# บทที่ 5 การทำงานในลินุกซ์

## จุดประสงค์เชิงพฤติกรรม

1 เพื่อให้สามารถอธิบายลักษณะของอินพุตและเอาต์พุตของลินุกซ์ได้
2 เพื่อให้สามารถอธิบายรูปแบบของรีไดเร็กชันในคำสั่งลินุกซ์ได้
3 เพื่อให้สามารถใช้รีไดเร็กชันกับคำสั่งลินุกซ์ได้
4 เพื่อให้สามารถใช้ไปป์ไลน์กับคำสั่งของลินุกซ์ได้
5 เพื่อให้สามารถใช้ซีเคว้นซ์กับคำสั่งลินุกซ์ได้
6 เพื่อให้สามารถใช้กรุ๊ปกับคำสั่งลินุกซ์ได้
7 เพื่อให้สามารถเปรียบเทียบการใช้ซีเคว้นซ์กับกรุ๊ปได้
8 เพื่อให้สามารถประยุกต์ใช้งานอินพุตและเอาต์พุตได้

เนื้อหาในบทนี้จะกล่าวถึงความสามารถของลินุกซ์หรือระบบอื่นๆที่มีพื้นฐานการทำงานของ ยูนิกซ์ที่เราสามารถใช้งานเพื่อให้ทำงานได้อย่างมีประสิทธิภาพมากขึ้น

## 5.1. อินพุตและเอาต์พุต

ในระบบปฏิบัติการลินุกซ์หรือยูนิกซ์นั้น เมื่อคำสั่งใดๆ ถูกสั่งให้ทำงาน ลินุกซ์จะเปิดไฟล์ขึ้นมาจำนวน 3 ไฟล์เพื่อใช้ทำงานร่วมกับคำสั่งและโปรเซสที่เกิดขึ้น โดยไฟล์ที่เกิดขึ้นในระบบจะแบ่ง ออกเป็น 3 ชนิดด้วยกัน ดังนี้

(1) **อินพุตมาตรฐาน** (Standard Input: STDIN) จะถูกกำหนดให้เป็นคีย์บอร์ดเป็นอินพุต มาตรฐานที่จะคอยรับข้อมูลต่างๆ เพื่อเข้าไปใช้ในการทำงาน จะแทนไฟล์ประเภทนี้ด้วย
(2) **เอาต์พุตมาตรฐาน** (Standard Output: STDOUT) โดยเบื้องต้นนั้น อุปกรณ์เอาต์พุตที่เป็นมาตรฐานคือ จอภาพ (Monitor)

**(3) เอาต์พุตแสดงความผิดพลาดมาตรฐาน** (Standard Error Output: STDERR) **เป็นเอาต์พุตที่ใช้สำหรับการแจ้งข้อความความผิดพลาดที่เกิดขึ้นในระบบขณะใช้งานออกมาทางจอภาพ**



รูปที่ 5.1 แสดงข้อความแสดงความผิดพลาดจากการใช้คำสั่ง อินพุตและเอาต์พุตที่มีการกำหนดเป็นมาตรฐานเบื้องต้นนั้นสามารถเปลี่ยนแปลงได้ เช่น ถ้า ต้องการเปลี่ยนทิศทางของอินพุตหรือเอาต์พุตให้รับจากไฟล์หรือส่งไปให้ไฟล์จะใช้วิธีการของรีไดเร็ก ชัน (Redirection) หรือถ้าต้องการเปลี่ยนเอาต์พุตที่ได้จากคำสั่งหนึ่งให้ไปเป็นอินพุตของอีกคำสั่ง หนึ่งจะใช้วิธีการของไปป์ไลน์ (Pipeline) และนอกจากนี้ก็ยังมีความสามารถอื่น ๆ ที่เป็นเช่นเดียวกับ ยูนิกซ์อีก เช่น การทำงานในเบื้องหลัง (Background Job) ซับเชลล์ (Subshell) และซ๊อคเก็ต (Socket) ที่จะได้กล่าวถึงในลำดับต่อไป

## 5.2. รีไดเร็คชัน (Redirection)

ปกติแล้วอินพุตจะถูกรับผ่านเข้ามาทางคีย์บอร์ด และผลลัพธ์ที่ได้จะแสดงออกทางจอภาพ แต่อย่างไรก็ตามเราสามารถเปลี่ยนทิศทางของอินพุตให้รับจากไฟล์แทนคีย์บอร์ดและเอาต์พุตให้ส่งไปเก็บ ในไฟล์แทนที่จะแสดงออกทางจอภาพได้ การเปลี่ยนทิศทางหรือรีไดเร็กชันจะใช้เครื่องหมายมากกว่า > หรือน้อยกว่า < เป็นสัญลักษณ์ในการกำหนดทิศทางของข้อมูล โดยจะใช้สัญลักษณ์ > แทนการส่ง ข้อมูลออก และใช้สัญลักษณ์ < แทนการรับข้อมูลเข้ามา ในการใช้งานสัญลักษณ์ทั้งสอง บางครั้งจะมีการใช้ร่วมกันในคำสั่งเดียวกัน ซึ่งมีรูปแบบของการรีไดเร็กชันแบบต่าง ๆ

ดังนี้

| รูปแบบ | คำอธิบาย |
|---|---|
| > File | จะเป็นการกำหนดให้ส่งเอาต์พุตลงไฟล์ |
| < File | จะเป็นการกำหนดให้รับอินพุตจากไฟล์ |
| >> File | จะเป็นการกำหนดให้ส่งเอาต์พุตที่ได้ไปเก็บต่อท้ายจากข้อมูลเดิมในไฟล์ที่กำหนด ในกรณีที่ไฟล์นั้นมีอยู่แล้ว |
| << Label | จะเป็นการกำหนดว่าข้อความหลังสัญลักษณ์จะเป็นอินพุต และจะหยุด รับอินพุตก็ต่อเมื่อเจอสัญลักษณ์นี้อีกครั้ง |
| n> File | จะเป็นการกำหนดให้ส่งชนิดของอินพุต – เอาต์พุต (n) ลงไฟล์ |
| n< File | จะเป็นการกำหนดให้รับเอาไฟล์เป็นชนิดของอินพุต–เอาต์พุต (n) ที กำหนด |



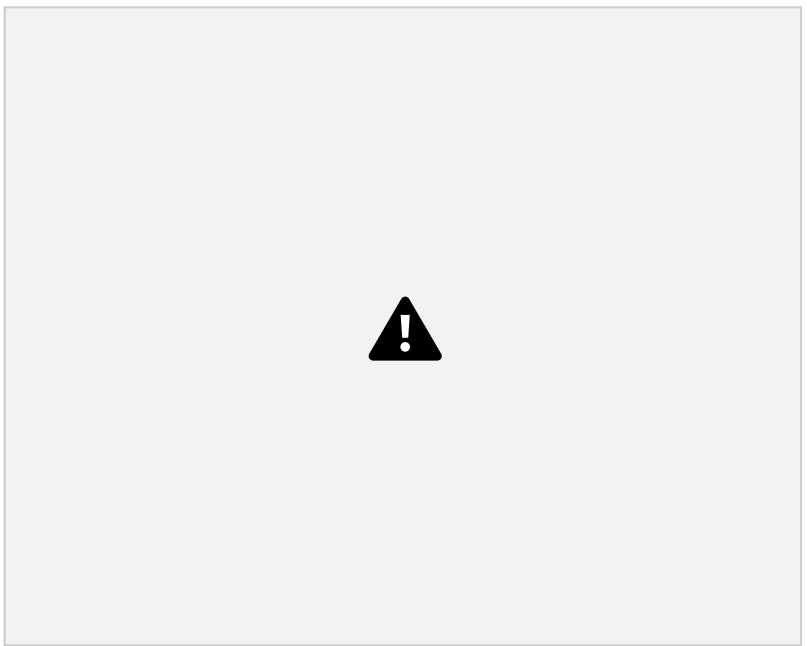
| | |
|---|---|
| >&n | จะเป็นการกำหนดให้สำเนาข้อมูลที่แสดงบนจอภาพไปเก็บในชนิดของ อินพุต–เอาต์พุต ( n) ที่กำหนด |
| <&n | จะเป็นการกำหนดให้สำเนาข้อมูลจากชนิดของอินพุต – เอาต์พุต ( n) ที กำหนดไปเป็นอินพุต |
| &> File | จะเป็นการกำหนดให้ส่งผลลัพธ์และข้อความแสดงความผิดพลาดลงไฟล์ |

ตัวอย่างของรีไดเร็กชันแต่ละรูปแบบ

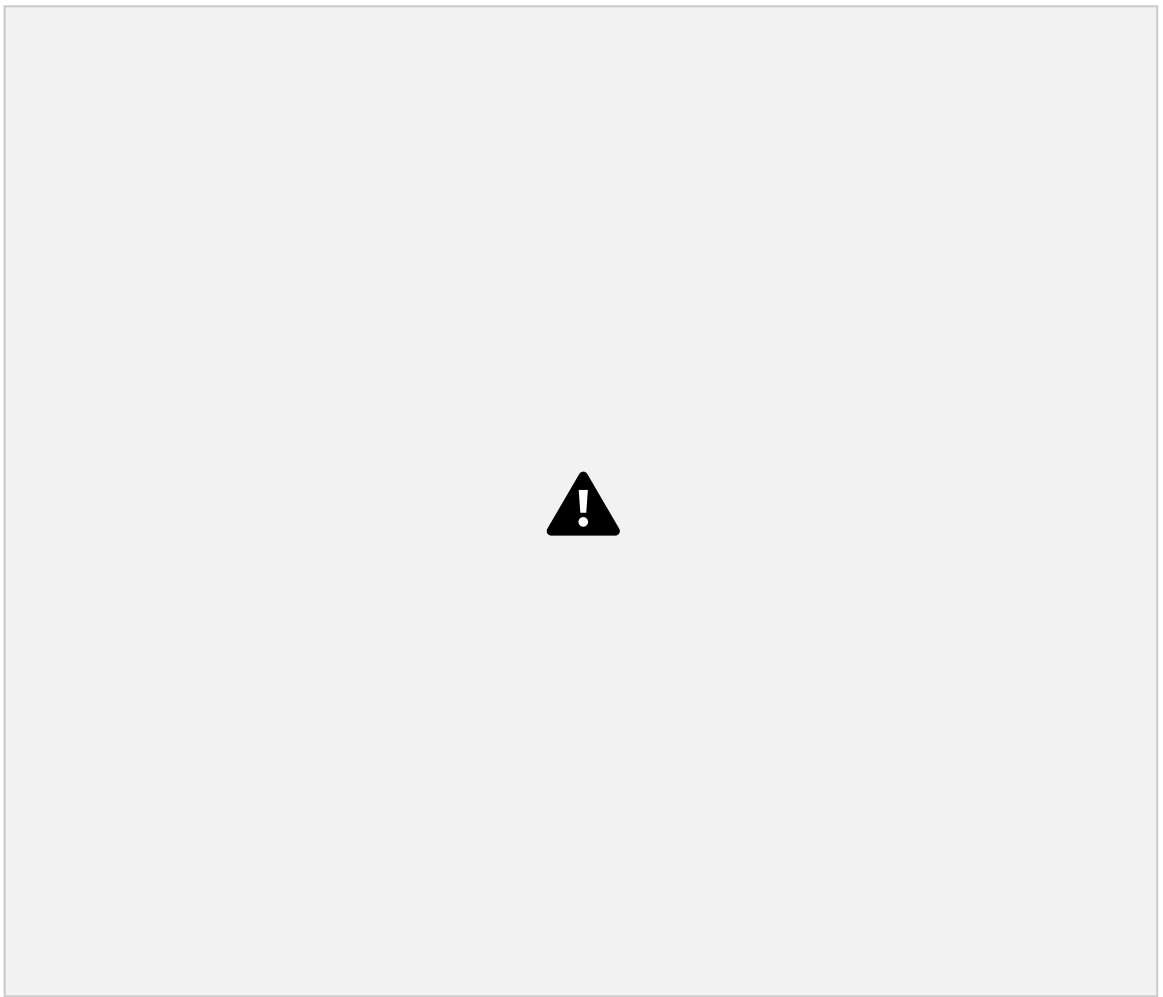
## > File

รูปแบบนี้จะเป็นการกำหนดให้เอาต์พุต ซึ่งปกติจะถูกแสดงออกมาทางหน้าจอเปลี่ยนไปเก็บลง ไฟล์ที่กำหนด

รูปที่ 5.2 ภาพอธิบายหลักการทำงานของรีไดเร็กชันลงไฟล์แบบ > File



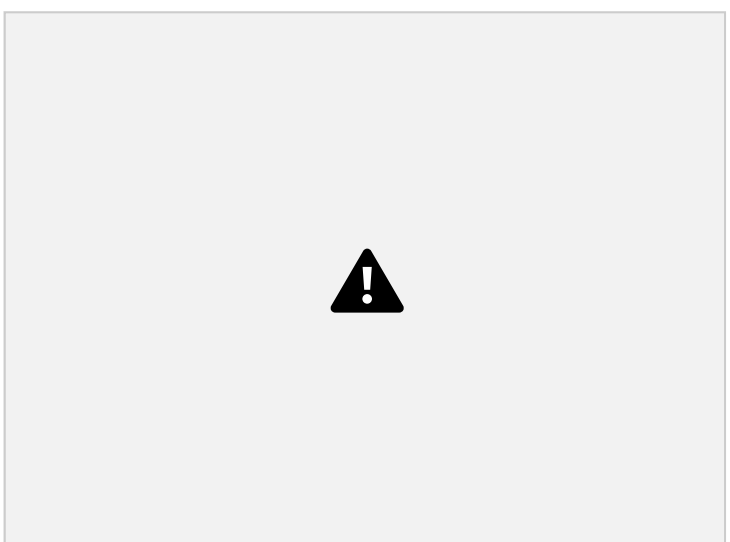
รูปที่ 5.3 ตัวอย่างการรีไดเร็กชันลงไฟล์แบบ > File ตัวอย่างในรูปที่ 5.3 ด้านบน เป็นการใช้คำสั่ง ls -l ซึ่งถ้าเป็นการใช้ตามปกติม่กำหนดรีไดเร็ก ชัน ผลการทำงานของคำสั่งจะปรากฏในเอาต์พุตมาตรฐานคือจอภาพ แต่เมื่อใช้คำสั่ง ls -l > list_file ผลลัพธ์ที่ได้จะเปลี่ยนทิศทางของข้อมูลไปเก็บในไฟล์ list_file ซึ่งเมื่อใช้คำสั่ง cat เรียกดูเนื้อหาใน list_file จะมีข้อมูลจากการทำงานของคำสั่ง ls -l



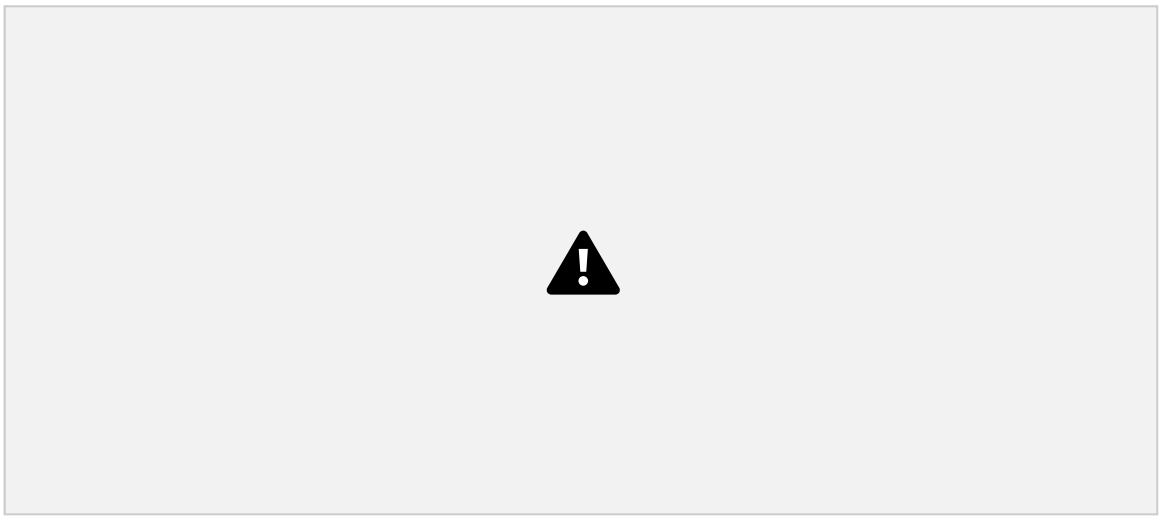
**< File**

**รูปแบบนี้จะเป็นการกำหนดให้ทำงานโดยรับอินพุตจากไฟล์ที่กำหนด และนำไปแสดงที่เอาต์พุต**

**รูปที่** 5.4 **หลักการรีไดเร็กชันแบบรับข้อมูลจากไฟล์แบบ** < File



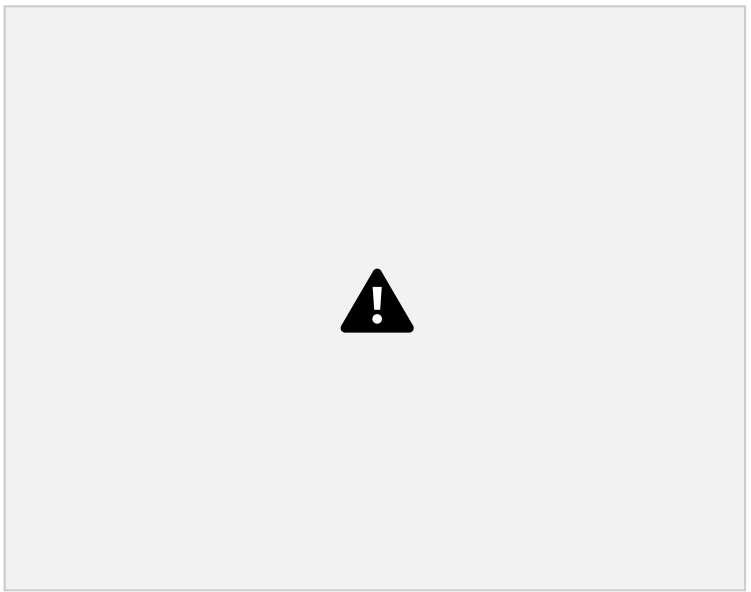
รูปที่ 5.5 ตัวอย่างการรีไดเร็กชันรับข้อมูลจากไฟล์แบบ < File จากตัวอย่างนี้จะเป็นการใช้คำสั่ง sort โดยให้มีการรับอินพุตมาจากไฟล์ข้อความ file1 ผลลัพธ์ที่ได้จะถูกแสดงออกทางหน้าจอตามปกติ



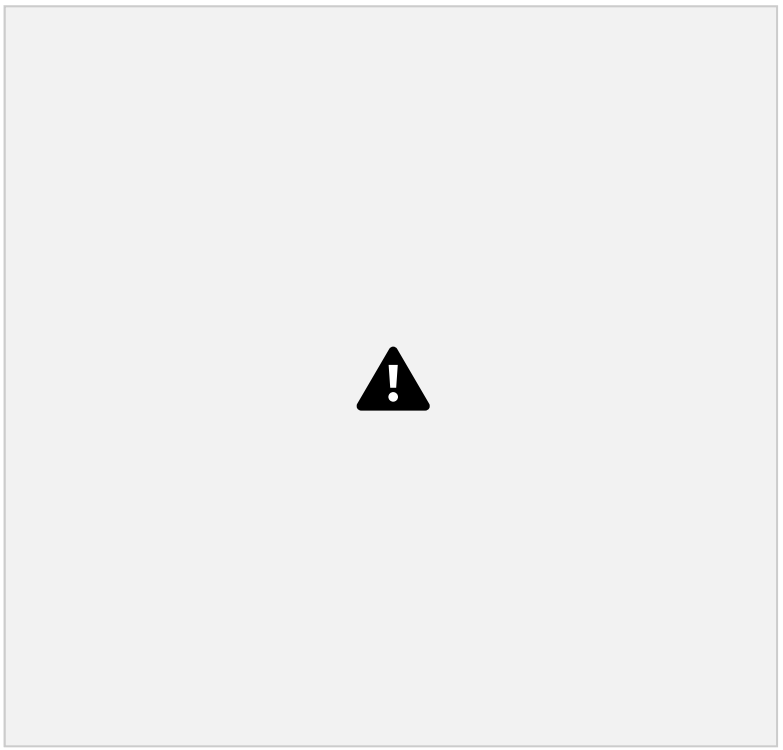
**>> File**

**รูปแบบรีไดเร็กชันแบบน ี้จะมีการเก็บผลลัพธ์ที่ได้ลงไฟล์เช่นเดียวกับแบบ** > File **แต่จะ แตกต่างกันตรงที่ถ้าไฟล์ที่กำหนดให้นำผลลัพธ์ไปเก็บนั้นมีอยู่แล้วก็จะนำผลลัพธ์ที่ได้ไปต่อไฟล์เดิม แต่ถ้าเป็น** > File **ถ้าไฟล์ที่กำหนดมีอยู่แล้วก็จะเขียนข้อมูลทับไฟล์เก่านั้นเลย จึงอาจจะทำให้สูญเสีย ข้อมูลที่สำคัญได ้และสำหรับรูปแบบนี้ถ้าไฟล์ที่กำหนดยังไม่มีก็จะทำการสร้างไฟล์นั้นขึ้นมาใหม่ให ้ฉะนั้นรูปแบบนี้จึงเป็นรูปแบบที่ปลอดภัยในการใช้งานมากกว่า ส่วนรูปแบบ** > File **เหมาะสมกับใน กรณีที่เราตั้งใจที่จะส่งผลลัพธ์ที่ได้เก็บลงไฟล์ทับไฟล์เดิมที่มีอยู่แล้ว**

**รูปท ี่** 5.6 **หลักการรีไดเร็กชันแบบรับข้อมูลจากไฟล์แบบ** >> File



**รูปที่** 5.7 **การรีไดเร็กชันแบบ** >> File

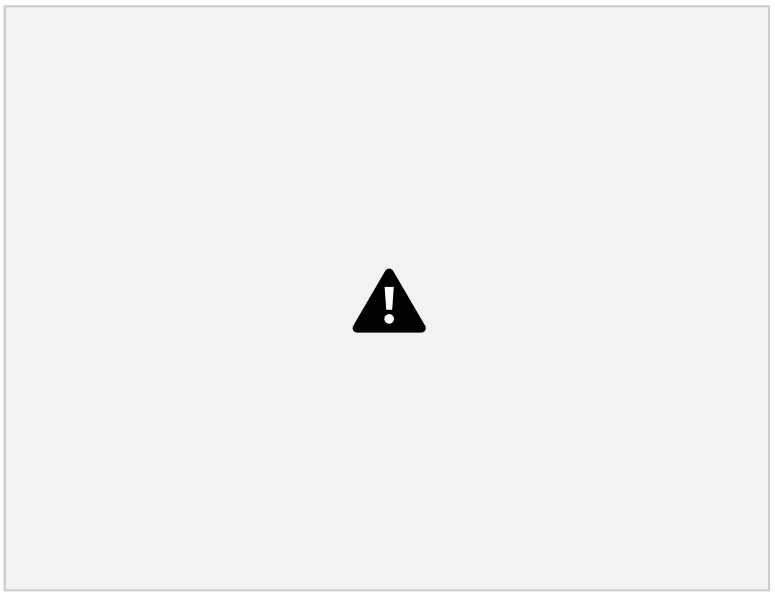


จากตัวอย่างในรูปที่ 5.7 เป็นการใช้คำสั่ง date '+%d-%m-%y' แล้วนำผลลัพธ์ที่ได้จากการ ใช้คำสั่ง date ไปเขียนต่อไฟล์ list_file ที่มีอยู่แล้ว

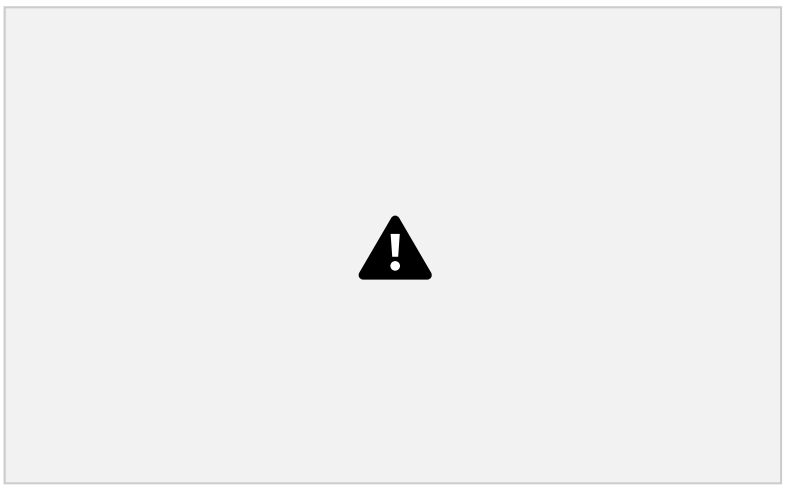
## << Label

รูปแบบนี้ไม่ค่อยนำมาใช้กันบ่อยนัก แต่ก็เป็นรูปแบบที่ค่อนข้างมีประโยชน์พอสมควร การ ทำงานของรูปแบบนี้คือ หลังเครื่องหมาย << จะเป็นสัญลักษณ์หรือตัวอักษรอะไรก็ได้ ระบบจะจดจำ ตัวอักษรนี้เอาไว้ และบรรทัดหลังจากสัญลักษณ์นี้จะถูกส่งไปเป็นอินพุตของคำสั่งด้านหน้าเครื่องหมาย << และระบบจะหยุดรับอินพุตเมื่อเจอสัญลักษณ์ที่กำหนดไว้ในตอบแรกอีกครั้ง รูปแบบนี้นักเขียน โปรแกรมมักจะนำไปใช้ประโยชน์ในการเขียนโปรแกรมในส่วนที่ต้องการมีการโต้ตอบระหว่างผู้ใช้กับ ระบบ เช่น คำสั่ง rm -i จะเป็นคำสั่งในการลบไฟล์ส่วนที่จะต้องมีการตอบยืนยันว่าจะลบหรือไม่ โดย ใช้ตัวอักษร y หรือ n ซึ่งถ้ารู้คำตอบอยู่แล้วก็สามารถใช้การรีไดเร็กชันแบบนี้เพื่อป้อนคำตอบเหล่านี้ให้กับคำสั่งได้อย่างอัตโนมัติ

รูปที่ 5.8 หลักการรีไดเร็กชันแบบรับข้อมูลจากไฟล์แบบ << Label



รูปที่ 5.9 การใช้รีไดเร็กชันแบบ << Label



จากตัวอย่างจะเป็นการใช้คำสั่ง rm -i เพื่อลบไฟล์ test และไฟล์ file1 ซึ่งออปชัน –i เป็นออป ชันที่จะต้องมีการยืนยันการลบ ฉะนั้นจึงต้องมีการตอบ y หรือ n เพื่อเลือกว่าจะลบหรือไม่ลบ โดยจาก รูปที่ 5.9 ข้างบน ในส่วนของ << a ตัว a จะเป็น Label เพื่อบอกขอบเขตของอินพุตและอีก 2 บรรทัด ต่อมาจะเป็นอินพุตที่ถูกส่งเข้าไป โดย y บรรทัดที่ 2 จะหมายถึง ยืนยันให้ลบไฟล์ test และ n ใน บรรทัดที่ 3 จะหมายถึงการตอบเพื่อไม่ลบไฟล์ file1 ส่วน a ในบรรทัดสุดท้ายนั้นจะเป็นตัวที่บอกถึง การสิ้นสุดการรับอินพุต

### n> File

รูปแบบนี้จะเป็นการส่งชนิดของอินพุตหรือเอาต์พุตที่กำหนดลงไฟล์

โดย n จะเป็นตัวเลขที่แสดงถึงชนิดของอินพุต เอาต์พุต โดยจะกำหนดดังนี้ ตัวเลข 0 หมายถึงอินพุตมาตรฐาน ตัวเลข 1 หมายถึงเอาต์พุตมาตรฐาน และตัวเลข 2 หมายถึงเอาต์พุตแสดงความผิดพลาดมาตรฐาน

**n< File**

**รูปแบบนี้จะเป็นการระบุการส่งไฟล์มายังชนิดของอินพุตหรือเอาต์พุตที่กำหนด**

**>&n**

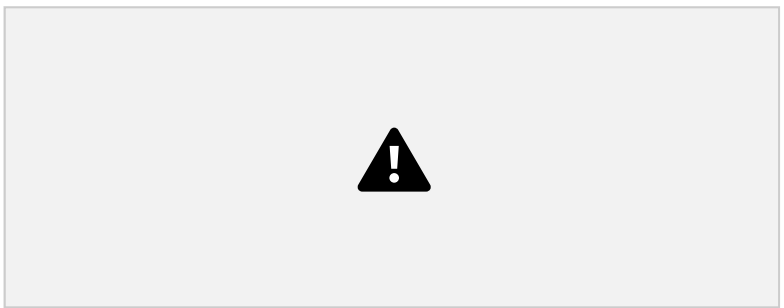
**รูปแบบนี้จะเป็นการสำเนาผลลัพธ์ที่แสดงบนหน้าจอไปแสดงที่ชนิดของอินพุตหรือเอาต์พุตที่กำหนด**

**<&n**

**รูปแบบนี้จะเป็นการนำเอาชนิดของอินพุตหรือเอาต์พุตที่กำหนดไปเป็นอินพุตของคำสั่งที่อยู่ด้านหน้า**

**&> File**

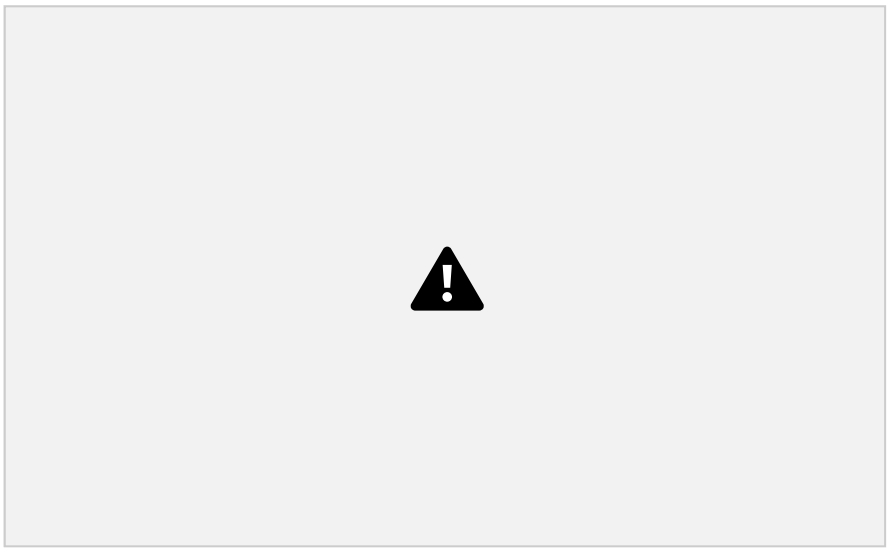
**รูปแบบนี้จะเป็นการเก็บผลลัพธ์และข้อความแสดงความผิดพลาดไว้ในไฟล์ที่กำหนด**



**รูปที่** 5.10 **การรีไดเร็กชันแบบผสมระหว่าง** n> **และ** >&n **จากตัวอย่างด้านบนจะเป็นการใช้รูปแบบของรีไดเร็กชันที่เป็นการผสมผสานกันระหว่างรีได เร็กชันแบบ** n> File **และ** >&n **คือ** ls -i ppp mylinux > test 2>&1 **ส่วนแรกคือ** ls –i ppp mylinux > test **นั้นจะเป็นการแสดงรายละเอียดของไฟล์** 2 **ไฟล์คือไฟล์** ppp **และไฟล์** mylinux **ซึ่งไฟล์** mylinux **นั้นจะมีอยู่แล้ว แต่ไฟล์** ppp **จะไม่มี และจากคำสั่งได้กำหนดให้ผลลัพธ์ถูกส่งไปเก็บที่ไฟล์** test **และส่วน**

2>&1 จะหมายถึงให้เอาข้อความที่แสดงความผิดพลาดที่ได้จากการใช้คำสั่ง ls -i ppp ไปเก็บไว้ในไฟล์ &1 ก็จะหมายถึงไฟล์เดียวกับที่เก็บเอาต์พุตซึ่งก็คือ ไฟล์ test นั่นเอง นอกจากนี้เรายังสามารถนำรูปแบบหลาย ๆ รูปแบบมารวมกันได้ เช่น คำสั่ง cp เพื่อสร้างสำเนา ไฟล์ ซึ่งผลลัพธ์จะได้ไฟล์ 2 ไฟล์ที่มีข้อมูลเหมือนกัน เราสามารถนำความสามารถของรีไดเร็คชั่นมาใช้แทนการใช้คำสั่ง cp ได้โดยใช้คำสั่ง cat < file1 > file2 ดังรูปที่ 5.11 จะใช้คำสั่ง cat < list_file > show_file ในการสำเนาไฟล์ list_file ให้กับ show_file

รูปที่ 5.11 การใช้คำสั่ง cat < list_file > show_file แทนคำสั่ง cp ของลินุกซ์



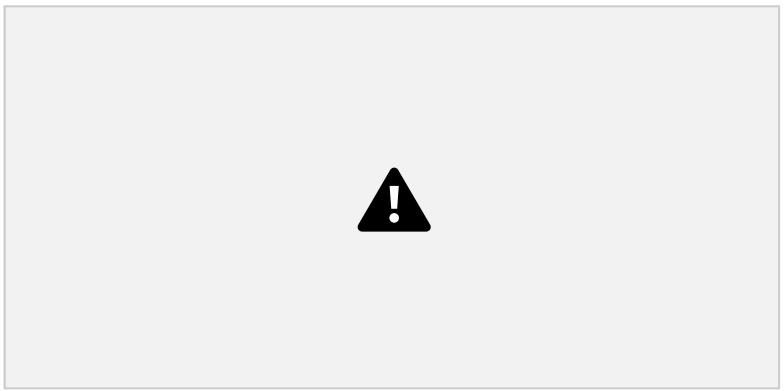
## 5.3. ไปป์ไลน์ (Pipeline)

นอกจากการใช้รีไดเร็กชันสำหรับการเปลี่ยนทิศทางที่เกี่ยวข้องกับไฟล์แล้ว ลินุกซ์และยูนิกซ์ยังสามารถเปลี่ยนทิศทางผลลัพธ์ของคำสั่งหนึ่งให้ไปเป็นอินพุตให้อีกคำสั่งหนึ่งได้โดยใช้ความสามารถของไปป์ไลน์สำหรับวิธีในการไปป์ไลน์นั้น ก็โดยใช้เครื่องหมาย | ไปคั่นอยู่ระหว่างคำสั่ง 2 คำสั่ง โดยผลลัพธ์ของคำสั่งด้านหน้าหรือก่อนเครื่องหมายไปป์นั้นจะกลายไปเป็นอินพุตของ คำสั่งที่อยู่หลังเครื่องหมายไปป์

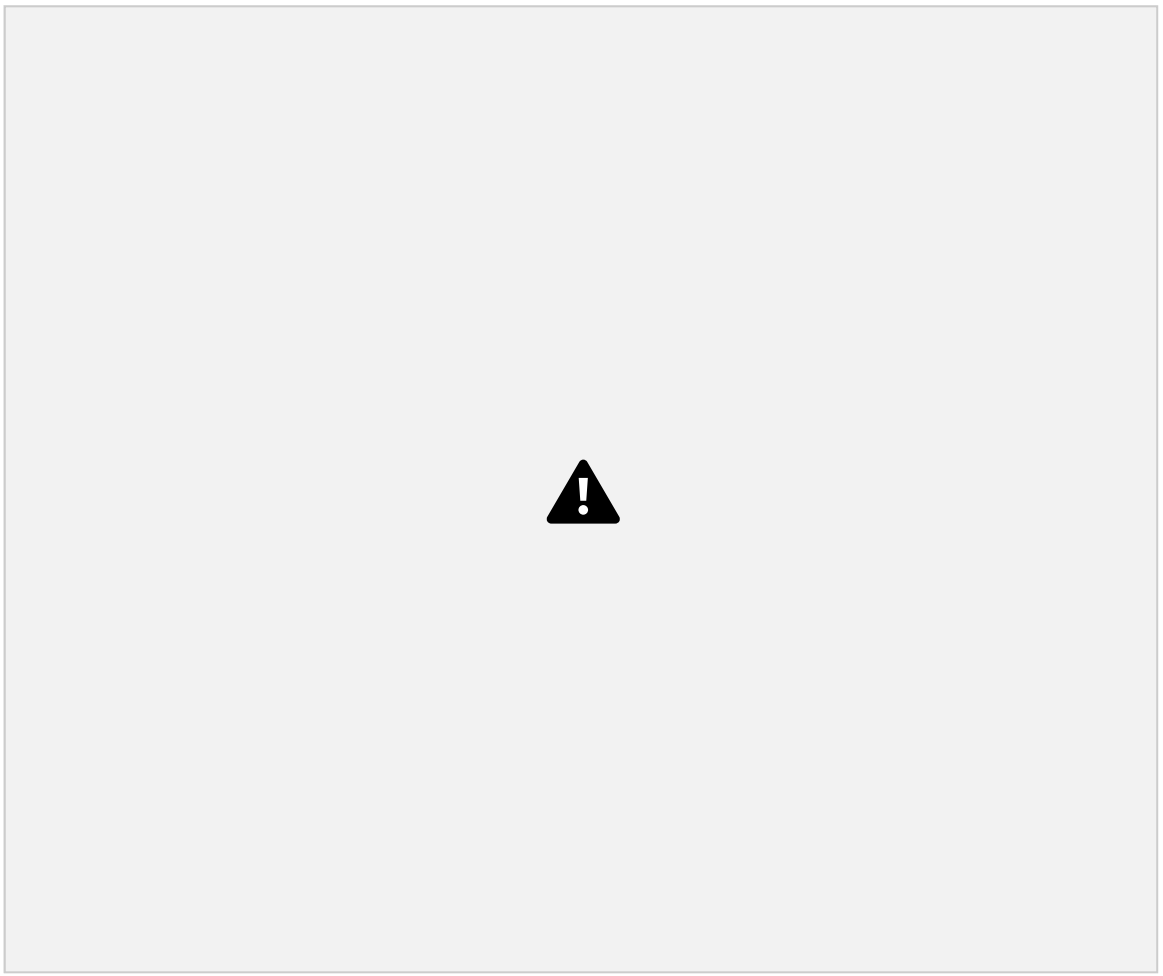


รูปที่ 5.12 ภาพอธิบายการทำงานของไปป์ไลน์ตัวอย่างของการใช้ประโยชน์จากไปป์ไลน์เช่น ในกรณีที่คำสั่ง finger ซึ่งใช้ในการแสดง รายละเอียดของผู้ใช้งานระบบ ได้เกิดความเสียหายขึ้นจนไม่สามารถใช้งานได้วิธีหนึ่งที่สามารถจะ ทราบรายละเอียดของผู้ใช้โดยไม่ใช้คำสั่ง finger โดยนำเอาความสามารถของไปป์ไลน์เข้ามาช่วย ใน ขั้นแรกเราจะต้องรู้ก่อนว่าในระบบไฟล์ของลินุกซ์นั้น จะมีไฟล์อยู่ไฟล์หนึ่งที่เก็บข้อมูลทั้ง

หมด เกี่ยวกับผู้ใช้ในระบบนั้นคือ ไฟล์ passwd ซึ่งเก็บอยู่ในไดเร็กตอรี่/etc แทนที่จะเปิดไฟล์ passwd ขึ้น มาแล้วก็หาชื่อที่ต้องการ ก็ใช้ไปป์ไลน์เข้ามาช่วยโดยการส่งผลลัพธ์ที่ไดจากการเปิดไฟล์ passwd ไป ให้กับคำสั่งที่ใช้ในการค้นหาที่ต้องการ เช่นถ้าเราต้องการหาคนชื่อ webmin ในไฟล์ passwd ก็ให้ใช้คำสั่ง cat /etc/passwd | grep webmin เป็นต้น ซึ่งก็จะมีผลลัพธ์เหมือนกับการให้คำสั่ง grep webmin /etc/passwd นั่นเอง



รูปที่ 5.13 การใช้ไปป์ไลน์จากรูปที่ 5.13 จะมีการนำผลลัพธ์จากคำสั่ง cat ไปเป็นอินพุตให้กับคำสั่ง grep เพื่อให้แสดง รายละเอียดของ webmin นอกจากนี้ยังนำความสามารถของไปป์ไลน์ไปรวมกับการรีไดเร็กชัน เพื่อการใช้งานอย่างมีประสิทธิภาพ เช่น คำสั่ง cut –d: -f1 < /ect/passwd | sort ตามตัวอย่างในรูปด้านล่างจะเป็นการ เรียงลำดับชื่อคนทั้งหมดในระบบถูกเก็บอยู่ในไฟล์ /ect/passwd ขึ้นมาแล้วเลือกเอาฟิลด์ที่ 1 ของไฟล์ที่ได้จากการแบ่งโดยใช้เครื่องหมาย : เป็นตัวแบ่ง ซึ่งผลลัพธ์ก็คือ รายชื่อของผู้ใช้ทั้งหมดที่อยู่ในระบบ จะถูกส่งไปเป็นอินพุตให้กับคำสั่ง sort หลังจากนั้นคำสั่ง sort ก็จะนำชื่อผู้ใช้เหล่านั้นมาเรียงลำดับตาม ตัวอักษรแล้วแสดงผลออกทางหน้าจอ แต่ถ้าในระบบที่ใหญ่มีข้อมูลที่จะแสดงผลจำนวนมาก การ แสดงผลบนหน้าจอจะเกิดขึ้นรวดเร็วมากและมีการเลื่อนข้อมูลไปจนจบ ทำให้ดูไม่ทัน ก็สามารถ แก้ไขด้วยการใช้คำสั่งพิมพ์ออกทางเครื่องพิมพ์คือ คำสั่ง lp เข้าช่วยในการแสดงผล ดังนั้นจากใน ตัวอย่าง ถ้าจะต้องการส่งผลลัพธ์ที่ได้ดังข้างต้นไปให้กับคำสั่ง lp อีกทีหนึ่ง คำสั่งที่ใช้ก็จะมีรูปแบบเป็น cut –d: -f1 < /ect/passwd | sort | lp



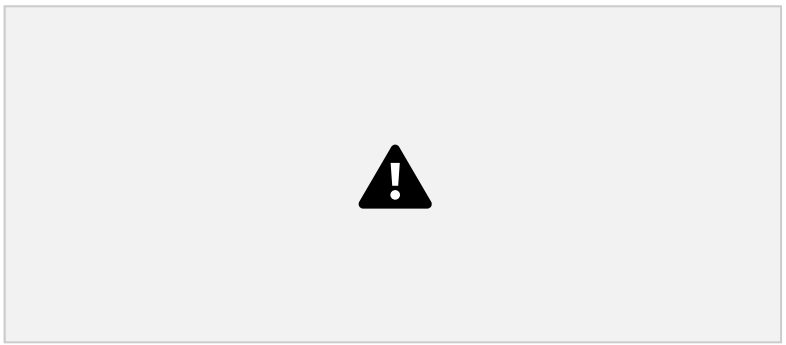
## 5.4. ซีเควนซ์ (Sequence)

การใช้งานคำสั่งบนระบบยูนิกซ์ที่ผ่าน ๆมาเราเรียกว่าเป็นการใช้คำสั่งแบบคอมมานด์ไลน์หรือ พูดง่าย ๆ ก็คือ เป็นการใช้คำสั่งในบรรทัดเดียวนั่นเอง แต่ก็ใช่ว่าในบรรทัดนั้นจะใช้งานคำสั่งได้เพียง คำสั่งเดียวเท่านั้น เราสามารถที่จะนำคำสั่งหลาย ๆ คำสั่งมาต่อกันได้เรื่อย ๆ ตามคุณสมบัติของซีเควนซ์โดยการนำเครื่องหมาย ; มาคั่นระหว่างคำสั่งแต่ละคำสั่ง ระบบก็จะทำคำสั่งเหล่านั้นทีละคำสั่งจาก ด้านซ้ายไปขวาเรื่อย ๆ ตามลำดับ ผลลัพธ์ที่ได้จากคำสั่งสุดท้ายก็จะถูกแสดงออกทางหน้าจอ

จากตัวอย่างด้านล่าง จะเห็นว่าระบบจะทำคำสั่งทีละคำสั่ง โดยจะเริ่มจากการสร้างไดเร็กตอรี my_folder หลังจากนั้นก็ก็อปปี้ไฟล์ list_file ไปไว้ในไดเร็กตอรี my_folder โดยตั้งชื่อเสียใหม่ว่า my_file แล้วก็ทำการย้ายได

เร็กตอรีที่จะทำงานไปอยู่ในไดเร็กตอรี my_folder และใช้คำสั่ง ls สั่งให้แสดงรายชื่อไฟล์ในไดเร็กตอรี my_folder

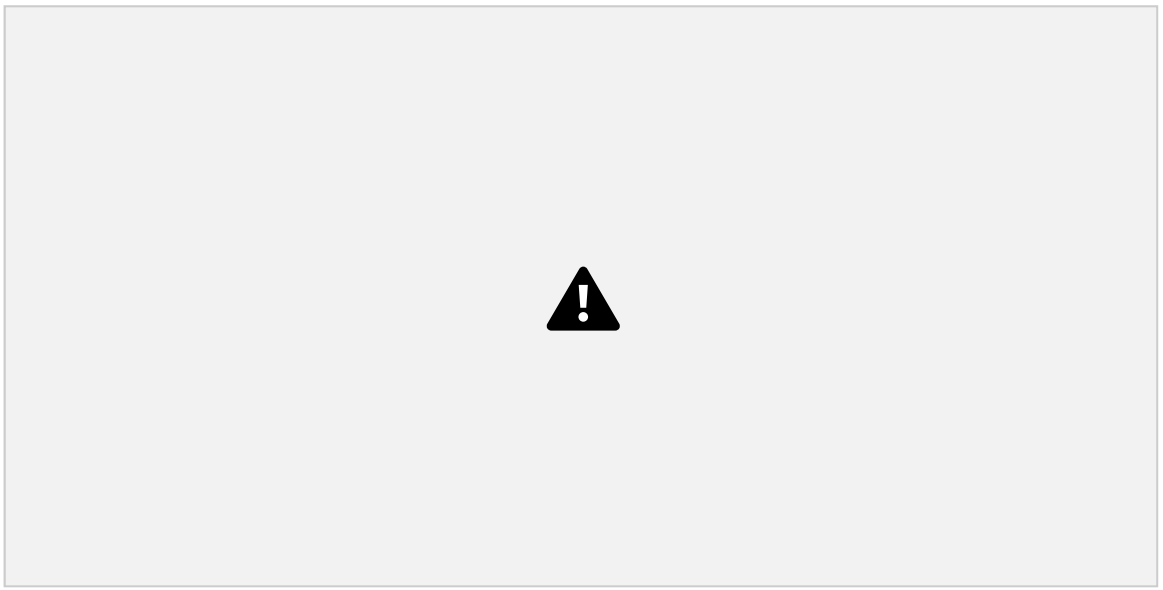
รูปที่ 5.15 การใช้คุณสมบัติของซีเควนซ์



## 5.5. กรุ๊ป (Group)

การใช้คำสั่งแบบซีเควนซ์นั้น ระบบจะทำคำสั่งเหล่านั้นทีละคำสั่งจากทางด้านซ้ายมือ แต่การ กรุ๊ปจะแบ่งคำสั่งหลายๆ คำสั่งนี้ออกเป็นกลุ่ม ๆ โดยใช้เครื่องหมายวงเล็บเปิด ( และวงเล็บปิด ) ครอบ คำสั่งที่ต้องการจะรวมกลุ่มกัน สำหรับตัวอย่างของการทำกรุ๊ปนี้เราจะใช้คำสั่งเดียวกับในตัวอย่างของ การทำซีเควนซ์เพียงแต่จะมีการกรุ๊ปคำสั่งเหล่านั้นไว้ด้วยกัน โดยการนำเอาเครื่องหมาย ( และ ) ครอบ คำสั่งเหล่านั้นไว ้ให้ลองสังเกตตำแหน่งที่ทำงานปัจจุบัน (Current Path) ของการทำงานแบบซีเควนซ ์และแบบกรุ๊ปดูว่าเหมือนหรือต่างกันอย่างไร

รูปที่ 5.16 การใช้คำสั่งด้วยคุณสมบัติของกรุ๊ป ซึ่งเมื่อพิจารณาแล้ว จะเห็นว่า การใช้กรุ๊ปกับคำสั่งนั้น ตำแหน่งที่ทำงานปัจจุบันจะไม ่เปลี่ยนแปลง ซึ่งจะแตกต่างกับคำสั่งแบบซีเควนซ์ที่เป็นเช่นนี้เพราะว่า การทำงานถูกควบคุมอยู่ใน วงเล็บ ( และ ) ไม่มีการแสดงออกมาทางหน้าจอได ้

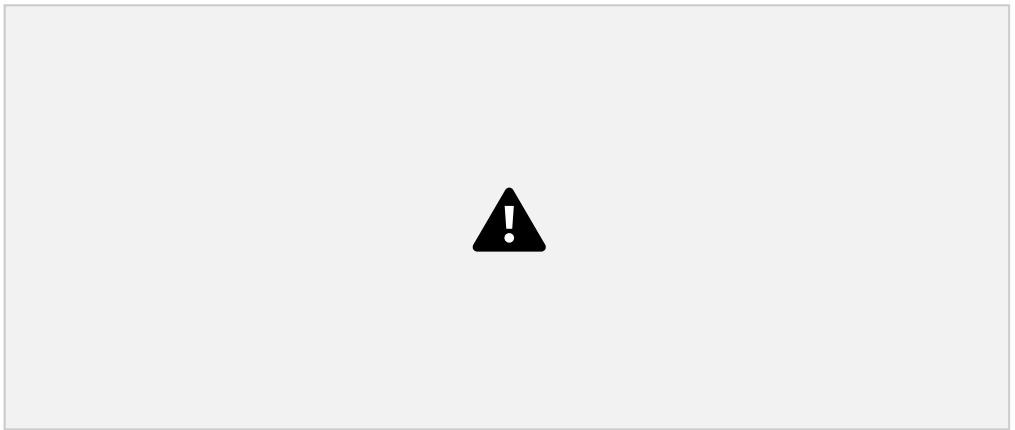


## 5.6. การทำงานเบื้องหลัง (Background Jobs)

การทำงานระบบลินุกซ์โดยปกติในเวลาหนึ่ง ๆ เราสามารถที่จะทำงานได้เพียงหนึ่งงานเท่านั้น หากต้องการจะทำงานอีกงานต่อไป จะต้องรอให้งานแรกทำเสร็จก่อนจึงจะสั่งงานต่อไปได้ แต่ด้วย คุณสมบัติของการทำงานในเบื้องหลัง ทำให้เราสามารถส่งงานที่จะไปทำงานในเบื้องหลังได้ โดยต้อง ใส่เครื่องหมาย & ไว้หลังสุดของคำสั่งที่จะทำ งานนั้นก็จะถูกส่งไปทำอยู่เบื้องหลังและการทำงานของ เราก็จะกลับมาที่เชลล์พร้อมป์โดยทันที งานที่เหมาะที่จะทำในแบบเบื้องหลังนั้น สามารถจะเป็นงาน

อะไรก็ได้ระบบจะยอมให้งานทุกงานถูกส่งไปทำงานในเบื้องหลังได้หมด แต่ส่วนมากมักนิยมที่จะส่ง งานที่ค่อนข้างใช้เวลาในการทำงานมากไปทำงานเบื้องหลังมากกว่า เพราะจะได้ไม่ต้องเสียเวลารอให้งานทำเสร็จ และถ้าเราส่งงานไปทำในเบื้องหลังแล้วเราสามารถตรวจสอบสถานะของงานเหล่านี้ได้โดย ใช้คำสั่ง jobs ระบบจะแสดงสถานะของงานเหล่านี้ว่ากำลังทำอยู่หรือว่าทำเสร็จไปแล้ว

รูปที่ 5.17 การส่งงานไปทำเบื้องหลังและการใช้คำสั่ง jobs เพื่อดูสถานะของงานในระบบ



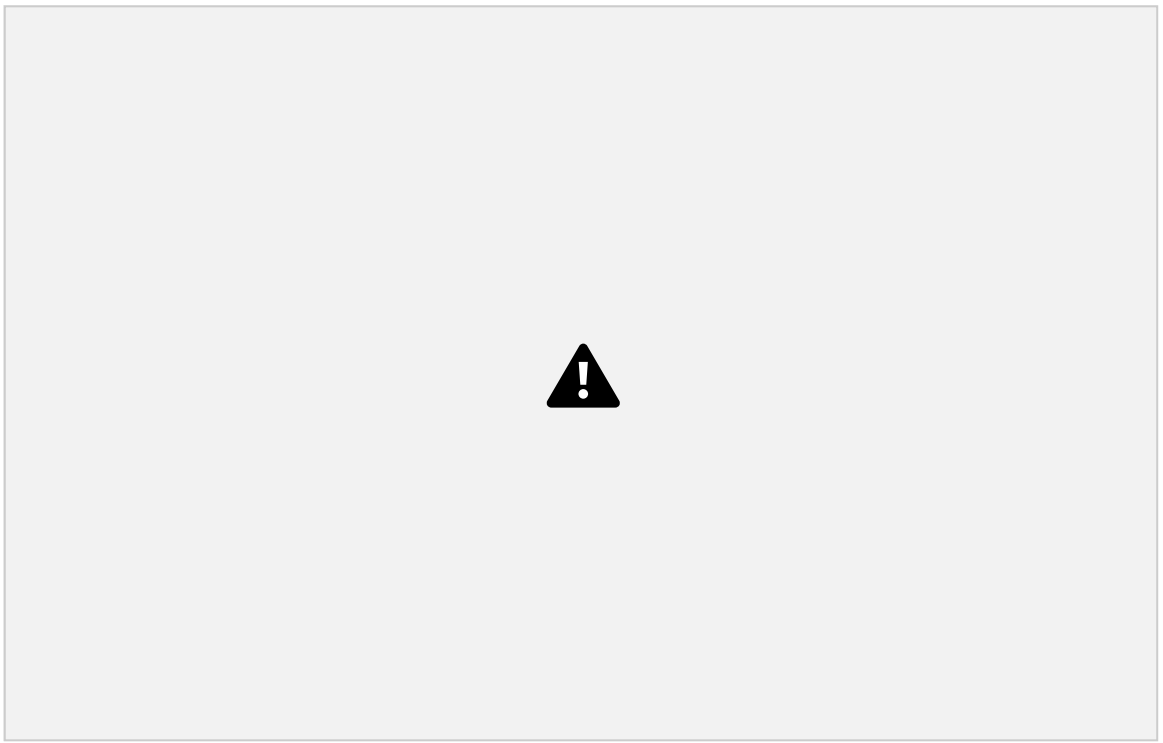
## 5.7. อินพุตและเอาต์พุตของงานเบื้องหลัง

การทำงานในเบื้องหลังจะมีข้อจำกัดอยู่อีกประการหนึ่งคือ ความไม่เหมาะกับงานที่จะต้องมีการโต้ตอบกับผู้ใช้กล่าวง่าย ๆ ก็คือ เป็นงานที่ต้องมีการป้อนอินพุตเข้าไปหรือแสดงผลลัพธ์ออกมา ทางหน้าจอ โดยถ้างานที่เราสั่งไปทำในเบื้องหลังเป็นงานที่ต้องการอินพุต งานนั้นก็จะไม่ทำงานและจะ รออินพุตไปเรื่อย ๆ หรือถ้างานนั้นเป็นงานที่จะต้องส่งผลลัพธ์ออกมาก็จะส่งผลลัพธ์ออกมาทางหน้าจอ โดยทันทีซึ่งก็จะทำให้การทำงานไม่ต่างอะไรกับการทำงานตามปกติ

เราสามารถที่จะแก้ปัญหานี้ได้โดยใช้คุณสมบัติของการรีไดเร็กชันเข้ามาช่วย โดยถ้างานนั้น ต้องการอินพุตก็ให้ส่งอินพุตไฟล์ให้กับงานนั้น (< File) หรือถ้างานนั้นต้องมีการส่งผลลัพธ์ออกมาทาง หน้าจอก็ให้เปลี่ยนทิศทางของผลลัพธ์นั้นลงไฟล์แทน(< File) แต่อย่างไรก็ตาม สิ่งที่ควรระวังในการ ใช้รีไดเร็กชันคือ งานนั้นอาจจะมีข้อผิดพลาด ซึ่งก็จะแสดงความผิดพลาด(Standard Error Output) นั้นออกมาทางหน้าจอเช่นเดียวกัน เราสามารถหลีกเลี่ยงได้ด้วยการส่งข้อความผิดพลาดเหล่านั้นไปเก็บ ในไฟล์ซึ่งอาจเป็นไฟล์เดียวกับไฟล์ที่เก็บผลลัพธ์หรือเป็นคนละไฟล์กันก็ได ้แต่ไฟล์ที่ผู้ดูแลระบบ มักจะส่งข้อความที่แสดงความผิดพลาดนี้ไปเก็บมักจะเป็นไฟล ์/dev/null ไฟล์นี้จะมีไว้ใช้เก็บอะไรก ็ได้ที่เราไม่ต้องการแล้ว เหมือนกับรีไซเคิลบิน (Recycle Bin) ในระบบปฏิบัติการวินโดว์นั่นเอง

รูปที่ 5.18 การใช้งานอินพุตและเอาต์พุตของงานเบื้องหลัง

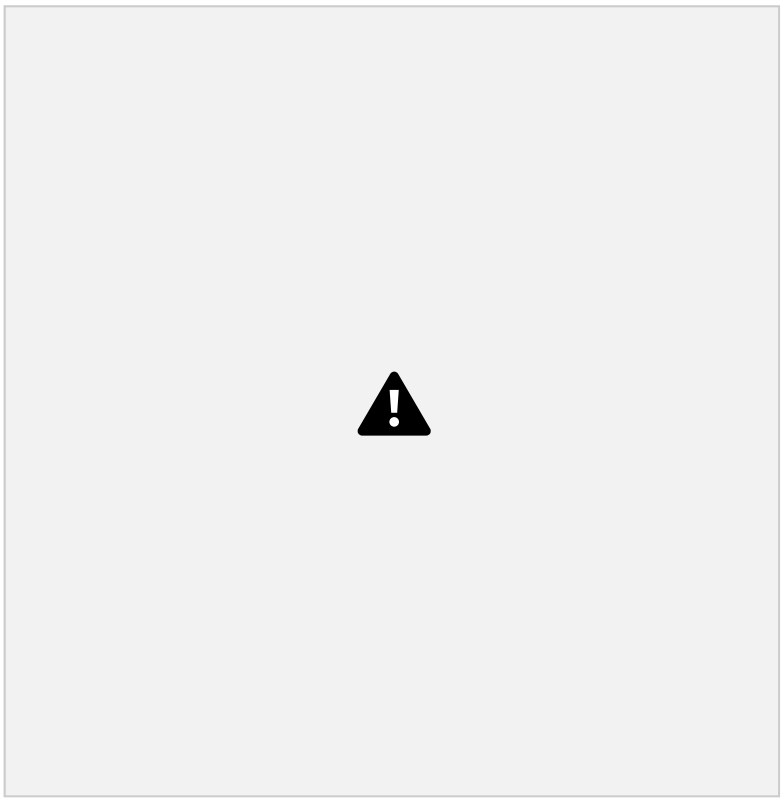
จากตัวอย่างจะเป็นการค้นหาไฟล์ core ซึ่งเป็นไฟล์ที่เกิดจากความผิดพลาดในการทำงานจาก คำสั่ง เราจะเริ่มค้นหาตั้งแต่ไดเร็กตอรีราก (Root Directory) แต่ว่าในบางไดเร็กตอรีนั้นเราจะไม่มี สิทธิที่จะเข้าไปค้นหา จึงทำให้เกิดข้อความผิดพลาดออกมามันก็จะถูกส่งไปเก็บที่ไฟล์ /dev/null และ ผลลัพธ์ของการค้นหาก็จะถูกส่งไปเก็บที่ไฟล์ core_file แทนที่จะถูกส่งไปที่หน้าจอ



## 5.8. คำสั่ง Tee

นอกจากลินุกซ์หรือยูนิกซ์จะสามารถเปลี่ยนทิศทางลงไฟล์ได้แล้ว ยังสามารถใช้คำสั่ง tee เป็น คำสั่งที่ใช้ในการเปลี่ยนทิศทางผลลัพธ์โดยจะยังแสดงผลลัพธ์ออกมาบนหน้าจอด้วย โดยจะใช้ร่วมกับ คุณสมบัติของไป ป์ เช่น ตามตัวอย่างรูปที่ 5.19 ด้านล่างจะเป็นการเรียกดูปฏิทินของเดือนกุมภาพันธ์ ปี 2000 ซึ่งจะใช้คำสั่ง tee ทำให้นอกจากจะแสดงผลลัพธ์บนหน้าจอแล้วยังนำผลลัพธ์ที่แสดงบน หน้าจอไปเก็บลงไฟล์ชื่อ calendar อีกด้วย

รูปที่ 5.19 การใช้คำสั่ง tee



## 5.9. Back Quote (` `)

เราสามารถที่จะนำผลลัพธ์จากการทำของคำสั่งหนึ่งไปใช้เป็นส่วนหนึ่งของอีกคำสั่งหนึ่งได ้โดยใช้เครื่องหมาย ` (เราเรียกมันว่า Back quote) การใช้งานสามารถทำได้โดยให้ใช้เครื่องหมายน ีครอบคำสั่งที่เราต้องการ เช่น `date'+%d%m%y` ระบบก็จะทำคำสั่งที่ได้ระบุไว้ในเครื่องหมาย back quote นั้น ซึ่งเราสามารถที่จะนำผลลัพธ์ของคำสั่งนี้ไปใช้งานต่อได้ (เครื่องหมาย ` จะอยู่บนคีย์เดียวกับเครื่องหมาย ~)

รูปที่ 5.20 การใช้ Back quote



## 5.10. ซับเชลล์ (Subshell)

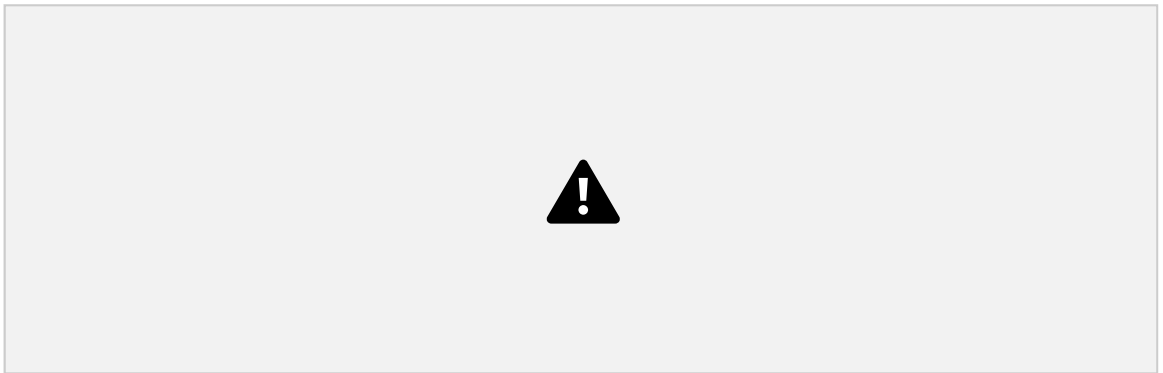
การใช้งานเซลล์ในระบบลินุกซ์/ยูนิกซ์นั้น สามารถที่จะใช้เชลล์ที่เรากำลังใช้งานอยู่สร้างเชลล์ใหม่ขึ้นมาใช้งานได้ เราจะเรียกเชลล์ที่เป็นเชลล์ที่สร้างเชลล์อีกเชลล์หนึ่งขึ้นมาว่าเชลล์หลักหรือ

parent และเรียกเชลล์ที่ถูกสร้างขึ้นมาใหม่ว่า child หรือสับเชลล ์การสร้างเชลล์ขึ้นมาใหม่เพื่อใช้งาน นั้น เราอาจะนำมาใช้ประโยชน์ในการกรุ๊ปหรืออาจจะใช้ในการเขียนสคริปต์ก็ได ้แล้วแต่ลักษณะงาน แต่สิ่งที่ควรจำก็คือขณะที่สับเชลล์กำลังทำงานอยู่นั้นเชลล์ที่เป็น parent จะทำงานไม่ได้จะต้องรอ จนกว่าสับเซลล์จะทำงานเสร็จเสียก่อน นอกเสียจากว่าสับเชลล์นั้นถูกสั่งให้ทำงานแบบเบื้องหลัง

(Background Process)

รูปที่ 5.21 การถ่ายทอดคุณสมบัติของเชลล์

เชลล์ที่ถูกสร้างขึ้นมาใหม่นี้จะได้รับการถ่ายทอดคุณสมบัติหลายอย่างจากเชลล์ที่เป็น parent ซึ่งคุณสมบัติเหล่านี้ได้แก่



- ไดเร็กตอรีที่ทำงานอยู่ (Current Directory) ของสับเชลล์จะเป็นไดเร็กตอรีเดียวกันกับ เชลล์ที่เป็น parent
- ค่าของตัวแปรแวดล้อม (Environment variables) ของสับเซลล์จะมีค่าเหมือนกับของ เชลล์ parent
- ชนิดของอินพุตและเอาต์พุตต่าง ๆ

แต่อย่างไรก็ตามก็ยังมีบางคุณสมบัติที่สับเชลล์ไม่ได้ถูกถ่ายทอดมา

จากเชลล์ที่เป็นเชลล์ parent เช่น ตัวแปรเชลล์ต่าง ๆ ที่ไม่ใช่ตัวแปรแวดล้อม

## 5.11. โปรเซส (Process)

โปรเซส หมายถึงโปรแกรมที่กำลังทำงานอยู่ในขณะนั้น ระบบปฏิบัติการลินุกซ์/ยูนิกซ์จะ กำหนดการสร้างโปรเซสออกเป็น 2 ลักษณะได้แก่

- โปรเซสหลัก หรือโปรเซสแม่ (Parent Process)
- โปรเซสรองหรือโปรเซสลูก (Child Process) ซึ่งอธิบายได้ดังนี้ โปรเซสใดๆ ก็ตามที่ทำการสร้างโปรเซสอื่นขึ้นมา จะเรียกโปรเซสนั้นว่า โปรเซสหลัก ส่วนโปรเซสที่ถูกสร้างโดยโปรเซสหลักจะเรียกว่า โปรเซสรอง โดยแต่ละโปรเซสจะมีหมายเลขประจำโปรเซส (Process Identification) หรือ PID กำกับไว้ในลินุกซ์จะมีโปรเซสที่เป็น

โปรเซสหลักของทุกๆโปรเซส และมี PID เป็นหมายเลข 1 เสมอ ได้แก่โปรเซสที่มีชื่อว่า Init ซึ่งจะ เริ่มต้นทำงาน (Start) ในขั้นตอนสุดท้ายของการบู๊ตระบบ และจะทำการเรียกโปรเซสอื่นให้เริ่มต้น

ทำงานต่อไป โดยที่แต่ละโปรเซสจะสามารถสร้างโปรเซสรองของตัวเองขึ้นได้และโปรเซสรองที่สร้าง ขึ้นนี้ก็สามารถเป็นโปรเซสหลักของโปรเซสอื่นที่สร้างขึ้น โดยโปรเซสนั้นได้

รูปที่ 5.22 แสดงการสร้างโปรเซสหลักและโปรเซสรอง



โปรเซสรองที่ถูกสร้างขึ้น จะมีคุณลักษณะเหมือนกันกับ Parent Process แต่จะมี PID ที่แตกต่างกัน ซึ่งระบบปฏิบัติการลินุกซ์หรือยูนิกซ์จะแบ่งประเภทของโปรเซสออกเป็น 4 ประเภท ได้แก่

- **โปรเซสทำงานเบื้องหน้า** (Foreground Process) เป็นโปรเซสที่ทำงานในแบบรอรับคำ สั่งจากผู้ใช้โดยแสดงเครื่องหมายพร้อมป์ รอรับคำสั่ง กล่าวโดยสรุปก็คือเป็นโปรเซสที่ทำงานโดยมีการ ติดต่อกับผู้ใช้นั่นเอง
- **โปรเซสทำงานเบื้องหลัง** (Background Process) เป็นโปรเซสที่ทำงานในลักษณะเป็น ฉากหลังแตกต่างกับโปรเซสทำงานเบื้องหน้าซึ่งมีลักษณะการทำงานเป็นแบบฉากหน้า มักจะเป็น โปรแกรมหรือโปรเซส ที่ไม่มีความต้องการติดต่อกับผู้ใช้และใช้เวลาในการทำงานมาก เช่นการ แปลภาษา (Compile) เป็นต้น การสั่งงานของโปรเซสทำงานเบื้องหลังจะสามารถสั่งงานใหม่ได้โดย ไม่ต้องรอให้งานเดิมทำงานเสร็จ ซึ่งแตกต่างกับโปรเซสทำงานเบื้องหน้าที่จะต้องรอให้งานเดิมเสร็จสิ้น ก่อนจึงจะเริ่มสั่งงานใหม่ได้
- **โปรเซสเดมอน** (Daemon Process) มีลักษณะเป็นโปรเซสทำงานเบื้องหลังที่ในสภาพ ปกติจะไม่ทำงาน จนกว่าจะมีการเรียกใช้
- **โปรเซสซอมบี้** (Zombie Process) คือโปรเซสที่ยุติการทำงานแล้วแต่ยังคงค้างอยู่ใน ตารางโปรเซส (Process Table)

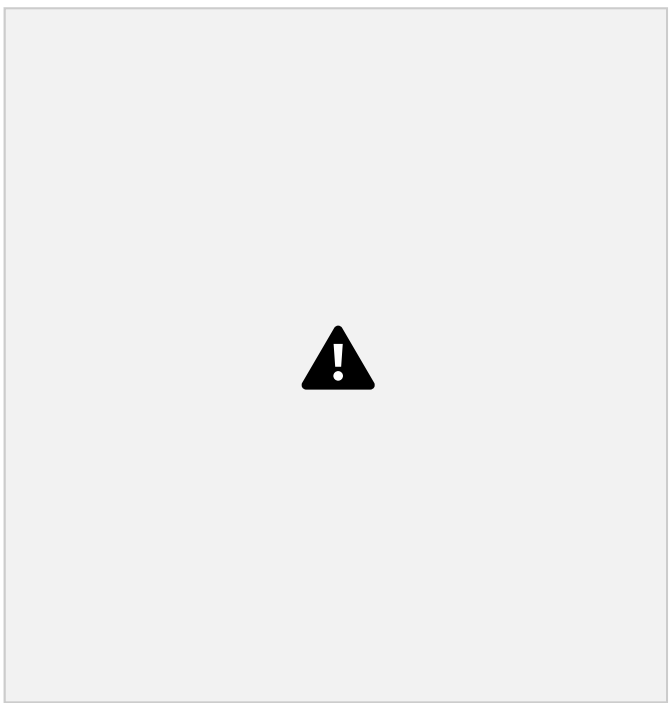
## 5.12. การติดต่อสื่อสาร (Communication)

การติดต่อสื่อสารของระบบปฏิบัติการลินุกซ์/ยูนิกซ์จะเป็นการติดต่อกันในลักษณะของไคล เอ็นต์และเซิร์ฟเวอร์ที่ทำงานอยู่เบื้องหลัง (Background Process) ในลักษณะของเดมอน (Daemon) โดยหลักการทำงานคือ เมื่อไคลเอ็นต์ติดต่อมายังเซิร์ฟเวอร์นั้นจะต้องมีการระบุที่อยู่ซึ่งก็คือ หมายเลข IP Address ของเซิร์ฟเวอร์ที่จะติดต่อ ไฟล์ที่จะใช้ตรวจสอบว่าพอร์ต โปรแกรมว่าเป็นอะไรและพอร์ตนั้น เปิดอยู่หรือไม่คือ ไฟล์ /etc/services ถ้าพอร์ตที่ไคลเอ็นต์ระบุมาว่าเซิร์ฟเวอร์เปิดให้ใช้ได้ก็จะไปดูต่อ

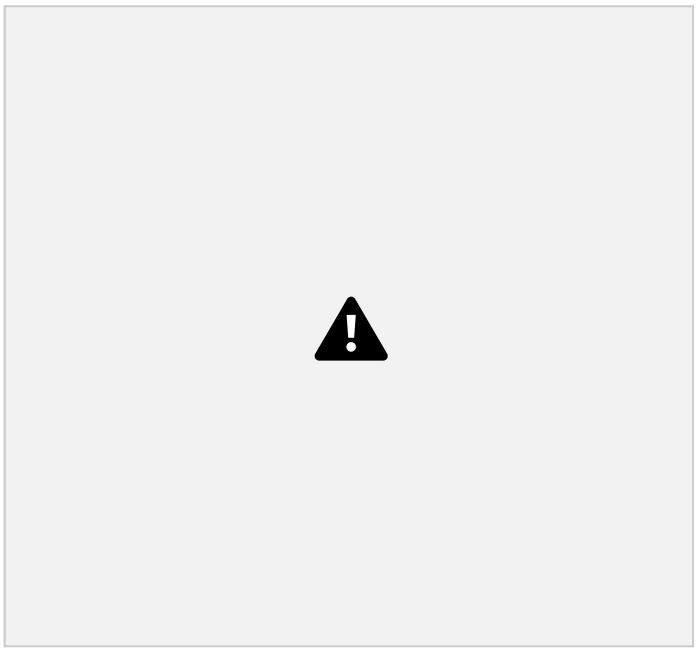
ที่ไฟล์ /etc/xinetd.d/ ซึ่งจะมีไฟล์ที่ใช้ตรวจสอบว่าถ้ามีการเรียกโปรแกรมอะไร หลังจากนั้นต้องไป เรียกโปรเซสอะไรขึ้นมาทำงานต่อ

ตัวอย่างเช่น ถ้ามีเครื่องไคลเอ็นต์ใช้คำสั่ง talk ซึ่งเป็นคำสั่งที่ใช้ในการคุยกันผ่านระบบเครื่อง เซิร์ฟเวอร์ก็จะไปทำการตรวจเช็คในไฟล์ /etc/services ก่อนว่าพอร์ตที่ใช้ในการ talk นี้เปิดให้ใช้งาน หรือไม่

รูปที่ 5.23 แสดงไฟล์ใน /etc/services

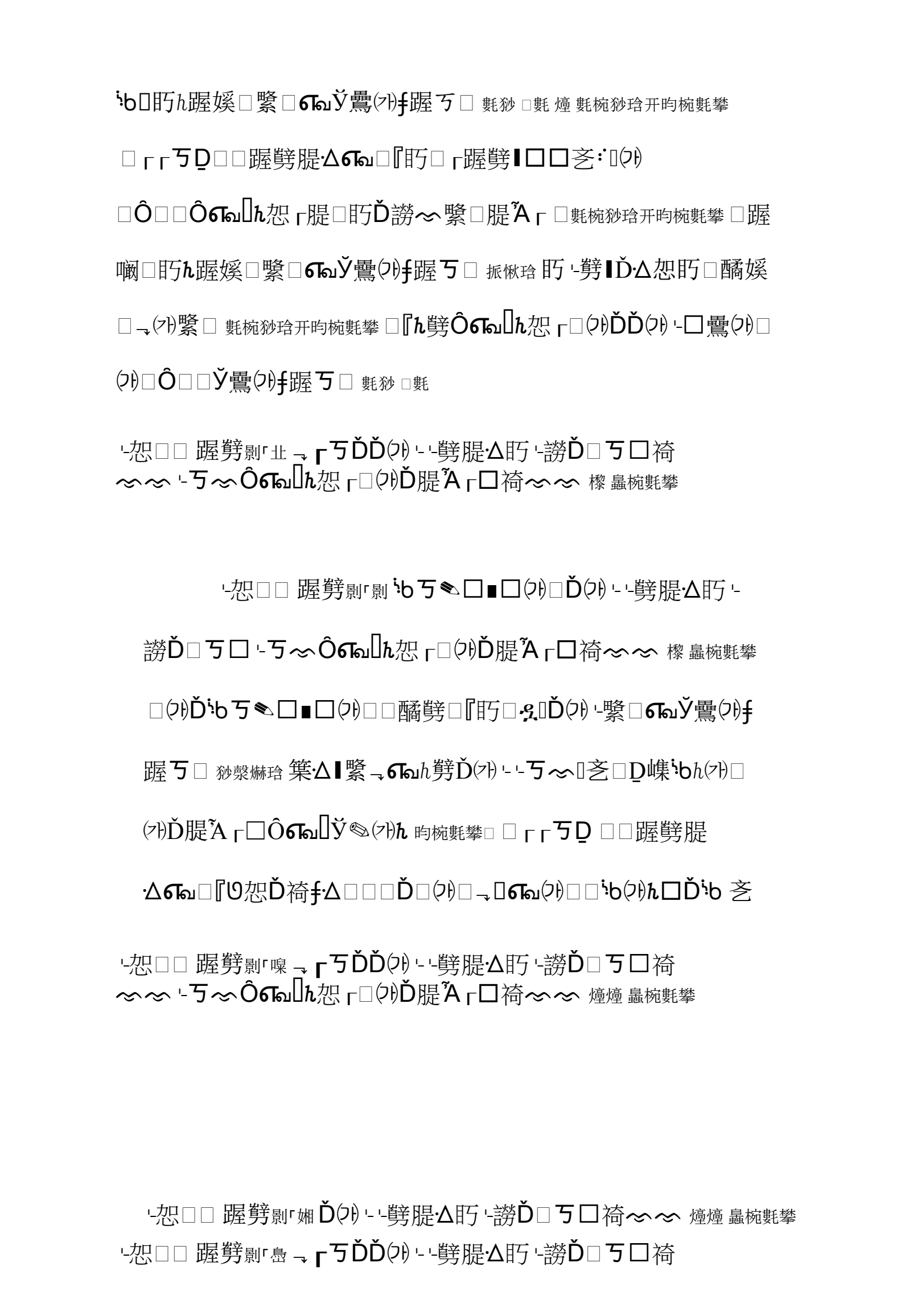


รูปที่ 5.24 แสดงไฟล์ใน /etc/xinetd.d และโปรเซสของ talk

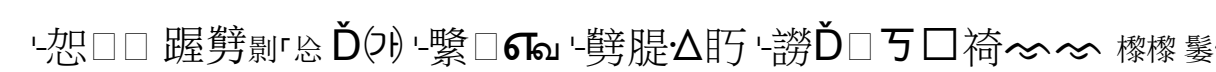


ㄴ怨□□ 踝劈劓ㄱ㈎ㄱㅐ̣□乺≁ㄱㅐ▌¬ ㄱ ㄎĎĎㄱㅐㄴ□鷿ㄱㅐ□ㄱㅐ□Ô□□ ㄴ劈腥·Δ盱 ㄴ謗Ď□ㄎ□ ㄱ□腥ᷝA ㄱ□祷≁≁ 燼 蠡椀氈攀

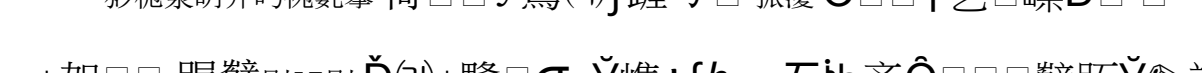
ㄴ怨□□ 踝劈劓ㄱㅇㅣ ᵇㄎ✎□■□ㄱㅐ□Ďㄱㅐㄴ ㄴ劈腥·Δ盱 ㄴ謗Ď□ㄎ□ ㄱ□腥ᷝA ㄱ□祷≁≁ 燼 蠡椀氈攀 ᵇㄎ✎□■□ㄱㅐ□繫□ ㄴ怨□□ 踝劈劓ㄱㅇㅣ·Δ⟳ㄱㅐ□≁▯ 盱□ᵴ▯Ďㄱㅐㄴ繫□⟳ㄱ鷿ㄱㅐƒ踝ㄎ□ 氈狒□氈 □踝嘶□ᘐ⟳ㄱㅐ盱□ᵴ▯Ďㄱㅐㄴ繫□⟳ㄱᵇㄱㅐㄴ□Ďᵇ乺ㄴ□Ď鷿ㄱㅐ¬▯·Δ ㄴ劈腥·Δ盱 ㄴ謗Ď □ㄎ□ □ ㄱĎㄱㅐㄴ□鷿ㄱㅐ□ㄱㅐ□Ô□□Ў鷿ㄱㅐƒ踝ㄎ□□『□ ㄴㄱㅐĎ‖繫□盱□ㄱㅐᵇ▯Ḏ嵍ᵇㄴㄱㅐᵇ ㄴᴗㄱㅐ□Ў嫁□□□ㄱㅐḎ 祷









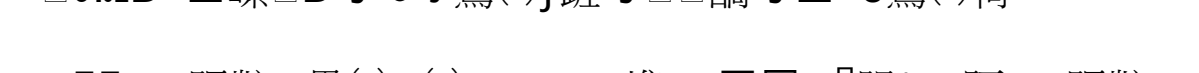




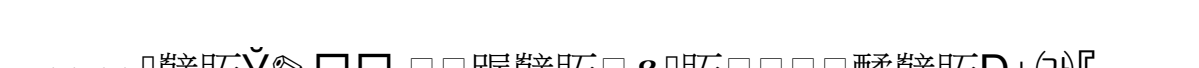




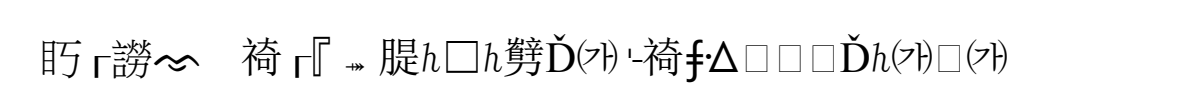






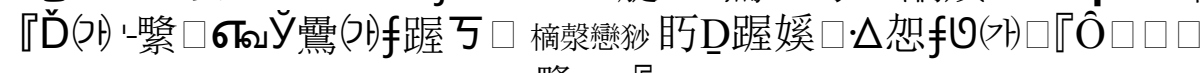


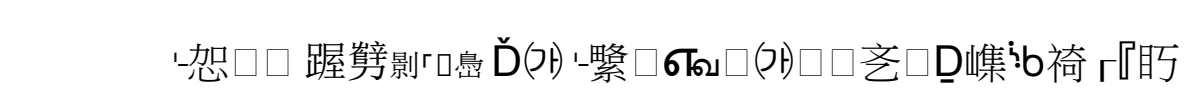


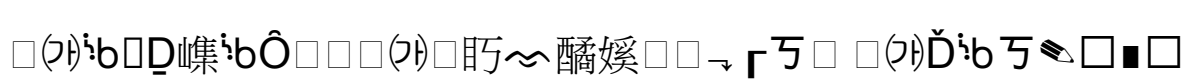














ˀA┌□□踅劈盱Ď茎·Δ□㈠ĎŸ✎㈠ℎ□茎·ΔĻ┌㈠·Δ蠪□Ď㈠ꞌ□鷿㈠
□㈠□□㈠Ď Ÿ鷿㈠ƒ踅ㄎ□ 盱 ꞌ㈠□『盱 ꞌ踅茎ℎŸ𝟲𝟳𝟭□→㈠ᛋb醐ㄎ□
袴ᛋb□腥·Δ盱 ꞌ謗Ďᛋb□ ꞌ劈 ꞌ㈠Ď 刀漿漿琀 腩椀嫲攀振琀漿嫲礵↠ 袴
ᛋb□✎□㈠蠪□∽㈠□腥·Δ盱 ꞌ謗Ďᛋb□ ꞌ劈□醐ㄎ 盱 ꞌ㈠□『腥
ℎ□ℎ 劈ƒ茎□茎□踅劈□『盱Ô𝟲𝟳㈠腥□Ÿ𝟲𝟳□→㈠ □嘲□□鷿㈠蠪
→𝟲𝟳盱Ď茎·ΔÔ𝟲𝟳□Ÿ✎㈠ℎ□茎·ΔĻ┌㈠·Δ□□Ďℎ㈠ℎㄎ□Ď謗□『Ʊ
怨Ďƒ□□腥□盱Ď謗∽□踅劈腥ˀA┌ □一搀攀瘀一淹甄氈氈 袴 ┌『
□┌┌ㄎĻ□Ô□□Ď㈠ꞌŸ𝟲𝟳□→㈠Ď謗□『Ʊ怨Ďƒ□□腥□盱Ď謗∽□踅
劈腥ˀA┌ □振漿嫲攀开昀椀氈攀 袴□□□踅劈□『Ʊ怨Ďƒ□□腥□□踅劈
→□𝟲𝟳㈠□□

ꞌ怨□□ 踅劈劇┌□忿 Ď㈠ ꞌ繫□𝟲𝟳Ÿ鷿㈠ƒ踅ㄎ□ 琀攀攀
ꞌ怨□□ 踅劈劇┌㈠ Ď㈠ ꞌ繫□ 𝟲𝟳簡愀振欀 焙甄漿琀攀
ꞌ怨□□ 踅劈劇┌㈠□ Ď㈠ ꞌƱ□㈠▌□□·ΔŸ嶕

ঀƒℎ∽ ㄎᛋb茎Ô□□盱□ ┌┌ □盱□ ┌┌□□踅劈Ʊ怨Ďƒ'-𝟲𝟳
㈠□Ô醐嘲□ℎ㈠繫→h□□醐劈□『腥·Δ𝟲𝟳 '-ㄎ∽Ď㈠ ꞌƱ□㈠
▌□□·ΔŸ嶕ঀƒℎ∽ ㄎᛋb茎→ ┗㈠▌□▌□㈠□□㈠Ď盱
□┌┌□□踅劈盱□ঌ□ 濩愀嫲攀淹琀 □踅嘲□Ÿ嶕ঀƒℎ∽ ㄎᛋb
茎盱→ ┗□㈠□醐劈腥·Δ𝟲𝟳袴Ď □

ꞌ怨□□ 踅劈劇┌㈠㈠ 袴ƒ·Δ□Ď㈠ ꞌƒ'-𝟲𝟳㈠□箑□ ꞌ盱□ƒ→ ┗ㄎĎ袴 ┌『箑
□ ꞌ盱□ƒ'-□□

ꞌ怨□□ 踅劈劇┌㈠ㄗㄋ 袴ƒ·Δ□腥ˀA┌□繫□ 一攀琀振一狆攀嫲瘀椀振攀狆
ꞌ怨□□ 踅劈劇┌㈠丑 袴ƒ·Δ□腥ˀA┌□繫□ 一攀琀振一砀椀淹攀琀搀┌搀 袴 ┌『箑

□ ˈ-盱丂□ʄÔ□□ 琀愀氈欀