่งในเชลล์ระบบจะฟ้องออกมาว่า

```
script_name: Permission denied
```

แสดงให้เห็นว่า เชลล์สคริพต์นี้ยังไม่สามารถเอ็กซิคิวได้ เนื่องจากคุณสมบัติของไฟล์หลังจากเราทำการ สร้างขึ้นใหม่นั้น ยังไม่มีคุณสมบัติในการเอ็กซิคิวนั่นเอง สำหรับการสั่งให้เชลล์สคริพต์เอ็กซิคิวหรือ ทำงานบนเชลล์ได้นั้น ถ้าแบ่งตามเชลล์ที่เกิดการทำงานตามสคริพต์แล้วนั้น มี 4 วิธีที่แตกต่างกัน ตาม รายละเอียด ดังนี้

(1) ส่งไฟล์สคริพต์รวมถึงค่าอากิวเม็นต์ที่จำเป็นให้ตัวแปลเชลล์ทำการแปลไฟล์สคริพต์โดยมีรูปแบบ คือ

- สำหรับบอร์นเชลล์หรือคอร์นเชลล์

```
# sh script_name
```

- สำหรับ ซีเชลล์

```
# csh script_name
```

- สำหรับบอร์นอะเกนเชลล์

```
# bash script_name
```

ตัวอย่าง เราเขียนไฟล์สคริพต์ชื่อ myscript ที่ใช้ bash เป็นตัวแปลสคริพต์ เราจะเอ็กซิคิวสคริพต์ได้ด้วยการพิมพ์คำสั่งที่เชลล์คือ bash myscript

(2) เรียกชื่อไฟล์สคริพต์โดยตรงรวมถึงอากิวเม็นต์ที่จำเป็น การสั่งให้

เชลล์สคริพต์ทำงานด้วย วิธีนี้จะต้องเปลี่ยนสิทธิในการใช้งานไฟล์ให้กับไฟล์สคริพต์เสียก่อนด้วยคำสั่ง cmod จึงจะสามารถ แอ็กซิคิวได้ เช่น cmod u+x myscript หรือ chmod 775 myscript จากนั้นจึงสามารถเรียกชื่อไฟล์โดยตรงดังนี้

```
# myscript
```

ในการเปลี่ยนแปลงสิทธิในการใช้งานไฟล์นั้น ผู้ที่เป็นเจ้าของไฟล์หรือผู้ใช้งานระดับ root เท่านั้นที่สามารถแก้ไขสิทธิได้

(3) สั่งให้เชลล์สคริพต์ทำงานโดยใช้คำสั่ง . *script_name* (ใช้เครื่องหมายจุดนำหน้าชื่อเชลล์สคริพต์) การใช้คำสั่งแบบนี้เชลล์สคริพต์จะทำงานที่เชลล์ปัจจุบัน นั่นคือ จะมีการทำงานทับบนเชลล์เดิมนั่นเอง ดังแสดงเป็นตัวอย่าง คือ

```
# . myscript
```

(4) สั่งให้เชลล์สคริพต์ทำงานโดยใช้คำสั่ง source *script_name* หรือคำสั่ง exec *script_name* โดยจะมีการทำงานที่เชลล์เดิม ตัวอย่างการใช้งาน คือ

```
# source myscript
```

หรือ

```
# exec myscript
```

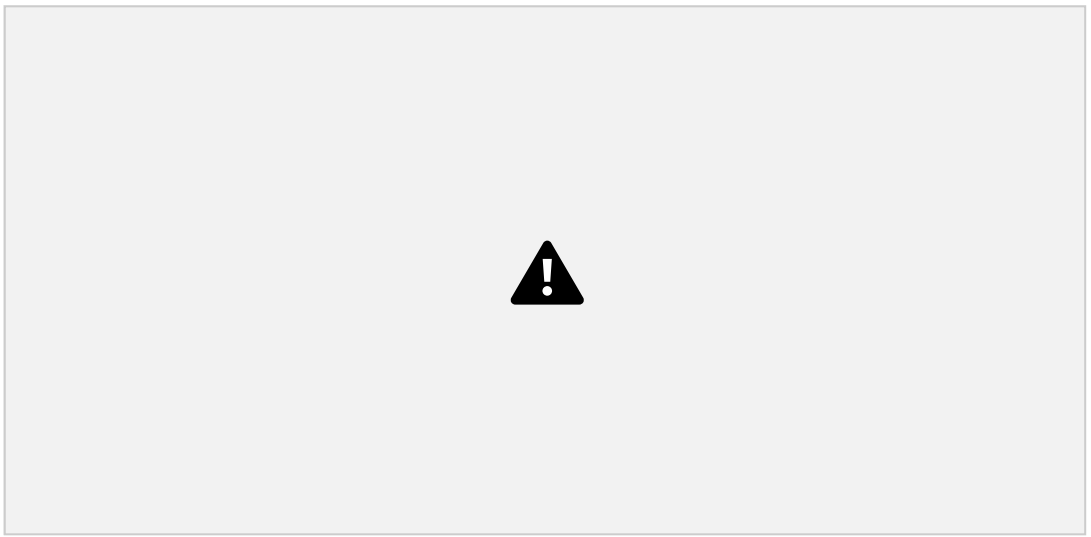
การทำงานของเชลล์สคริพต์จะเกิดขึ้นได้ 2 ลักษณะ คือ เกิดการทำงานขึ้นในเชลล์ปัจจุบัน และในเชลล์อื่นๆ หรือสับเชลล์ ความแตกต่างของการทำงานของเชลล์สคริพต์ระหว่างในเชลล์ปัจจุบัน และสับเชลล์ ดังจะยกตัวอย่างในการอธิบาย ดังนี้

จากตัวอย่างเชลล์สคริพต์ข้างล่าง

```
echo “Information Technology” day=`date` echo “Today is $day”
```

เมื่อเราบันทึกไฟล์สคริพต์ข้างต้นด้วยชื่อ first_script แล้ว เมื่อเราเรียกใช้งานสคริพต์จะได้ผลการ ทำงานดังแสดงในรูปที่ 7.2

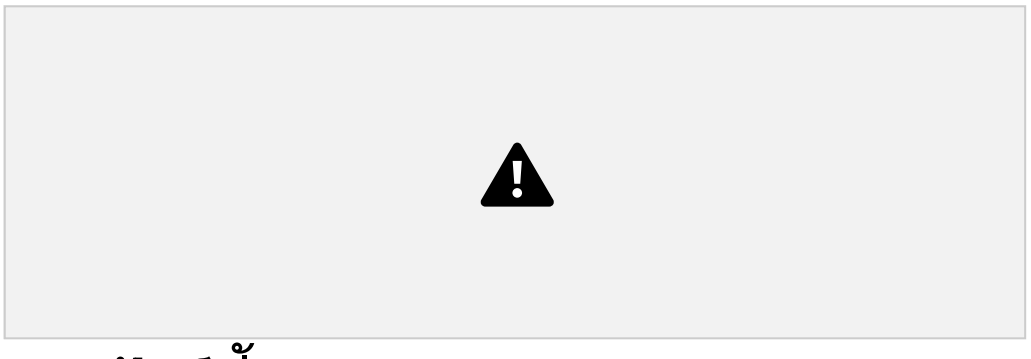
รูปที่ 7.2 การทำงานในเชลล์ปัจจุบัน จากนั้นทำการแก้ไขสคริพต์ firat_script ให้เป็นดังข้างล่าง



```
echo “Information Technology” day=`date` exit echo “Today is $day”
```

เมื่อเราทดลองเรียกใช้งานสคริพต์จะได้ผลการทำงานดังรูปที่ 7.3 จะเห็นว่าจะมีการแสดงข้อความ เพียงบรรทัดแรกของสคริพต์เท่านั้น ทั้งนี้เพราะว่ามีคำสั่ง exit ให้ออกจากเชลล์ที่สคริพต์นั้นกำลัง ทำงานอยู่ไปสู่เชลล์อื่นๆ นั่นเอง

รูปที่ 7.3 การทำงานของเชลล์สคริพต์ในสับเชลล์



7.2. ฟังก์ชั่น (Function)

นอกจากการเขียนสคริพต์ด้วยการใช้คำสั่งเรียงลำดับการทำงานตามความต้องการแล้ว เรายัง สามารถเขียนเป็นฟังก์ชั่นขึ้นในสคริพต์ได้ด้วยเช่นเดียวกับการเขียนโปรแกรมด้วยภาษาคอมพิวเตอร์ ซึ่งถ้าพิจารณาแล้ว ฟังก์ชั่นก็เป็นสคริพต์หนึ่งที่อยู่ในเชลล์สคริพต์นั่นเอง กล่าวคือ ภายในฟังก์ชั่นจะ ประกอบด้วยคำสั่งหลายๆคำสั่ง ฟังก์ชั่นจะถูกนำมาใช้งานในกรณีที่เราต้องมีการเรียกใช้คำสั่งกลุ่มใด กลุ่มหนึ่งบ่อยๆ โดยไม่ต้องเขียนคำสั่งกลุ่มนั้นทุกครั้งที่ต้องการใช้งาน

รูปแบบของฟังก์ชั่น

การเขียนฟังก์ชั่นจะมีรูปแบบในการเขียน 2 รูปแบบ คือ

รูปแบบที่ 1

```
function function_name { shell commands ......
}
```

รูปแบบที่ 2

```
function_name () { shell commands ......
}
```

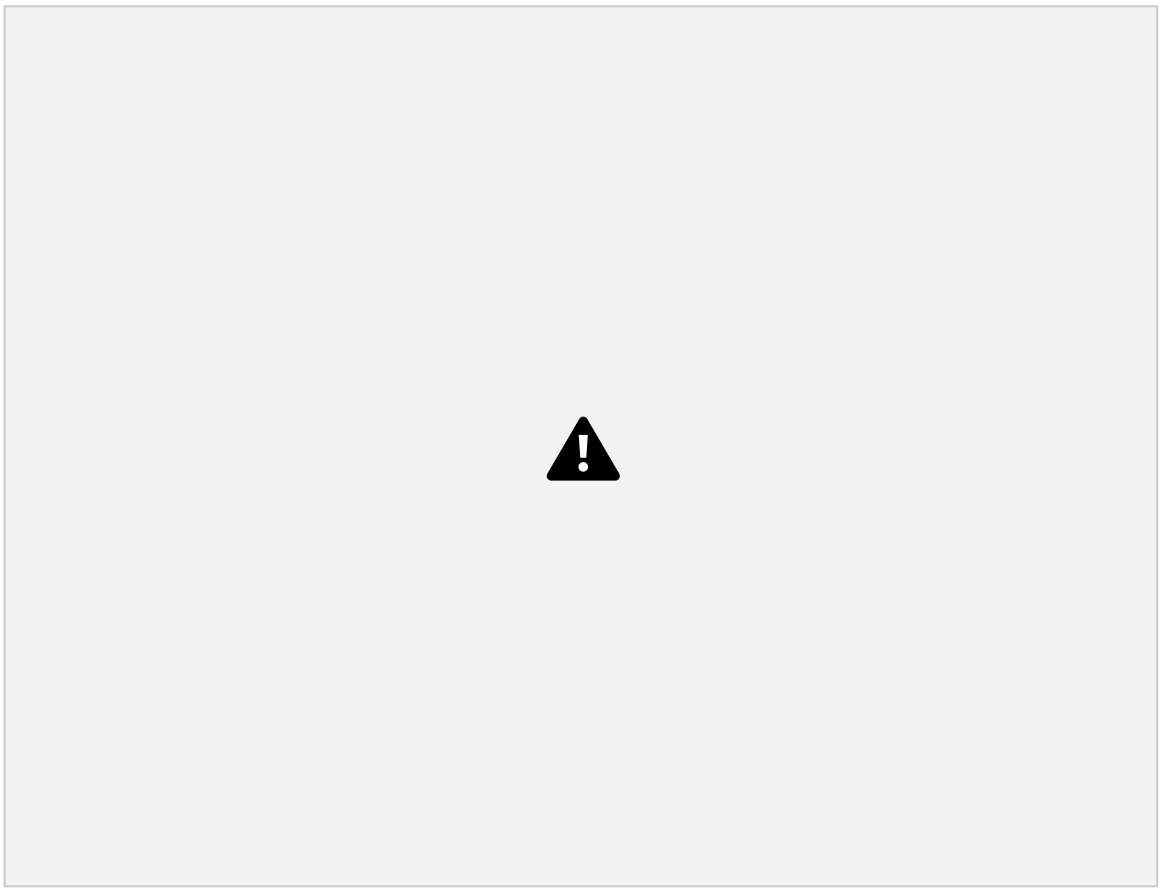
ตัวอย่างของการเขียนฟังก์ชั่น ดังแสดงต่อไปนี้ โดยเป็นสคริพต์ที่ใช้คำสั่งเชลล์สั้นๆ

```
function func_demo { echo “First number is $a” echo “Second number is $b”
} a=8 b=5
```

```
func_demo echo a=25 b=35 func_demo echo
```

จากตัวอย่างข้างต้น จะเห็นว่ามีฟังก์ชันอยู่ 1 ฟังก์ชัน ซึ่งใช้สำหรับการแสดงค่าของตัวแปร a และ b สำหรับในส่วนของโปรแกรมเชลล์นั้นจะเป็นการกำหนดค่าให้กับตัวแปรทั้งสอง และหลังจากที่กำหนดค่าให้กับตัวแปรทั้งสองแล้ว จะมีการเรียกฟังก์ชัน func_demo เพื่อแสดงผลค่าของตัวแปรทั้ง สอง โดยการดำเนินการนี้จะมีสองครั้งด้วยกัน ดังแสดงผลการทำงานของเชลล์สคริพต์ในรูปที่ 7.4

รูปที่ 7.4 แสดงการเรียกใช้ฟังก์ชัน จากตัวอย่างจะเห็นว่า การใช้ฟังก์ชันในเชลล์สคริพต์จะทำให้ประหยัดการเขียนรหัสคำสั่งและ ยังทำให้สคริพต์นั้นดูง่ายยิ่งขึ้น ในการทำงานของโปรแกรมจะเว้นไม่ถูกเรียกทำงานในช่วงตั้งแต่ชื่อ ฟังก์ชันจนกระทั่งจบบล็อก { shell commands } เรานิยมเขียนฟังก์ชันไว้ที่ต้นสคริพต์เพื่อให้สามารถถูกเรียกจากโปรแกรมหลักได้



7.3. ตัวแปร

ในภาษาคอมพิวเตอร์นั้น ตัวแปรหมายถึงเนื้อที่หน่วยความจำที่ถูกกันไว ้เพื่อใช้ในการถ่ายเท ข้อมูล โดยตัวแปรจะมีการอ้างถึงด้วยการกำหนดชื่อให้กับตัวแปร ค่าที่เรานำมาเก็บในตัวแปรอาจจะ เป็นตัวอักษรหรือตัวเลขก็ได ้ตัวแปรในระบบปฏิบัติที่เลียนแบบยูนิกซ์รวมถึงลินุกซ ์จะไม่มีการ

กำหนดประเภทของตัวแปรก่อน โดยเราสามารถที่จะระบุค่าให้กับตัวแปรได้เลย บางกรณีต้องใช้ตัว แปรในเชลล์สคริพต์ เช่น การใช้ตัวแปรรับข้อมูลจากผู้ใช้ทางคีย์บอร์ด

ตัวแปรเกิดขึ้นได้ด้วยการกำหนดค่าพร้อมทั้งตั้งชื่อให้มัน ซึ่งทำได้ 2 วิธี คือการกำหนดจาก ในสคริพต์และจากทางคีย์บอร์ดโดยตรง การกำหนดตัวแปรจากในสคริพต์จะเป็นตัวแปรที่ใช้ภายใน สคริพต์เท่านั้น เมื่อการทำงานของสครพต์จบลงตัวแปรก็จะหายไป ส่วนตัวแปรที่กำหนดทางคีย์บอร์ด โดยตรงนั้น ตัวแปรจะยังคงอยู่ตลอดชั่วเลาที่ผู้ใช้งานยังคงใช้ระบบอยู่ นั่นคือค่าที่อยู่ในตัวแปรจะไม่เปลี่ยนแปลงจนกว่าเราจะเลิกใช้งานระบบ

ตัวอย่างในการกำหนดตัวแปร ดังนี้

กำหนดตัวแปรจากคีย์บอร์ด

$ ob=openbox	ob=openbox
$ echo ob	echo ob
ob	echo $ob
$ echo $ob	$ พมพชอสคริพต์
openbox	ob
$	openbox

กำหนดจากสคริพต์

7.3.1. การตั้งชื่อและกำหนดค่าตัวแปร

การตั้งชื่อให้กับตัวแปรนั้น ตัวแปรจะต้องขึ้นต้นด้วยตัวอักขระหรือเครื่องหมายขีดล่าง (underscore _) เท่านั้น จะขึ้นต้นด้วยตัวเลขหรือเครื่องหมายอื่นๆไม่ได้ ส่วนที่เหลือของชื่อจะเป็น ตัวอักษรอะไรก็ได้ไม่ว่าจะเป็น

ตัวเลข ตัวอักขระ และตัวเครื่องหมายต่างๆ ก็ได้ส่วนการกำหนดค่า ให้กับตัวแปรนั้นจะมีความแตกต่างกันขึ้นอยู่กับประเภทของเชลล์ที่ใช้ โดยแบ่งเป็น 2 วิธี คือ

(1) บอร์นเชลล์และคอร์นเชลล์การกำหนดค่าให้กับตัวแปร สำหรับเชลล์เหล่านี้สามารถ กำหนดโดยใช้เครื่องหมาย = ได้โดยตรง นั่นคือ

varname=value of variable

เช่น book="Linux Basics" จะเป็นการเก็บค่าข้อความสตริง Linux Basics ในตัวแปร book

(2) ซีเชลล์สำหรับซีเชลล์นั้น การกำหนดค่าให้กับตัวแปรจะแตกต่างออกไปจากบอร์เชลล์และคอร์นเชลล์จะใช้คำสั่ง set ในการกำหนดค่าให้กับตัวแปร โดยมีรูปแบบคือ

set varname= value of variable

เช่น set x=50 เป็นการกำหนดให้ตัวแปร x มีค่า 50

สำหรับบอร์นอะเกนเชลล์นั้น สามารถที่จะกำหนดค่าให้กับตัวแปรได้ทั้งสองวิธี ทั้งนี้เพราะใน การพัฒนาบอร์นอะเกนเชลล์นั้น ได้พัฒนาให้มีความสามารถครอบคลุมเชลล์หลายประเภทเข้าด้วยกัน

7.3.2. ประเภทของตัวแปร

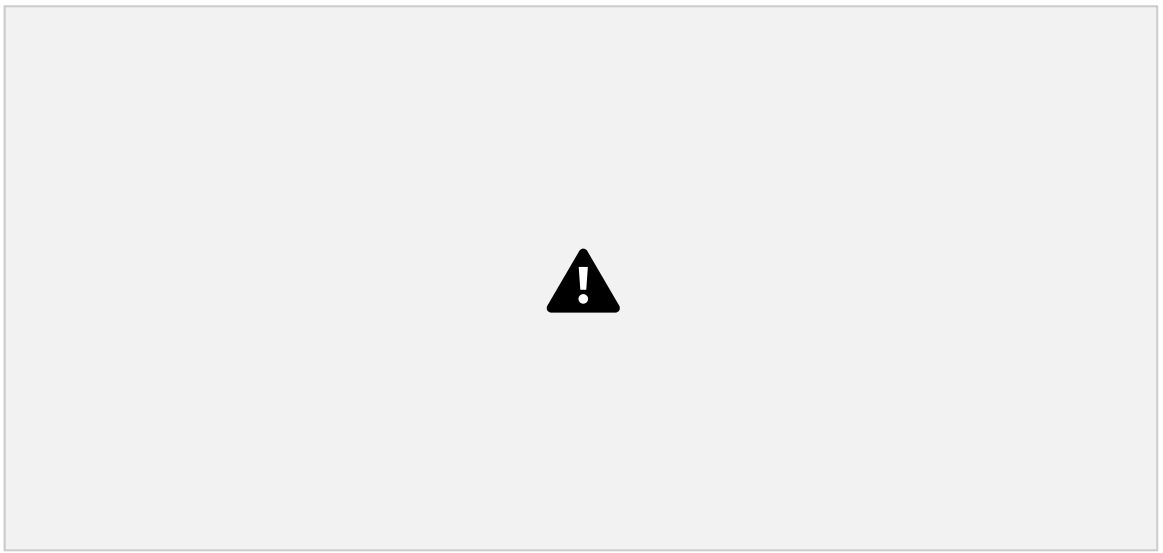
ในระบบปฏิบัติเลียนแบบยูนิกซ์มีตัวแปรอยู่หลายประเภท โดยสามารถแบ่งประเภทของตัว แปรได้เป็น 3 ประเภท ดังต่อไปนี้

- ตัวแปรของผู้ใช้ (User defined variables)
- ตัวแปรระบบ (System predefined variables)

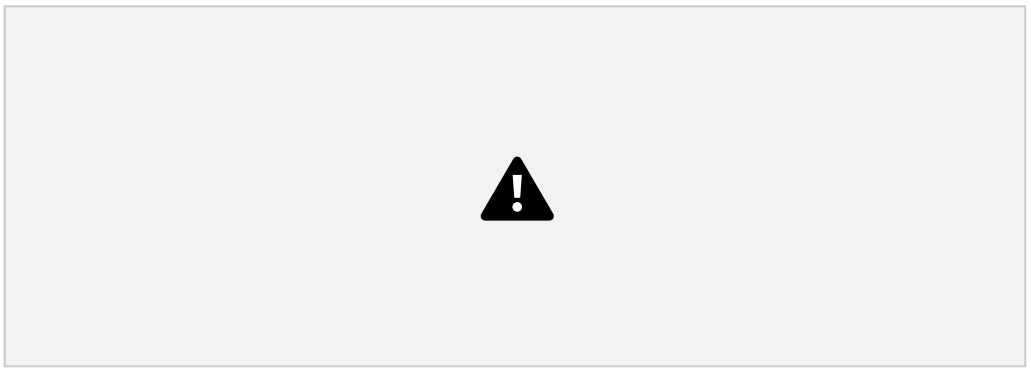
7.3.2.1 ตัวแปรของผู้ใช้

ตัวแปรประเภทนี้จะเป็นตัวแปรที่ผู้ใช้กำหนดขึ้นเอง จึงสามารถกำหนดให้ตัวแปรมีค่าเป็น อะไรก็ได้ ตลอดจนสามารถเปลี่ยนแปลงค่าในตัวแปรได้ตลอดเวลา การกำหนดค่าในตัวแปรจะใช้วิธีที่กล่าวมาแล้วข้างต้น เมื่อต้องการเลิกใช้งานตัวแปรก็สามารถใช้คำสั่ง unset ในการยกเลิกตัวแปรได้ทันที

รูปที่ 7.5 การกำหนดและยกเลิกค่าให้กับตัวแปรของผู้ใช้สำหรับตัวแปรที่เราไม่ต้องการให้มีการเปลี่ยนแปลงค่าในตัวแปร เราจะต้องกำหนดให้ตัวแปร นั้นมีคุณสมบัติอ่านได้อย่างเดียว ด้วยการใช้คำสั่ง readonly ที่หน้าชื่อของตัวแปร คำสั่งนี้มีประโยชน์ในกรณีที่เรามีตัวแปรที่เก็บค่าที่สำคัญไว้และไม่ต้องการให้มีการแก้ไข



รูปที่ 7.6 แสดงการใช้คำสั่ง readonly กำหนดให้ตัวแปรอ่านได้อย่างเดียว



7.3.2.2 ตัวแปรระบบ

ตัวแปรอีกประเภทหนึ่งนอกจากตัวแปรที่ผู้ใช้กำหนดขึ้นมาเอง คือ ตัวแปรระบบ ซึ่งจะเป็นตัว แปรที่ระบบปฏิบัติการยูนิกซ์รวมทั้งลินุกซ์ได้สร้างขึ้นเองตั้งแต่ระบบเริ่มต้นทำงาน ตัวแปรเหล่านี้จะ ถูกกำหนดขึ้นในขณะที่ผู้เข้างานเริ่มเข้าสู่ระบบใหม่ๆ โดยค่าเหล่านี้จะถูกกำหนดไว้ในไฟล์ .bash_profile หรือ .profile และจะถูกเรียกขึ้นมาใช้งานเมื่อมีผู้ใช้งานล็อกอินเข้าสู่ระบบ ตัวแปร เหล่านี้เป็นตัวที่บอกถึงสภาวะแวดล้อมของระบบ เช่น ใครเป็นเจ้าของเชลล์ที่กำลังใช้งานอยู่ เทอร์มินัล ที่ใช้งานอยู่เป็นเทอร์มินัลประเภทไหน เป็นต้น

สำหรับการตรวจสอบว่าในระบบมีตัวแปรอะไรบ้างและตัวแปรนั้นมีการกำหดค่าไว้อย่างไร สำหรับคอร์นเชลล์และบอร์นเชลล์นั้น จะต้องใช้คำสั่ง set ส่วนซีเชลล์จะใช้คำสั่ง env แทน

ตัวแปรระบบจะมีอยู่อย่างมากมาย และในแต่ละระบบก็จะมีจำนวนตัวแปรระบบที่แตกต่างกัน เราจะแบ่งตัวแปรระบบได้เป็น 2 กลุ่ม คือ ตัวแปรเชลล์ (Shell variable) และตัวแปรสภาพแวดล้อม (Environment variable) แต่อย่างไรก็ตามตัวแปรระบบจะมีตัวแปรหลักๆ ที่เหมือนกันและควรรู้จัก

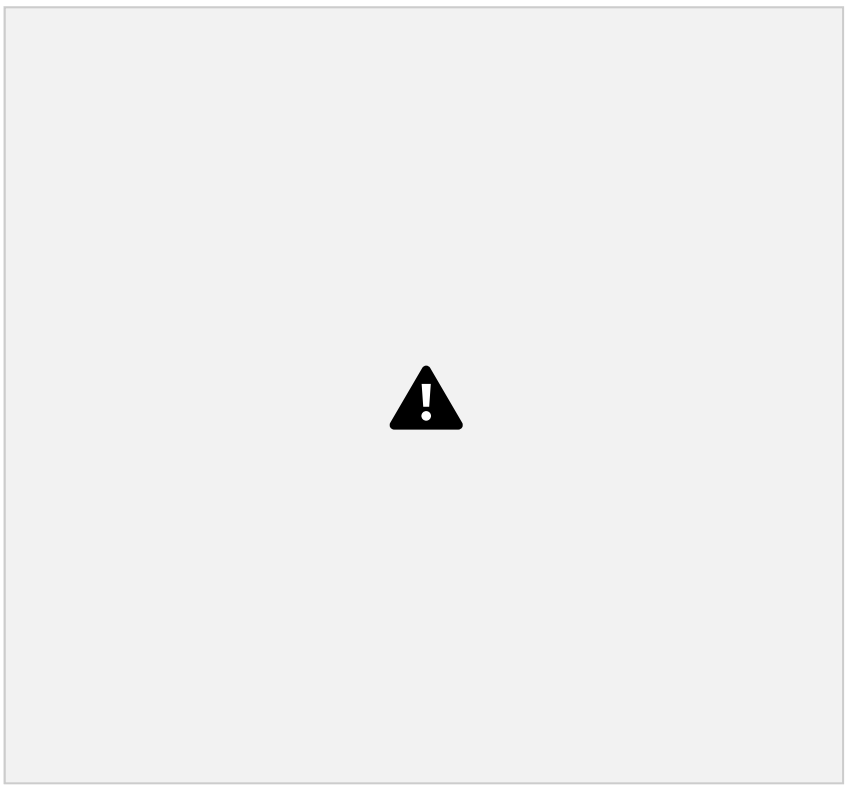
ดังนี้		
	MAIL	ตัวแปรสำหรับแจ้งชื่อไฟล์ที่ใช้เก็บจดหมายอิเล็กทรอนิกส์ของผู้ใช้
	SHELL	ตัวแปรสำหรับเก็บชื่อเชลล์ที่จะถูกเรียกมาใช้ในตอนแรกที่ล็อกอินเข้าสู่ระบบ หรือเป็นเชลล์

		เริ่มต้นนั่นเอง
	TERM	ตัวแปรแจ้งชื่อเทอร์มินัลของระบบว่ากำลังใช้เทอร์มินัลแบบใด
	HOSTNAME	ตัวแปรสำหรับแจ้งชื่อของโฮสต์
	LOGNAME	ตัวแปรสำหรับแจ้งชื่อของผู้ใช้งานที่เป็นผู้ล็อกอินเข้าระบบ
	PATH	ตัวแปรสำหรับเก็บตำแหน่งที่ใช้ในการค้นหาคำสั่ง เมื่อมีการเรียกใช้คำสั่ง หรือโปรแกรม ระบบจะไปค้นหาไฟล์ในตัวแปร PATH โดยตัวแปรนี้สามารถกำหนดได้หลายเส้นทาง โดยมีเครื่องหมาย : คั่นระหว่างเส้นทาง การค้นหาไฟล์จะดำเนินการในทุดเส้นทางที่ระบุไว้และถ้าไม่พบก็จะ แสดงข้อความผิดพลาดออกมาให้ทราบ
	HOME	ตัวแปรสำหรับบอกตำแหน่งเริ่มต้นสำหรับการค้นหา หรือเป็นไดเร็กตอรี่ Home ของผู้ใช้งาน นั่นเอง โดยตัวแปรนี้มักจะถูกกำหนดอยู่ใน /etc/passwd ซึ่งตามปกติเมื่อผู้ใช้ใช้คำสั่ง cd เพียงอย่างเดียว ตำแหน่ง การทำงานก็จะถูกย้ายไปอยู่ในตำแหน่งตามที่ระบุในตัวแปร HOME

PS1 ตัวแปรสำหรับเก็บรูปแบบของเครื่องหมายพร้อมท์หลักของเชลล์ โดยปกติแล้วระบบจะกำหนดให้เป็นเครื่องหมาย # สำหรับผู้ใช้ root และ เครื่องหมาย $ สำหรับผู้ใช้งานทั่วไป

PS2 ตัวแปรสำหรับเก็บรูปแบบของพร้อมท์รองของเชลล์ นั่นคือในกรณีที่มีการต่อบรรทัดจากบรรทัดก่อนหน้านั้น ระบบก็จะนำเอาเครื่องหมาย พร้อมท์รองมาแสดงเพื่อรอรับคำสั่งต่อ ซึ่งโดยปกติแล้วระบบมักจะ กำหนดไว้เป็นเครื่องหมาย >

รูปที่ 7.7 การใช้คำสั่ง set แสดงตัวแปรระบบ



ตัวแปรระบบที่ใช้อ่านอย่างเดียว (Read only shell variables)

ตัวแปรระบบที่อ่านได้อย่างเดียวเป็นตัวแปรที่ถูกกำหนดขึ้นมาโดยระบบ และในการเรียกค่า จากตัวแปรประเภทนี้ในแต่ละครั้งจะได้ค่าของตัวแปรที่ไม่เหมือนเดิม โดยค่าตัวแปรจะขึ้นกับ สภาพแวดล้อมของระบบใน

ขณะนั้น ถึงแม้ว่าตัวแปรประเภทนี้จะถูกกำหนดโดยระบบเหมือนกับตัว แปรระบบ แต่ตัวแปรประเภทนี้เราจะไม่สามารถเปลี่ยนแปลงค่าของมันได ้จะสามารถอ่านค่าได้อย่าง เดียว ซึ่งตัวแปรประเทนี มีดังนี้

เอกสารประกอบการสอน วิชา พื้นฐานลินุกซ์ (3901-2103) ธีระ โชคพระสมบัติ วิทยาลัยเทคนิคชุมพร	
$0	เป็นตัวแปรสำหรับการเก็บชื่อของโปรแกรมหรือสคริพต์ที่ได้ทำการเรียกใช ้งานอยู่
$1...$9	เป็นตัวแปรที่เก็บค่าอาร์กิวเม็นต์ (ค่าที่ถูกส่งผ่านเข้าในโปรแกรมเพื่อนำไปใช ้งาน) ของโปรแกรม ซึ่งถูกกำหนดมากับโปรแกรม
$*	เป็นตัวแปรที่เก็บค่าอาร์กิวเม็นต์ของโปรแกรมทั้งหมด ซึ่งค่าที่เก็บในตัวแปร $1- $9 จะถูกเก็บในตัวแปรนี้ด้วยเช่นกัน
$#	เป็นตัวแปรที่เก็บจำนวนของอาร์กิวเม็นต์ทั้งหมด
$$	เป็นตัวแปรที่เก็บหมายเลขโปรเซสของโปรแกรมที่กำลังเรียกใช้ในปัจจุบัน
$!	เป็นตัวแปรที่เก็บหมายเลขโปรเซสของโปรแกรมหรือคำสั่งที่มีการทำงานใน เบื้องหลังล่าสุด
$?	เป็นตัวแปรที่เก็บค่าที่ส่งคืนกลับมาจากโปรแกรม หรือคำสั่งที่เพิ่งทำงานเสร็จ (Return code) ซึ่งค่านี้จะใช้เพื่อบอกว่าโปรแกรมที่เพิ่งทำงานเสร็จไปนั้น ล่าสุดทำงานสำเร็จหรือมีข้อผิดพลาดใดเกิดขึ้น
$@	เป็นตัวแปรที่มีหน้าที่เช่นเดียวกับตัวแปร $* นั่นคือจะใช้เก็บค่าอาร์กิวเม็นต์ ของโปรแกรมทั้งหมด

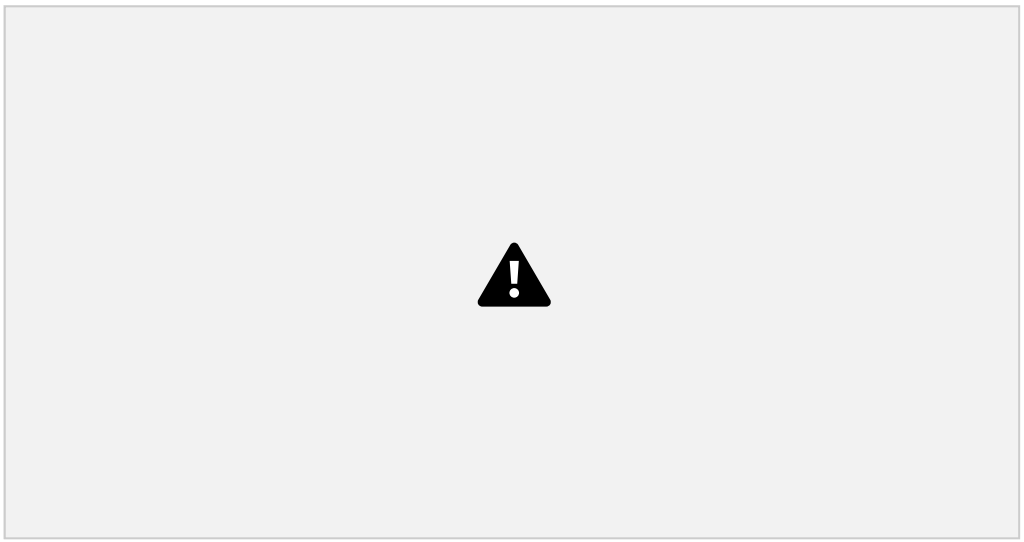
เพื่อให้เกิดความเข้าใจเกี่ยวกับการเก็บค่าของตัวแปรยิ่งขึ้น จึงขอยกตัวอย่างเชลล์สคริพต์ demo_argument ต่อไปนี้

```
echo "The $0 is the name of this script" echo "First argument is $1" echo "Second argument is $2" echo "This script has $# argument" echo "1. Total arguments are $*" echo "2. Total arguments are $@"
```

เราจะเรียกใช้งานสคริพต์นี ด้วยการส่งค่าอาร์กิวเม็นต์ให้กับคำสั่งด้วย ดังนี้

```
# bash demo_argument one two
```

รูปที่ 7.8 การเรียกใช้งานสคริพต์ demo_script



ผลจากการเรียกให้เชลล์สคริพต์ demo_script ทำงาน จะสังเกตได้ว่าตัวแปร $* และตัวแปร $@ นั้นจะให้ผลการทำงานเหมือนกัน แต่ในความเป็นจริงนั้น ตัวแปรทั้งสองมีความแตกต่างกันกันอยู่เล็กน้อย คือ ตัวแปร $* นั้นเมื่อมีการเก็บค่าอาร์กิวเม็นต์หลายๆค่า ตัวแปร $* จะนำเอาค่าของอาร์กิวเม็นต์เหล่านั้นมารวมกันโดยไม่สนใจช่องว่างระหว่างอาร์กิวเม็นต์ แต่สำหรับตัวแปร $@ จะมีจำนวน เท่ากับจำนวนอาร์กิวเม็นต์จริงๆ ทั้งนี้เพราะเมื่อใช้ตัวแปร $@ ระบบจะมองว่าช่องว่างระหว่างอาร์กิว เม็นต์เป็นตัวแยกค่าอาร์กิวเม็นต์ออกจากกัน ดังแสดงให้เห็นความแตกต่างนี้ด้วยเชลล์สคริพต์ต่อไปนี้

```
function test_argument {
        echo " This function has $# arguments"
}
echo " This script has $# arguments"
test_argument "$@"
```

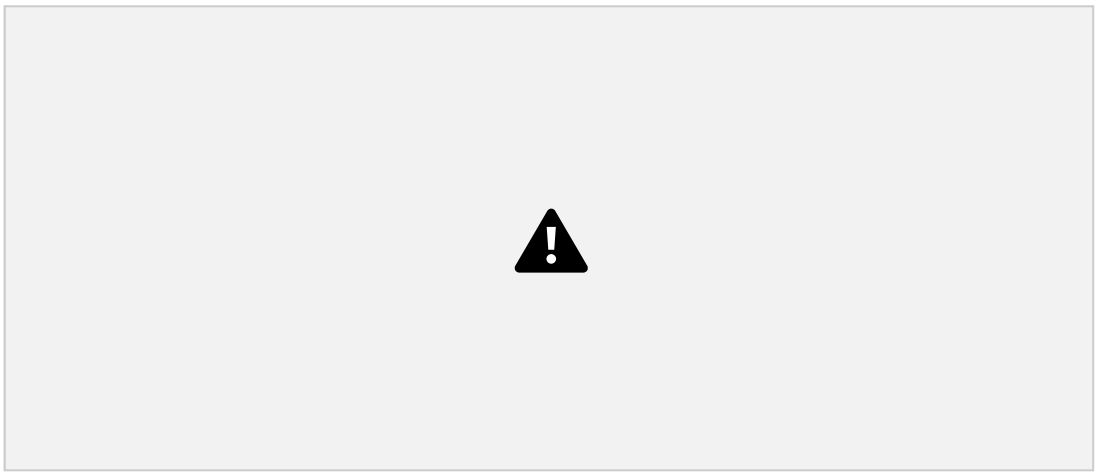
เมื่อเราทำการเรียกไฟล์สคริพต์ให้ทำงาน พร้อมกับส่งค่าอาร์กิวเม็นต์ดังนี้

```
# bash test_argument one two three
```

จากผลการทำงานของเชลล์สคริพต์ จะเห็นว่าจำนวนของอาร์กิวเม็นต์ของเชลล์สคริพต์และ ฟังก์ชั่นจะเท่ากัน คือ 3 ตัว ดังนั้น ตัวแปร $@

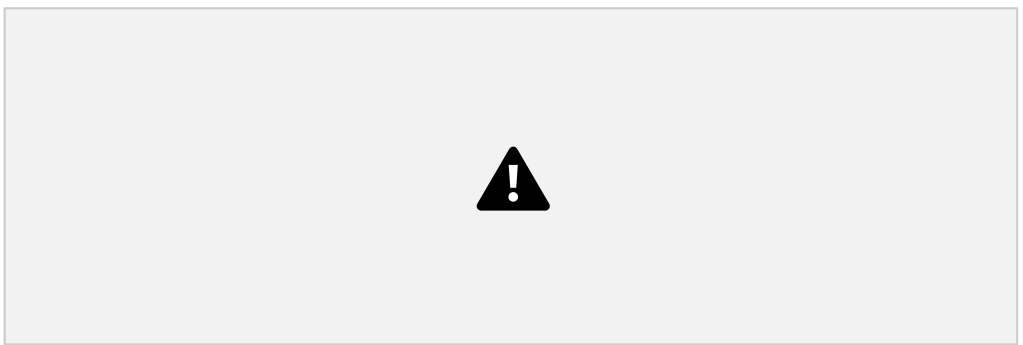
จึงเก็บจำนวนของอาร์กิวเม็นต์ได้ตรงกับจำนวนจริงๆ ของอาร์กิวเม็นต์

รูปที่ 7.9 แสดงการเก็บค่าในตัวแปร $@ สำหรับความแตกต่างกับตัวแปร $* ทำได้ด้วยการแก้ไขไฟล์สคริพต์เสียใหม่ดังต่อไปนี้



```
function test_argument {
        echo " This function has $# arguments" } echo " This script has $# arguments"
test_argument "$*"
```

จากนั้นเรียกใช้งานเชลล์สคริพต์ ดังที่กล่าวมาแล้วข้างต้น จะได้ผลการทำงานดังรูปที่ 7.10



รูปที่ 7.10 แสดงการเก็บค่าในตัวแปร $* จะเห็นว่าจำนวนอาร์กิวเม็นต์ของเชลล์สคริพต์และของฟังก์ชันจะไม่เท่ากัน กล่าวคือ เชลล์สคริพต์จะมีอาร์กิวเม็นต์จำนวนเท่ากับ 3 แต่ฟังก์ชันจะมีอาร์กิวเมนต์จำนวนเท่ากับ 1 เหตุที่เป็นเช่นนี้เพราะ ตัวแปร $* จะไม่สนใจช่องว่างระหว่างอาร์กิวเม็นต์ที่อยู่ในตัวแปร โดยจะถือว่าค่าคงที่อยู่ในตัว แปร $* นี้เป็นข้อมูลชุดเดียวเท่านั้น ในการกำหนดตัวแปรขึ้นมาใช้งานในระบบนั้น ปัญหาหนึ่งที่พบกันก็คือ การ

ที่เราจะเรียกใช้ตัวแปรที่สร้างขึ้นในเชลล์อื่นๆ มาใช้บนเชลล์ที่เราต้องการไม่ได้ทั้งนี้เพราะในการกำหนดตัวแปรขึ้น ในเชลล์ใดๆ นั้น ตัวแปรที่เกิดขึ้นจะมีคุณสมบัติเป็นตัวแปรใช้เฉพาะที่เท่านั้น ถ้าเราต้องการให้สามารถ

ใช้ได้กันทั่วไปในเชลล์อื่นๆ ด้วย เราจะต้องส่งตัวแปรตัวนั้นออกไปจากเชลล์เสียก่อน จากคุณสมบัติที่กล่าวมา เราจึงแบ่งตัวแปรในเชลล์เป็นประเภท ดังนี้

(1) **ตัวแปรกลาง** (Global variable) ตัวแปรประเภทนี้จะมีการถ่ายทอดคุณสมบัติต่างๆ จาก โปรเซสแม่ (Parent process) ไปยังโปรเซสลูก (Child process) ซึ่งเป็นโปรเซสที่ถูกสร้างขึ้นโดย โปรเซสแม่อีกทีหนึ่ง ดังนั้นจากคุณสมบัติของตัวแปรกลางนี้จึงทำให้ขณะที่เราใช้งานเชลล์ลูกอยู่จึง สามารถรียกใช้ค่าจากตัวแปรที่ถูกกำหนดหรือประกาศไว้ในโปรเซสแม่ได้ สำหรับวิธีในการสร้างตัว แปรกลางนั้น จะมีความแตกต่างกันขึ้นอยู่กับเชลล์ที่ใช้งาน โดยสำหรับบอร์นเชลล์และคอร์นเชลล์จะใช้คำสั่ง

```
variable_name=value
```

เพื่อประกาศค่าให้กับตัวแปร จากนั้นจึงใช้คำสั่ง

```
export variable_name
```

เพื่อกำหนดให้มีตัวแปรมีคุณสมบัติเป็นตัวแปรกลาง สำหรับซีเชลล์สามารถสร้างตัวแปลกลางได้ด้วย คำสั่ง

```
setenv variable_name value
```

(2) **ตัวแปรภายใน** (Local variable) ตัวแปรชนิดนี้จะถูกสร้างขึ้นมาเพื่อใช้อยู่ภายในเชลล์เท่านั้น จะไม่มีการถ่ายทอดคุณสมบัติจากโปรเซสแม่ไปยังโปรเซสลูก จึงทำให้ไม่สามารถที่จะเรียกใช้งานตัวแปรในเชลล์อื่นได้ ในคอร์นเชลล์และบอร์นอะเกนเชลล์จะใช้คำสั่งในการสร้างตัวแปร คือ

```
variable_name=value
```

ส่วนซีเชลล์จะใช้คำสั่ง

```
set variable_name=value
```

ในระบบปฏิบัติการลินุกซ์นั้นจะใช้บอร์นอะเกนเชลล์เป็นเชลล์หลัก ดังนั้นวิธีการสร้างตัวแปร ภายในให้เป็นตัวแปรกลางจึงเหมือนกับบอร์นเชลล์และคอร์นเชลล์ คือให้ประกาศตัวแปรขึ้นก่อน แล้วจึงใช้คำสั่ง export เพื่อเปลี่ยนตัวแปรภายในให้กลายเป็นตัวแปรกลางต่อไป เช่น

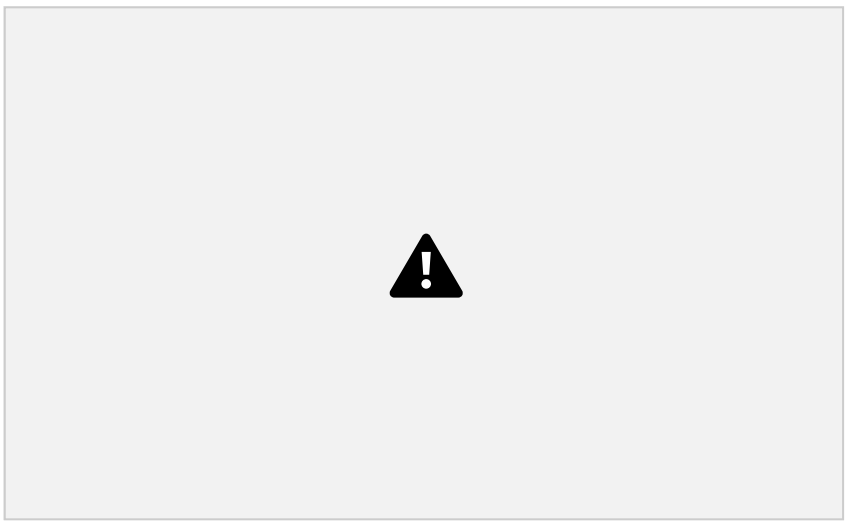
```
book=Linux export book
```

จากขั้นตอนการใช้คำสั่งข้างต้น จะทำให้ตัวแปร book กลายเป็นตัวแปรกลาง นอกจากการใช้คำสั่ง export เพื่อกำหนดตัวแปรกลางแล้ว ยังสามารถทำให้ตัแปรที่อยู่ในเชลล์สคริพต์กลายเป็นตัวแปรกลาง ได้ด้วยการใช้ . (จุด) ในการเรียกไฟล์สคริพต์นั้น ซึ่งจะไม่กล่าวถึงรายละเอียดในที่นี้

(3) **ตัวแปรแบบมีเงื่อนไข** ตัวแปรชนิดนี้มีค่าที่ไม่คงที่แน่นอน ค่าของตัวแปรจะถูกกำหนดตาม เงื่อนไขต่างๆ ตัวแปรชนิดนี้มีอยู่หลายประเภท ทำให้สามารถกำหนดค่าให้กับตัวแปรได้หลายรูปแบบ โดยรูปแบบของตัวแปรแบบมีเงื่อนไขมีดังนี้

${varname:-word} รูปแบบการใช้งานแบบนี้คือ หากตัวแปร varname มีค่าก็จะส่งคืนค่า (return) ของตัวแปร varname กลับไปให้ แต่ถ้าตัวแปร varname ไม่มีค่า (หรือเป็น null) ก็จะส่งค่าที่กำหนดไว้คือ word กลับไปให้ ดังตัวอย่างในรูปที่ 7.11

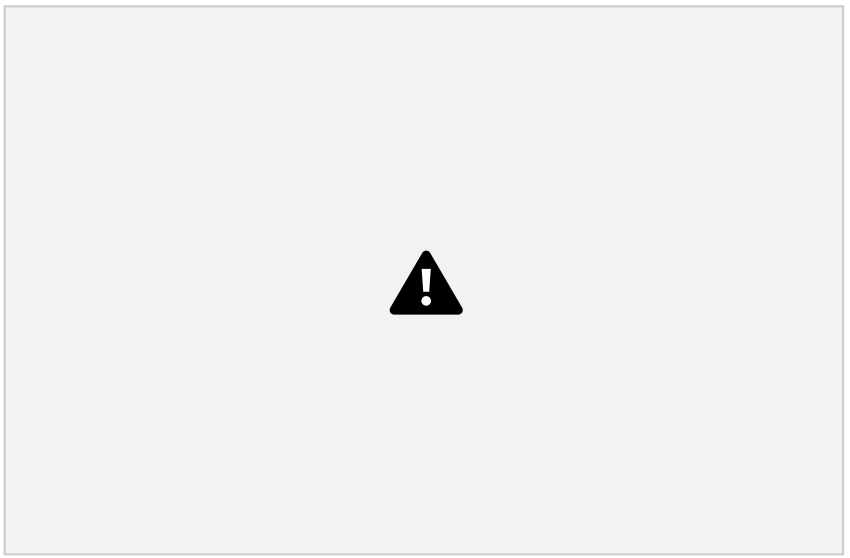
รูปที่ 7.11 ตัวอย่างการใช้ตัวแปร ${varname:-word}



จากตัวอย่างจะเห็นว่าในตอนแรกไม่ได้กำหนดค่าให้กับตัวแปร NAME ดังนั้นจากรูปแบบ คำสั่ง จึงมีการส่งค่า "Peter" ไปให้กับตัวแปร STUDENT เมื่อใช้คำสั่ง echo $STUDENT จึงเห็น เป็นค่า "Peter" แต่ในช่วงที่สองได้ทำการกำหนดค่าให้ NAME มีค่าเป็น "John" แล้ว ตัวแปร STUDENT จึงถูกกำหนดค่าเป็น "John" ตามไปด้วย

${varname:=word} รูปแบบของกำหนดตัวแปรแบบน ี้คือถ้าหากตัวแปร varname มีค่าก็จะส่งค่าของตัวแปร varname กลับไปให้ แต่ถ้าตัวแปร varname ไม่มีค่า ก็จะมีการทำงานสองขั้นตอน คือทำการกำหนดค่า word ให้กับตัวแปร varname และส่งค่า word กลับไปให้ด้วย ดังตัวอย่างในรูปท ี่ 7.12

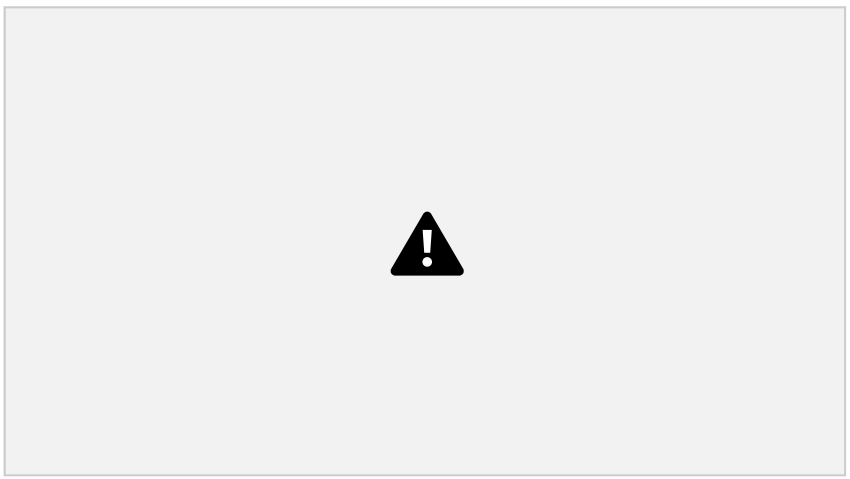
รูปท ี่ 7.12 ตัวอย่างการใช้ตัวแปร ${varname:=word}



จากตัวอย่างจะเห็นว่าหากตัวแปร NAME ไม่มีค่าแล้ว ตัวแปร NAME ก็จะถูกกำหนดค่าให้เป็น "Peter" และรวมทั้งมีการส่งค่า "Peter" ไปให้กับตัวแปร STUDENT ด้วย (ซึ่งจะต่างกับแบบแรกที่ไม่มีการกำหนดค่าให้กับตัวแปร NAME, ยังคงไม่มีค่า) แต่ถ้าหากตัวแปร NAME มีค่าอยู่แล้ว (เป็น "John") ก็จะคืนค่าของตัวแปร NAME ไปให้เหมือนกัน

${varname:?message} วิธีการกำหนดตัวแปรแบบนี้หากตัวแปร varname ไม่มีค่าแล้ว เชลล์จะทำการแจ้งว่ามีข้อผิดพลาดเกิดขึ้น โดยทำการแจ้งข้อความตามที่ได้กำหนดไว้ใน message แต่ถ้าตัวแปร varname มีค่าก็จะส่งค่าของตัวแปร varname ให้ตามปกติ ดังตัวอย่าง

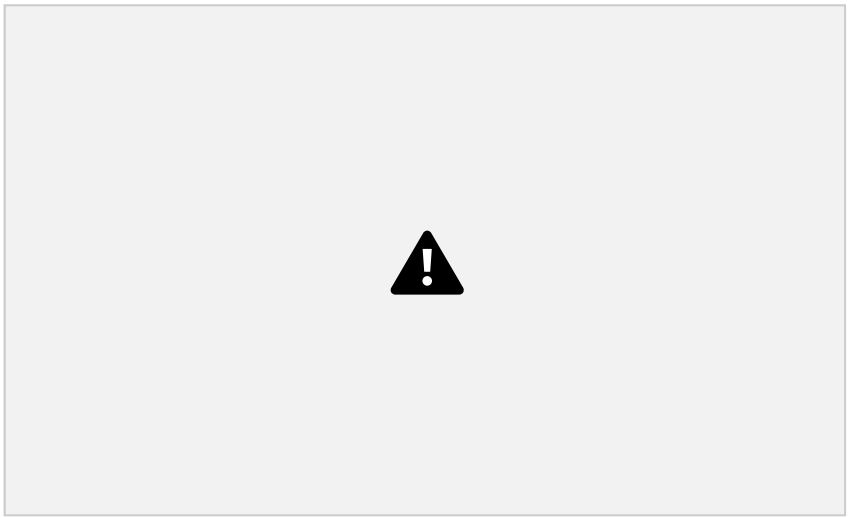
รูปที่ 7.13 ตัวอย่างการใช้ตัวแปร ${varname:?message}



จากตัวอย่างรูปที่ 7.13 ในตอนแรกจะเห็นว่าเมื่อไม่ได้ทำการกำหนดค่าให้กับตัวแปร NAME เชลล์จะทำการแจ้งตัวแปรที่มีข้อผิดพลาดขึ้นมา (ในที่นี้คือ NAME) และตามด้วยข้อความผิดพลาดที่เราได้กำหนดไว้ (ในที่นี้คือ "Not assigned") และทั้งตัวแปร STUDENT และ NAME ก็จะยังคง ไม่ได้มีการกำหนดค่าไว้ตามเดิม (เป็น null) ในช่วงที่สองเมื่อตัวแปร NAME ถูกกำหนดค่าเป็น "John" ก็จะทำการส่งค่า "John" ให้กับตัวแปร STUDENT ด้วย

${varname:+word} วิธีการกำหนดตัวแปรแบบนี้จะแตกต่างจากสามแบบที่กล่าวมาแล้ว ข้างต้น กล่าวคือถ้าตัวแปร varname มีค่าและไม่เป็น null ก็จะส่งค่า word ไปให้แต่ถ้าไม่มีค่า ก็จะส่ง ค่า null ไปให้สิ่งที่แตกต่างจากสามแบบข้างต้นก็คือ มีการส่งค่าไปให้หากตัวแปร varname ถูก กำหนดค่ามา

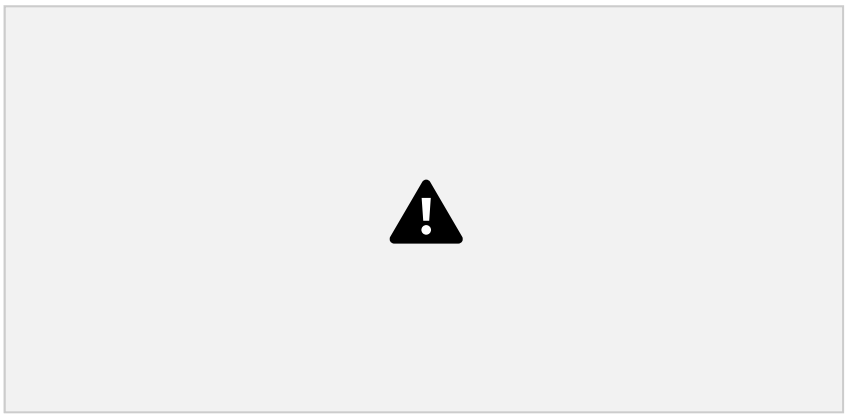
รูปที่ 7.14 ตัวอย่างการใช้ตัวแปร ${varname:+word}



จากตัวอย่างจะเห็นว่าถ้าตัวแปร NAME ไม่มีค่า ตัวแปร STUDENT ก็จะไม่มีค่า (เป็น null) ตามไปด้วย แต่เมื่อมีการกำหนดค่า "John" ให้กับตัวแปร NAME ระบบก็จะมีการส่งค่าของ "exist" ไปให้กับตัวแปร STUDENT

${varname#pattern} รูปแบบการกำหนดตัวแปรแบบนี้เมื่อถูกนำไปใช้งาน ค่าที่ถูกเก็บไว้ในตัวแปร varname จะถูกนำมาเปรียบเทียบกับ pattern ถ้า pattern ที่ระบุมาตรงกับส่วนต้นของค่าใน ตัวแปร varname ระบบจะตัดส่วนที่ตรงกันออกแล้วส่งค่าที่เหลือกลับมา ดังตัวอย่างในรูปที่ 6.15

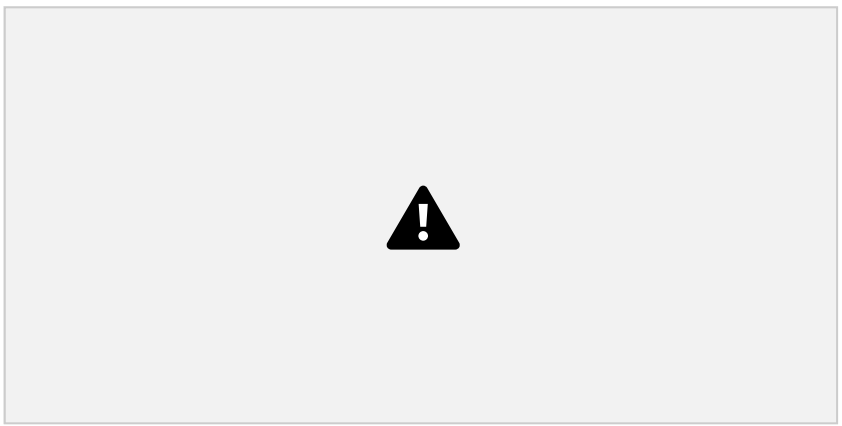
รูปที่ 7.15 ตัวอย่างการใช้ตัวแปร ${varname#pattern}



จากตัวอย่าง ตัวแปร NAME **จะมีค่าคือ** “/HOME/MYBOOK/LINUX BASICS” **ดังนั้น การกำหนดค่าตัวแปร** ${NAME#/*/} **จะหมายถึง** “MYBOOK/LINUX BASICS” **นั่นคือ ระบบ จะนำ** pattern **คือ** /*/ **ไปเปรียบเทียบกับค่าในตัวแปร โดยที่** /*/ **นั้นหมายถึง อะไรก็ตามที่ขึ้นต้นด้วย** / **และลงท้ายด้วย** / **ซึ่งก็คือ** /HOME/ **จึงถูกตัดทิ้งไป แล้วส่งค่าที่เหลือออกมา คือ** MYBOOK/LINUX BASICS

${varname##pattern} รูปแบบของการกำหนดตัวแปรแบบนี้ระบบจะนำเอา pattern ไป เปรียบเทียบกับค่าส่วนต้นของค่าในตัวแปร varname เช่นเดียวกับที่กล่าวผ่านมา แต่รูปแบบนี้ระบบจะ ตัดออกให้มากที่สุดก่อนที่จะส่งค่ากลับมา ดังตัวอย่างในรูปที่ 7.16

รูปที่ 7.16 ตัวอย่างการใช้ตัวแปร ${varname##pattern}

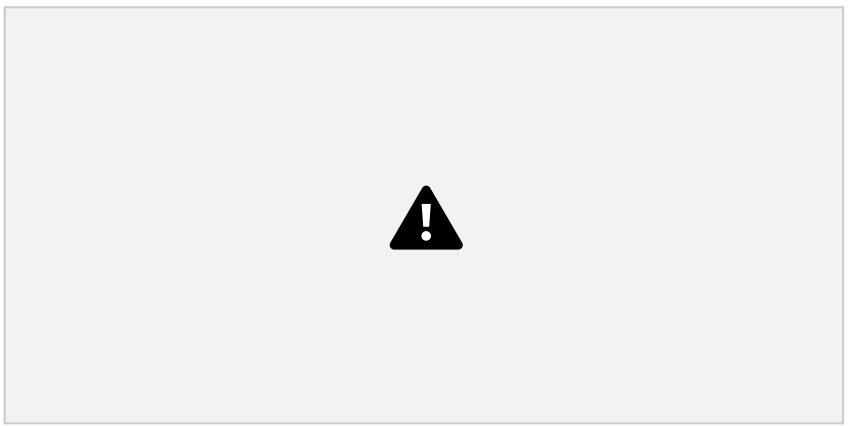


จากตัวอย่างในรูปที่ 7.16 ตัวแปร NAME จะมีค่าคือ "/HOME/MYBOOK/LINUX BASICS" ดังนั้นการกำหนดค่าตัวแปร ${NAME##/*/} จะหมายถึง "LINUX BASICS" นั่นคือ ระบบจะนำ pattern คือ /*/ ไปเปรียบเทียบกับค่าส่วนต้นในตัวแปรที่ยาวที่สุด โดยที่ /*/ นั้นหมายถึง อะไรก็ตามที่ขึ้นต้นด้วย / และลงท้ายด้วย / ซึ่งก็คือ /HOME/MYBOOK/ จะถูกตัดทิ้งไป แล้วส่งค่าที่เหลือออกมา คือ LINUX BASICS

${varname%pattern} รูปแบบของการกำหนดตัวแปรแบบนี้จะคล้ายกับแบบ ${varname#pattern} แต่แทนที่ระบบจะนำเอา pattern ไปเปรียบเทียบกับค่าส่วนต้นของค่าในตัว แปร varname เช่นเดียวกับที่กล่าวผ่านมา กลับนำไปเปรียบเทียบค่าในส่วนท้ายของตัวแปร varname แทน แล้วตัดส่วนที่ตรงกัน

ออก ก่อนที่จะส่งค่าที่เหลือกลับมา ดังตัวอย่างในรูปที่ 7.17

รูปที่ 7.17 ตัวอย่างการใช้ตัวแปร ${varname%pattern}

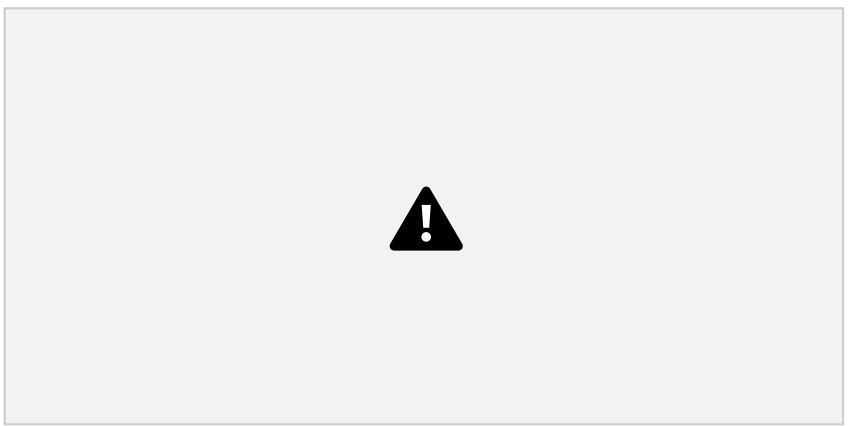


ในรูปที่ 7.17 ตัวแปร NAME จะถูกกำหนดให้มีค่าคือ "/HOME/MYBOOK/LINUX BASICS.DOC.ZIP" ดังนั้นจากการกำหนดค่าตัวแปรด้วยรูปแบบ ${NAME%.*} ระบบจะนำ pattern คือ .* ไปเปรียบเทียบกับค่าในส่วนท้ายของค่าในตัวแปร NAME โดยที่ .* นั้นจะหมายถึง

อะไรก็ตามที่ลงท้ายด้วย . ดังนั้นในที่นี้จึงหมายถึง .DOC.ZIP เมื่อนำ pattern มาเปรียบเทียบจึงทำให้ค่าที่ถูกส่งกลับออกมาเหลือแค่ /HOME/MYBOOK/LINUX BASICS.DOC

${varname%%pattern} รูปแบบของการกำหนดตัวแปรแบบนี้จะคล้ายกับแบบ ${varname%pattern} แต่ต่างกันที่ระบบจะหาส่วนที่ตรงกันให้มากที่สุดแล้วตัดทิ้ง จากนั้นจึงส่งค่าที่เหลือกลับออกมา ดังตัวอย่างในรูปที่ 7.18

รูปที่ 7.18 ตัวอย่างการใช้ตัวแปร ${varname%%pattern}



ในรูปที่ 7.18 จะได้กำหนดค่าให้กับตัวแปร NAME คือ "/HOME/MYBOOK/LINUX BASICS.DOC.ZIP" ดังนั้นการกำหนดค่าตัวแปร ${NAME%%.*} ระบบจะนำ pattern คือ .* ไป เปรียบเทียบกับค่าส่วนท้ายของค่าในตัวแปร NAME โดย .* จะหมายถึงอะไรก็ตามที่ลงท้ายด้วย . ใน ที่นี้จึงหมายถึง .DOC.ZIP จากนั้นระบบจะตัดค่าตาม pattern จึงทำให้ค่าที่ถูกส่งกลับออกมาเหลือแค่ /HOME/MYBOOK/LINUX BASICS

รูปแบบการกำหนดตัวแปรแบบมีเงื่อนไขที่กล่าวมาข้างต้น สามารถนำมาใช้ในการตรวจสอบ และจัดการกับตัวแปรที่ไม่ได้มีการกำหนดค่าตาม

ที่ต้องการได้

7.4. การควบคุมลักษณะการแสดงผล

เนื้อหาในหัวข้อนี้จะได้กล่าวถึงการควบคุมการแสดงผลบนหน้าจอให้มีลักษณะที่แตกต่างจาก ปกติซึ่งมักจะใช้คำสั่ง echo ในการแสดงผลหน้าจอภาพ นอกจากการใช้คำสั่ง echo ตามปกติแล้ว เรา ยังสามารถใช้ echo ร่วมกับคีย์พิเศษ เพื่อใช้ในการควบคุมการแสดงบนหน้าจอได้ด้วย

การใช้คีย์พิเศษหมายถึง การใช้รหัสเอ็สเค็ป (escape) เป็นรหัสพิเศษไปทำการควบคุมลักษณะ การแสดงผล โดยรหัสพิเศษจะต้องขึ้นต้นด้วยรหัสของ escape (\033) ซึ่งจะเป็นรหัสชุด ดังนั้นจึง เรียกรหัสชุดว่า เอ็สเค็ปซีเควันซ์ (escape sequence) การใช้รหัสพิเศษกับการแสดงผล ทำให้เรา สามารถบังคับควบคุมให้ระบบทำการแสดงผลตัวอักษรแบบกระพริบ กลับขาวเป็นดำ หรือมีเส้นใต้ตัวอักษร เป็นต้น รหัสของคีย์พิเศษต่างๆ มีดังนี้

- \033[line;colH ตำแหน่งของเคอร์เซอร์ prompt จะไปอยู่บนบรรทัดที่ระบุด้วย line และคอลัมน์ระบุด้วย col
- \033[2J ลบหน้าจอแสดงผล (คล้ายกับคำสั่ง clear)
- \033[4m เข้าสู่โหมดขีดเส้นใต้ตัวอักษรที่พิมพ์ต่อจากรหัสชุดนี้จะเป็นตัวอักษรที่มีการขีดเส้นใต้ทั้งหมด
- \033[5m เข้าสู่โหมดตัวอักษรกระพริบ (blink)
- \033[7m เข้าสู่โหมดตัวอักษรกลับขาวดำ (reverse) โดยจะเปลี่ยนสีตัวอักษรสีดำ ให้เป็นสีขาว และตัวอักษรสีขาวให้เป็นสีดำ
- \033[m กลับสู่โหมดปกติเพื่อยกเลิกการใช้รหัสพิเศษทั้งหมด

สำหรับรหัสเอ็สเค็ป เหล่านี้สามารถสร้างได้ด้วยการกดปุ่ม <Ctrl> + <v> + <Esc> ซึ่งเป็น รหัส \033 จากนั้นจึงกดปุ่ม <[> และตัวอักษรอื่นตามลำดับ เช่น เราต้องการส่งรหัสชุดสำหรับเข้าสู่โหมดตัวอักษรกระพริบ จะต้องใช้รหัส \033[5m ก็จะต้องกดปุ่ม <Ctrl + v> จากนั้นปล่อยปุ่มทั้งสอง และกดปุ่ม <Esc> จะปรากฏบนหน้าจอเป็น ^[จากนั้นจึงกดปุ่ม <[>, <5>, <m> ตามลำดับ ซึ่งจะ เห็นรหัสคำสั่งบนหน้าจอเป็น ^[[5m สำหรับรหัสคำสั่งอื่นๆ ก็มีการใช้คีย์คำสั่ง ดังนี้

- \033[line;colH ^[[x;yH
- \033[2J ^[[2j
- \033[4m ^[[4m
- \033[5m ^[[5m
- \033[7m ^[[7m
- \033[m ^[[m

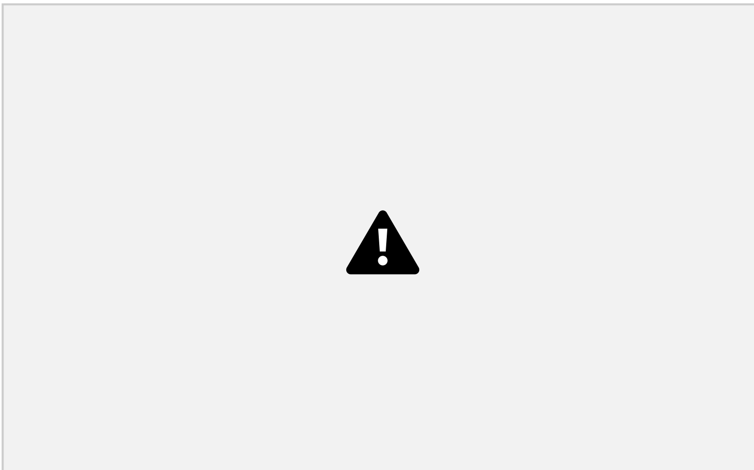
เพื่อให้เข้าใจการใช้รหัสเอ็สเค็ปยิ่งขึ้น ให้พิจารณาเชลล์สคริพต์ต่อไปนี้

```
clear echo "^[[07;20H+--------------------------------------+" echo "^[[07;38H^[[7m
MENU ^[[m" echo "^[[08;20H| .....|" echo "^[[09;20H| 1. Option 1.
.....|" echo "^[[10;20H| 2. Option 2. .....|" echo "^[[11;20H| 3. Option 3.
.....|" echo "^[[12;20H| 4. Option 4. .....|" echo "^[[13;20H| 5. Exit.
.....|" echo "^[[14;20H| .....|" echo
"^[[15;20H+--------------------------------------+"
```

echo "^[[16;20H Please select your choice :" read choice

หลังจากที่ทำการบันทึกเชลล์สคริพต์นี้เป็นไฟล์แล้ว ทำการเรียกไฟล์เชลล์สคริพต์นี้จะได้ผลลัพธ์ตามรูปที่ 7.19 ข้างล่าง

รูปที่ 7.19 การใช้เชลล์สคริพต์ควบคุมการแสดงผล



จะเห็นว่าเชลล์สคริพต์สามารถที่จะนำมาใช้ในการควบคุมการแสดงผลบนหน้าจอ และเมื่อ นำมาใช้ร่วมกับคำสั่งของลินุกซ์จะทำให้สามารถใช้เชลล์สคริพต์ในการเลือกการตัดสินใจการทำงานได้ดังตัวอย่างในรูปที่ 7.19 ที่ผ่านมา

ㄴ恕□□ 踞賰婟ㄱ Ď㈎ ㄴ□鸗

㈎□㈎□縶□盱□ ┌┌□□□□□嵲〰ㄎ□ □㈎Ď□

醻ㄎ□□鸗㈎Ď㈎ ㄴ徛Ďอ腱Ôƒ Ў乞D̲ b □昫椀嚇

愀琀开猀搀嚇椀瀀琀 縶ㄱอ盱□ ⚠ ㄎ□Ôอ㈎

□ ┌□㈎□

ㄴ恕□□ 踞賰婟 Ď㈎ ㄴ□鸗㈎□㈎□Ô□□盱□ ┌┌□ƒ Ў乞D̲ b□縶

□ƒ丂≁盯□ㄱㄱ□

ㄴ怨□□ 踝劈媚ㄱ止 裔ƒ△□Ď㈜ㄴ盯 ㄴ劈▌Ď繴

□ᏀᏯ̃A□□Ď□□踝丂□ □㈜Ď⅛ь丂✎□■□㈜□□『盯→謗□✎□㈜Ď㈜ㄴ繴□ᏀᏯ̃A□□Ď□□踝丂□繴□盯□ㄱㄱ□ƒŸˊ-乏Ḏ⅛ь □□『□鷺㈜繴→ᏀᏯ□ㄴ『→■丂·ΔĎ㈜ㄴ盯Ô劈▌□ ㄴ→ 丂ƒŸ鷺㈜ƒ踝丂□裔ㄱ『▌丂□□鷺㈜繴→ᏀᏯƒŸˊ-乏Ḏ⅛ь□□醐丂□·Δ怨□□㈜▌▌踝乏□Ô醐嘲□ 繴□Ď㈜ㄴ□鷺㈜□㈜□Ô□□篥□ ㄴ裔Ď ㄴh□『盯✎ᏀᏯ□腥ㄏ□Ⴓ怨Ď盯 ㄴ劈▌Ď□鷺㈜□㈜□繴□□□✎□⅛ь醐丂□裔⅛ь□□踝嫂□ ̃A□□Ď□□丂□□□Ď ㄴ『□踝丂□□≁≁ ㄱ謗□Ď 笀 狻栀攀氈氈 振褽滑滑愀滂搀狻 紀 盯 ㄴ㈜□乏▌ㄏ盯Ô劈▌□̃A□□Ď□□丂 腥✎ᏀᏯ□踝劈⅛ьᏀᏯ□ƒŸˊ-乏Ḏ⅛ь □盯Ḏ踝嫂□繴→ ᏀᏯƒ㈜ㄏ㈜ ㄴⴚⴚ怨Ď盯 ㄴ劈▌Ď□㈜Ď篥□ ㄴ裔Ď ㄴh→ ㄱ丂Ď腥·Δ ᏀᏯ

ㄴ怨□□ 踝劈媚ㄱ劇 Ď㈜ㄴĎ鷺㈜→□·Δ裔 ㄱ『▌Ď盯 ㄱ乏ĎŸ□㈜繴→ᏀᏯĎ丂≁⅛ь丂✎裔□ ㄴÔ□□□ᏀᏯ怨繴□ ᏀᏯƒ鷺㈜→ˊ-丂≁⅛ь丂✎裔□ ㄴ□踝劈盯 ㄴ㈜腥ㄏ□⅛ьᏀᏯ□□Ď㈜ ㄴ繴→ᏀᏯㄏ劈Ď㈜ㄴ盯□ ㄱ踝劈▌□裔□ ㄱ□Ÿ□㈜繴□⅛ь丂✎裔□ ㄴ- 盯 ㄴ㈜□『⅛ьᏀᏯ□□Ď鷺㈜→□·Δ繴→ᏀᏯ⅛ь丂✎裔□ ㄴ- □醐丂□h劈Ÿ嵍ᛁƒh≁丂⅛ь乏□□㈜□腥·ΔᏀᏯ□▌□㈜□盯·Δ劈▌✎ ·ΔᏀᏯ✎▌Ď㈜ㄴ繴□ᏀᏯŸ鷺㈜ƒ踝丂□ 嫲攀愀搀褽滂氈礵□踝劈→□ᏀᏯ㈜□踝嫂□Ô□□⅛ь丂✎裔□ ㄴ Ÿ鷺㈜ƒ踝丂□□醐劈ㄏ劈□ ㄴ『篥▌□□ □繴

□Ď ᅳᄼ篣□踒篣盯 ᅳ㈎ℎ篣ᇲᑲ丂✎裿□ ᅳ☐踒篣盯Ď謗〰Ў□㈎□踒

篣ɟ鸞㈎Ў丂ᆗ腿✎ᡴ裿 ┌『腿ℎ☐ᇲᑲᡴ▯□Ď㈎ ᅳ繄¬ᡴℎ篣Ď㈎

ᅳ裿Ďᡴ腿Ô

ᅳ㤁□□ 踒篣婟┌嗥 裿ɟ·∆□Ď㈎ ᅳ繄□ᡴЎ鸞㈎ɟ踒丂□ 婡攀愀挽磬渀甀磪 Ď
鸞㈎¬▯·∆繄¬ᡴᇲᑲ丂✎裿□ ᅳ☐☐㈎□腿·∆ᡴ▯▮☐㈎□盯·∆篣▮✎

ᅳ㤁□□ 踒篣婟┌婟 Ď㈎ ᅳ繄□ᡴЎ鸞㈎ɟ踒丂□ 渺攀琀 裿ɟ·∆□ᇲᑲ丂✎裿
□ ᅳ ᅳ『〰〰

ᅳ㤁□□ 踒篣婟┌蟲 Ď㈎ ᅳ盯 ᅳ篣▮Ď繄□ᡴ□㈎□ɟЎᅳ乫D̲ᇲᑲ □挽攀渀磬开渺振
婡椀瀀琀

ᅳ㤁□□ 踒篣婟┌㤀 裿ɟ·∆□Ď㈎ ᅳ盯Ď謗〰Ў□㈎繄

□ᇲᑲ丂✎裿□ ᅳ␀盈 ɟ鸞㈎¬ᅳ丂〰Ў✎㈎ℎ裿ᇲᑲĎᇲᑲ▯㈎□Ď丂〰ᇲᑲ丂✎

裿□ ᅳ␀⊙ □鸞㈎腿·∆ᡴ·∆ᡴ✎▮Ď㈎ ᅳ裿Ďᡴ腿Ô腿Ȁ ┌☐ɟЎᅳ乫D̲ᇲᑲ▯

盯ɟ篣▮繄¬ℎ □·∆丂□ᇲᑲ▯▯腿□□ 醏篣

ᅳ㤁□□ 踒篣婟┌▯▯ ᇲᑲ丂✎☐▪☐㈎□Ď㈎ ᅳ繄□ᡴᇲᑲ丂✎裿□ ᅳ␀笀瘀愀婡渀
愀渀攀揀▯明磬婡挽紀

ᅳ㤁□□ 踒篣婟┌▯㈀ ᇲᑲ丂✎☐▪☐㈎□Ď㈎ ᅳ繄
□ᡴᇲᑲ丂✎裿□ ᅳ␀笀瘀愀婡渀愀渀攀揀淺明磬婡挽紀

ᅳ㤁□□ 踒篣婟┌▯㌀ ᇲᑲ丂✎☐▪☐㈎□Ď㈎ ᅳ繄□ᡴᇲᑲ丂✎裿□ ᅳ␀笀瘀愀婡
渀愀渀攀揀璡渀攀渺渺愀最攀紀

ᅳ㤁□□ 踒篣婟┌▯址 ᇲᑲ丂✎☐▪☐㈎□Ď㈎ ᅳ繄□ᡴᇲᑲ丂✎裿□ ᅳ␀笀瘀愀婡
渀愀渀攀揀⃡明磬婡挽紀

ᅳ㤁□□ 踒篣婟┌▯剽 ᇲᑲ丂✎☐▪☐㈎□Ď㈎ ᅳ繄□ᡴᇲᑲ丂✎裿□ ᅳ␀笀瘀愀婡
渀愀渀攀⌀瀀愀琀琀攀婡渀紀

ᅳ㤁□□ 踒篣婟┌▯嗥 ᇲᑲ丂✎☐▪☐㈎□Ď㈎ ᅳ繄□ᡴᇲᑲ丂✎裿□ ᅳ␀笀瘀愀婡
渀愀渀攀⌀⌀瀀愀琀琀攀婡渀紀

ᅳ㤁□□ 踒篣婟┌▯婟 ᇲᑲ丂✎☐▪☐㈎□Ď㈎ ᅳ繄□ᡴᇲᑲ丂✎裿□ ᅳ␀笀瘀愀婡

湤愀洀攀—瀀愀琀琀攀爀渀湤紀

ᑊ꜀恕□□ 踵艻媩ᒥ瞐ᐹᔪᑲ ᑯ ᘔ ✎ ☐ ∎ ☐ ㈎ □ Ď ㈎ ᑊ꜀縶 □ ᏼᏽᑲ ᑯ ᘔ ✎ 裿 □ ᑊ꜀ ᴺᵁᴸ 笁瘀愀爀

湤愀洀攀——瀀愀琀琀攀爀渀湤紀

ᑊ꜀恕□□ 踵艻媩ᒥ瞐ᐹᔪ Ď ㈎ ᑊ꜀縶 □ ᏼᏽ盯 □ ┌ ┌ ☐ ʄ Ў ᑊ꜀ 乫 D̲ ᑲ ᑯ □ Ў ✎ ∽ Ў 嶕ƞ Ď ㈎ ᑊ꜀

裿 ʄ ∆ □ □ ┌