เอกสารประกอบการอบรม

การใช้ ระบบปฏิบัติการ UNIX พื้นฐาน

เรียบเรียงโดย
วิบูลย์ วราสิทธิชัย
นักวิชาการคอมพิวเตอร์
กลุ่มงานบริหารจัดการระบบคอมพิวเตอร์และเครือข่าย
ศูนย์คอมพิวเตอร์ ม.อ.

จัดทำโดย
กลุ่มงานบริการวิชาการ
ศูนย์คอมพิวเตอร์ มหาวิทยาลัยสงขลานครินทร์
CC2501REV-0

บังคับใช้วันที่ 15 มี.ค. 47

คำนำ

ปัจจุบัน (พ.ศ.2546) เริ่มมีผู้ให้ความสนใจระบบปฏิบัติการลินุกซ์ (Linux) กันมากขึ้นเรื่อย ๆ ซึ่ง Linux ก็คือระบบปฏิบัติการยูนิกซ์ (UNIX) ชนิดหนึ่งที่ได้รับการพัฒนาขึ้นมาจากยูนิกซ์เวอร์ชันต่าง ๆ ที่มีมาก่อนหน้านั้น ลินุกซ์ นั้นมีรูปแบบการใช้งานที่เรียกว่า graphical user interface มีลักษณะเหมือนกับการใช้ Windows 98 ซึ่งเหมาะกับการใช้งานทั่วไป เช่น แอพพลิเคชันสำนักงาน เป็นต้น นอกจากนี้ยังมีรูปแบบการใช้งานอีกแบบซึ่งเรียกว่า textual command line interface เป็นการสั่งงานโดยเขียนชื่อคำสั่ง ซึ่งเป็นรูปแบบที่มีอยู่และเหมือนกันในระบบปฏิบัติการยูนิกซ์ทุกชนิด เหมาะสำหรับการจัดการเรื่องต่าง ๆ เช่นการจัดการไฟล์และไดเรกทอรี การค้นหาข้อมูลในไฟล์ การติดต่อไปยังเครื่องที่อยู่ห่างไกล การถ่ายโอนไฟล์ด้วยยูทิลิตี้ ftp เป็นต้น

หลักสูตร “การใช้ระบบปฏิบัติการ UNIX พื้นฐาน” นี้จะเน้นที่การเรียนรู้ระบบปฏิบัติการยูนิกซ์ให้เข้าใจถึงหลักการ ปรัชญาของยูนิกซ์ การเขียนคำสั่งต่าง ๆ ผู้เขียนได้ออกแบบบทเรียนและแบบฝึกหัดต่าง ๆ ในเอกสารนี้เพื่อให้ผู้เรียนเกิดทักษะในการใช้คำสั่งอย่างง่าย ๆในเวลาอันสั้น และเป็นจุดเริ่มต้นให้ผู้เรียนได้ไปค้นคว้าเองต่อได้

เนื้อหาประกอบด้วย

วิบูลย์ วราสิทธิชัย
ผู้เขียน

สารบัญ

บทที่ 1 เริ่มต้นอย่างง่าย

วัตถุประสงค์

เนื้อหาในบทนี้ (เวลาโดยประมาณ 3 ชั่วโมง)

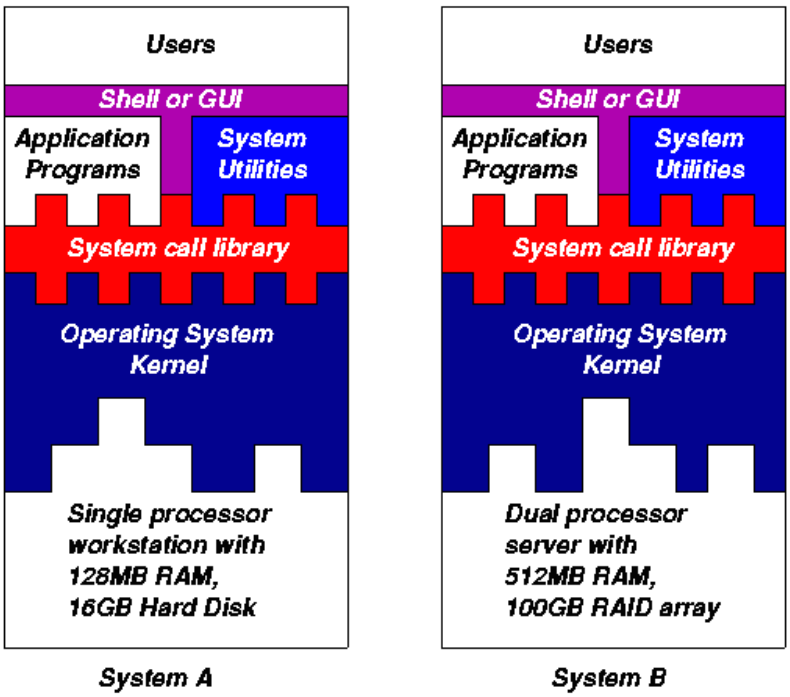
- หลักการทำงานของระบบปฏิบัติการทั่วไป
- สถาปัตยกรรมภายในระบบปฏิบัติการทั่วไป
- วิวัฒนาการของระบบปฏิบัติการจากอดีตจนมาถึงปัจจุบัน
- ยูนิกซ์คืออะไร ปรัชญาของยูนิกซ์ และองค์ประกอบของยูนิกซ์
- สถาปัตยกรรมระบบปฏิบัติการลินุกซ์
- วิธีเข้าใช้และออกจากยูนิกซ์ทำอย่างไร และวิธีเปลี่ยนรหัสผ่านของคุณ
- รูปแบบทั่วไปของคำสั่งยูนิกซ์
- Terminal Control Keys

1.1 ระบบปฏิบัติการคืออะไร

ระบบปฏิบัติการคือตัวที่ทำหน้าที่จัดสรรทรัพยากรของเครื่อง มันคือชุดคำสั่งที่จะอนุญาตผู้ใช้งานและโปรแกรมแอพพลิเคชันต่าง ๆในการขอใช้ทรัพยากรของระบบเช่น ซีพียู, หน่วยความจำ, ดิสก์, โมเด็ม, การ์ด และอื่นๆ ด้วยวิธีที่ ปลอดภัย, มีประสิทธิภาพ และมีรูปแบบที่อ้างถึงสะดวก

ยกตัวอย่าง ระบบปฏิบัติต้องแน่ใจว่ามีความปลอดภัยในการใช้เครื่องพิมพ์ โดยให้มีโปรแกรมเพียงหนึ่งโปรแกรมเท่านั้นส่งข้อมูลไปที่เครื่องพิมพ์ ณ เวลาหนึ่ง ระบบปฏิบัติการช่วยให้เกิดการใช้ซีพียูอย่างมีประสิทธิภาพโดยการหยุดให้บริการโปรแกรมที่กำลังรอคอยการทำงานของ I/O เสร็จแล้วจัดสรรซีพียูเพื่อบริการโปรแกรมอื่นๆ ในเวลาเดียวกัน ระบบปฏิบัติการจัดเตรียมรูปแบบที่ทำให้ผู้ใช้สะดวก เช่น อ้างถึงไฟล์แทนที่จะอ้างถึงตำแหน่งที่เก็บข้อมูลบนดิสก์ เป็นต้น

รูปที่ 1.1 สถาปัตยกรรมของระบบปฏิบัติการทั่วไป



รูปที่ 1.1 แสดงสถาปัตยกรรมของระบบปฏิบัติการทั่วไป และแสดงให้เห็นว่าระบบปฏิบัติมีช่องทางอย่างไรในการติดต่อกับผู้ใช้และโปรแกรม โดยไม่สนใจว่าฮาร์ดแวร์จะมีคุณลักษณะที่แตกต่างกัน เราจะเห็นได้ว่า

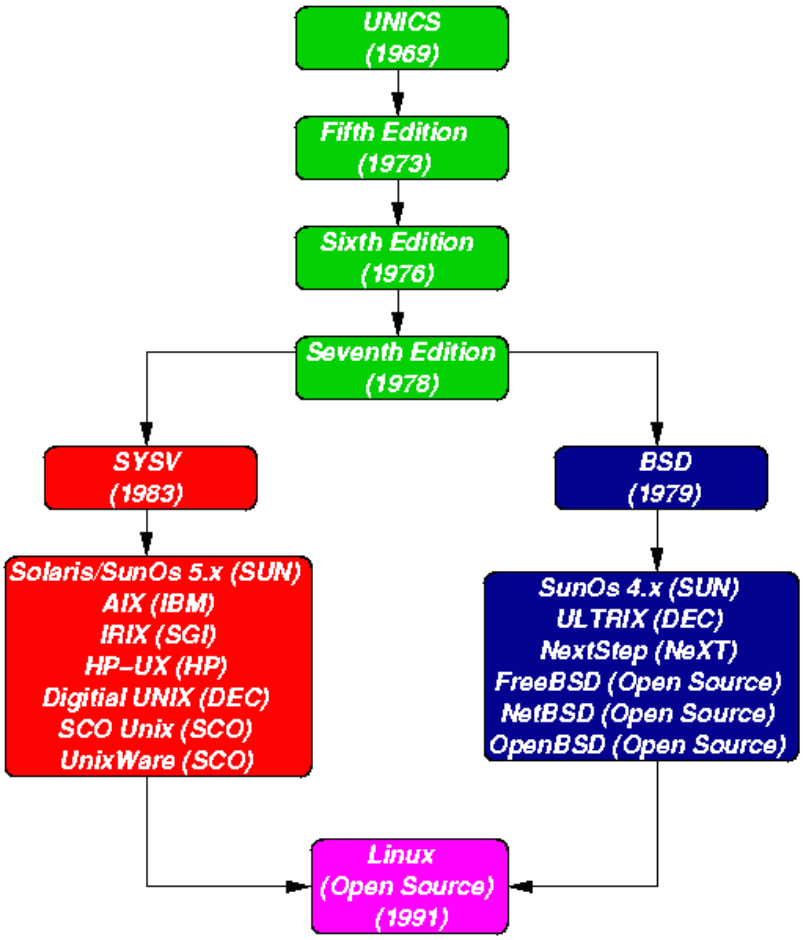
- เคอร์เนล (kernel) ของระบบปฏิบัติการทำหน้าที่โดยตรงเพื่อควบคุมฮาร์ดแวร์ ภายในเคอร์เนลจะมีฟังก์ชันในการจัดการอุปกรณ์ระดับล่าง หน่วยความจำ และหน่วยประมวลผล (ต.ย. การจัดการกับการร้องขอจากอุปกรณ์ต่างๆ, การแบ่งปันหน่วยประมวลผลให้กับโปรแกรมต่างๆในขณะที่มันทำงานพร้อมกัน, การจัดสรรหน่วยความจำให้กับโปรแกรมต่างๆ เป็นต้น)
- โปรแกรมระดับสูงจะมีช่องทางติดต่อกับบริการของเคอร์เนลผ่านทาง system call library (ต.ย. สร้างไฟล์ รันโปรแกรม หรือเปิดการติดต่อเครือข่ายไปยังเครื่องอื่น)
- พวกโปรแกรมแอพพลิเคชัน (ต.ย. เวิร์ดโปรเซสเซอร์, สเปรดชีต) และยูทิลิตี้ระบบ (เรียบง่ายแต่มีประโยชน์ที่มาพร้อมระบบปฏิบัติการ เช่น โปรแกรมที่ใช้ค้นหาข้อความภายในไฟล์หรือหลายๆไฟล์) จะใช้ system call โปรแกรมต่างๆจะถูกรันผ่านเชลล์ (shell) ซึ่งเป็น textual command line interface หรือรันผ่าน graphical user interface

ระบบปฏิบัติการหลายๆระบบ (และรุ่นต่างๆของระบบปฏิบัติการ) สามารถชี้ชัดได้ว่ามันแตก

ต่างจากระบบอื่นด้วย system call, system utilities และ user interface ที่มันมีให้ใช้

1.2 ประวัติของยูนิกซ์อย่างย่อ

ยูนิกซ์กลายเป็นระบบปฏิบัติการที่ได้รับความนิยมนานกว่า 2 ทศวรรษแล้ว เนื่องมาจากความสามารถต่างๆ ในเรื่อง มีผู้เข้าใช้งานได้หลายคนพร้อมกัน (multi-user), ทำงานได้หลายงานพร้อมกัน (multi-tasking), เสถียรภาพ (stability), การย้ายโปรแกรมที่เขียนข้ามไปสู่อีกระบบ (portability) และมีสมรรถนะสูงในการใช้งานเครือข่าย ที่จะกล่าวต่อไปคือประวัติอย่างย่อว่ายูนิกซ์ได้ถูกพัฒนามาอย่างไร



รูปที่ 1.2 ต้นตระกูลยูนิกซ์

ในปลายทศวรรษ 1960 นักวิจัยหลายคนจากบริษัท General Electric, MIT และ Bell Labs ได้ทำโครงการร่วมกันในการพัฒนาระบบปฏิบัติการที่เป็น multi-users, multi-tasking สำหรับเครื่องคอมพิวเตอร์ชนิดเมนเฟรม เรียกชื่อว่า MULTICS (Multiplexed Information and Computing System) แต่ MULTICS ไม่ประสบผลสำเร็จ อย่างไรก็ตาม ก็เป็นแรงดลใจให้คุณเคน ทอมป์สันซึ่งเป็นนักวิจัยที่ Bell Labs เขียนระบบปฏิบัติการอย่างง่ายด้วยตัวเขาเอง เขาเขียน MULTICS รุ่นที่ใช้ง่ายบนเครื่อง PDP7 ด้วยภาษา Assembly และเรียกสิ่งที่เขาพยายามทำจนเสร็จนี้ว่า UNICS (Uniplexed Information and Computing System) (ซึ่งต่อมาเรียกว่า UNIX) เพราะว่าหน่วยความจำและความสามารถของซีพียูในเวลานั้นเป็นสิ่งสำคัญอย่างมาก ดังนั้น UNICS จึงใช้คำสั่งที่สั้นเพื่อประหยัดเนื้อที่ในหน่วยความจำที่ต้องเก็บไว้และลดเวลาในการแปลคำสั่ง นั่นเป็นที่มาของคำสั่งที่สั้นๆที่เราใช้กันอยู่ในปัจจุบัน เช่น ls, cp, rm, mv และอื่นๆ

หลังจากนั้นคุณเคน ทอมป์สัน ก็ร่วมมือกับคุณเดนนิส ริทชี่ ผู้เขียนคอมไพเลอร์ภาษา C ในปี ค.ศ. 1973 พวกเขาเขียนเคอร์เนลของยูนิกซ์ขึ้นใหม่ด้วยภาษา C อันนี้เป็นก้าวสำคัญในเรื่อง portability และออกยูนิกซ์เวอร์ชัน 4 ไปใช้ในมหาวิทยาลัยต่าง ๆในปี ค.ศ. 1974 และออกเวอร์ชัน 7 ในปี ค.ศ. 1978 นั่นเป็นจุดเริ่มต้นของการแยกทีมในการพัฒนายูนิกซ์ออกเป็น 2 พวก คือ SYSV (System 5) และ BSD (Berkeley Software Distribution)

BSD เกิดขึ้นจากมหาวิทยาลัยแคลิฟอร์เนียที่เบิร์คเลย์ในขณะที่คุณเคน ทอมป์สันได้หยุดพักไป และถูกพัฒนาต่อโดยนักศึกษาที่เบิร์คเลย์และสถาบันวิจัยอื่นๆ ส่วน SYSV ได้ถูกพัฒนาโดย AT&T และบริษัทธุรกิจอีกหลายบริษัท ความนิยมยูนิกซ์ SYSV ได้รับการสนับสนุนมากกว่า BSD

สายพันธ์สุดท้ายของ SYSV (SVR4 หรือ System 5 Release 4) และ BSD เหมือนกันอย่างมาก มีความแตกต่างที่เล็กน้อยเกิดขึ้นในเรื่องของโครงสร้างระบบไฟล์ ชื่อและตัวเลือกยูทิลิตี้ระบบ และ system call library ดังแสดงในรูปที่ 1.3

รูปที่ 1.3 ความแตกต่างระหว่าง SYSV และ BSD

Feature	Typical SYSV	Typical BSD
kernel name	/unix	/vmunix
boot init	/etc/rc.d directories	/etc/rc.* files
mounted FS	/etc/mnttab	/etc/mtab
default shell	sh, ksh	csh, tcsh
FS block size	512 bytes->2K	4K->8K
print subsystem	lp, lpstat, cancel	lpr, lpq, lprm
echo command (no new line)	echo "\c"	echo -n
ps command	ps -fae	ps -aux
multiple wait syscalls	poll	select
memory access syscalls	memset, memcpy	bzero, bcopy

และในปีค.ศ. 1991 ก็เกิดลินุกซ์ (Linux) ขึ้นมา

1.4 ยูนิกซ์คืออะไร

- ยูนิกซ์คือระบบปฏิบัติการที่ใช้กับคอมพิวเตอร์ทั่วไป
- ยูนิกซ์คือระบบปฏิบัติการชนิด multi-user และ multi-tasking
- ยูนิกซ์คือระบบปฏิบัติการที่ไม่ขึ้นกับฮาร์ดแวร์
- ยูนิกซ์เกิดขึ้นมาในสภาวะแวดล้อมการทำงานที่เอื้อต่อการพัฒนาโปรแกรม

ปรัชญาของยูนิกซ์

- สร้างโปรแกรมเพื่อให้ทำงานได้ดีเป็นอย่าง ๆ เป็นเครื่องมือที่นำมาใช้ได้เรื่อย ๆ(1 tool = 1 function)
- ผลลัพธ์ของคำสั่งหนึ่งจะเป็นข้อมูลให้กับคำสั่งต่อไป รวมคำสั่งหลายๆคำสั่งเพื่อจัดการงานที่ซับซ้อน
- ทำโปรแกรมให้เล็กและทำงานได้
- ใช้คำสั่งสั้น ไม่เสียเวลาพิมพ์

ทำไมเลือกใช้ยูนิกซ์

- ระบบปฏิบัติการที่ไม่ขึ้นกับฮาร์ดแวร์ (Portability) พัฒนาขึ้นจากภาษาซี ซึ่งสามารถใช้กับ

ฮาร์ดแวร์ใดก็ได้

- สภาพแวดล้อมเหมาะสมในการพัฒนาซอฟต์แวร์ มีคำสั่งมากมายให้ใช้เป็นเครื่องมือในการพัฒนา
- มียูนิกซ์ให้ใช้มากมายในสถาบันการศึกษาที่มีคอมพิวเตอร์ประมวลผลสมรรถนะสูง
- มันสามารถทำงานแบบ Distributed processing and multi-tasking

องค์ประกอบของยูนิกซ์

- Kernel
- Shells
- Utilities

1.3 ลินุกซ์

ลินุกซ์ (Linux) เป็นระบบปฏิบัติการยูนิกซ์ชนิดโอเพนซอร์สที่แจกฟรีสำหรับเครื่องพีซี ซึ่งพัฒนาขึ้นครั้งแรกเมื่อปี ค.ศ. 1991 โดยคุณลินุส ทอร์วัลด์ส นักศึกษาชั้นปริญญาตรีชาวฟินแลนด์ ลินุกซ์ไม่เป็นทั้ง SYSV และ BSD แต่ว่ามันรวมเอาความสามารถในการทำงานจากแต่ละระบบ (ต.ย. ใช้ไฟล์เริ่มต้นระบบแบบ SYSV แต่การวางตำแหน่งไฟล์แบบ BSD เป็นต้น) และมีเป้าหมายเพื่อให้เป็นไปตามมาตรฐานการพัฒนาซอฟต์แวร์ของ IEEE ที่เรียกว่า POSIX (Portable Operating System Interface) ในการที่จะทำให้เกิด portability สูงสุด มันจึงสนับสนุนทั้ง SYSV, BSD และ POSIX system call (ต.ย. poll, select, memset, memcpy, bzero และ bcopy ทั้งหมดนี้ก็ใช้ได้)

ความเป็นโอเพนซอร์สของลินุกซ์หมายถึงว่ารหัสต้นฉบับสำหรับเคอร์เนลของลินุกซ์นั้นแจกจ่ายได้ฟรีทำให้ใครก็ได้สามารถเพิ่มขีดความสามารถและแก้ไขข้อผิดพลาดที่พบได้เองเลย วิธีการแบบนี้ประสบผลสำเร็จอย่างมากและจากโครงงานที่เริ่มต้นโดยคนหนึ่งคน กลับกลายเป็นมีอาสาสมัครจำนวนหลายร้อยคนทั่วโลกร่วมกันพัฒนา วิธีการโอเพนซอร์สนี้นอกจากจะทำให้การพัฒนาเคอร์เนลประสบผลสำเร็จแล้ว ยังรวมถึงการพัฒนาโปรแกรมแอพพลิเคชันต่างๆที่รันบนลินุกซ์อีกด้วย (ดู http://www.freshmeat.net)

จากการที่ลินุกซ์ได้รับความนิยมมากขึ้น จึงเกิดมี distribution ต่าง ๆ เกิดขึ้นมากมายไม่ว่าจะ

เป็น Redhat, Slackware, Mandrake, Debian, และ Caldera แต่ละ distribution ประกอบด้วยเคอร์เนลที่พร้อมใช้ ยูทิลิตี้ระบบ อินเตอร์เฟสแบบ GUI และโปรแกรมแอพพลิเคชัน

Redhat เป็น distribution ที่ได้รับความนิยมสูงสุด เพราะว่ามันได้ถูกนำไปใช้กับฮาร์ดแวร์ชนิดต่าง ๆ เป็นจำนวนมาก (รวมถึง Intel, Alpha และ SPARC) มันใช้ง่าย ติดตั้งง่าย มีโปรแกรมแอพพลิเคชันให้ใช้มากมายรวมทั้ง X Windows graphics system คือ GNOME และ KDE GUI และชุดออฟฟิศเช่นเดียวกับ MS-Office ก็มีด้วย

1.5 สถาปัตยกรรมของระบบปฏิบัติการลินุกซ์

ลินุกซ์มีทุกองค์ประกอบที่ระบบปฏิบัติการทั่วไปมี

- Kernel

  เคอร์เนลของลินุกซ์มีไดร์ฟเวอร์อุปกรณ์ของพีซีจำนวนมากมายให้มาด้วย (เช่น graphics cards, network cards, hard disks เป็นต้น) มีความสามารถในการจัดการหน่วยประมวลผลและหน่วยความจำขั้นสูง และรองรับการใช้งานไฟล์ประเภทต่างๆ (เช่นดอสไฟล์ในแผ่นฟลอปปี้ และ มาตรฐาน ISO9660 ที่ใช้กับ CDROM) ในส่วนให้บริการแก่โปรแกรมแอพพลิเคชันและยูทิลิตี้ระบบ มันก็ติดตั้ง BSD และ SYSV system call ไว้เกือบทั้งหมดรวมทั้ง system call ที่ระบุไว้ต้องมีตามมาตรฐาน POSIX.1

  เคอร์เนลของลินุกซ์จะอยู่ที่ไฟล์ /boot/vmlinuz ในขณะที่ซอร์สไฟล์จะอยู่ที่ /usr/src/linux ส่วนเวอร์ชันล่าสุดของเคอร์เนลของลินุกซ์สามารถดาวน์โหลดได้จาก http://www.kernel.org

- Shells และ GUI

  ลินุกซ์ให้มีการใช้งาน 2 แบบคือสั่งงานผ่านทาง shell เช่นเดียวกับยูนิกซ์อื่นๆ (ต.ย. sh – the Bourne shell, bash – the Bourne again shell และ csh – the C shell) และผ่าน GUI เช่น KDE และ GNOME Windows Manager ถ้าคุณติดต่อจากเครื่องอื่นมายังเซิร์ฟเวอร์ การเข้าใช้งานของคุณก็มักจะเป็นแบบใช้งาน shell

- System Utilities

  System Utilities ที่คุณจะพบในยูนิกซ์ตระกูลต่างๆ จะมีอยู่ในลินุกซ์แทบทั้งสิ้น เช่น

คำสั่ง ls, cp, grep, awk, sed, bc, wc, more และอื่น ๆ พวกเหล่านี้ได้ถูกออกแบบเพื่อให้เป็นเครื่องมือที่มีประสิทธิภาพสูงในการทำงานเพียง 1 อย่างได้เป็นอย่างดี (ต.ย. grep ใช้ค้นหาคำที่อยู่ในไฟล์ในขณะที่ wc ใช้นับจำนวนคำ บรรทัด และไบต์ที่อยู่ในไฟล์) ผู้ใช้สามารถแก้ปัญหาในการทำงานด้วยการนำคำสั่งเหล่านี้มาใช้งานต่อเนื่องกันแทนที่การเขียนโปรแกรมใหญ่ๆขึ้นมา 1 โปรแกรมเพื่อใช้งาน

เช่นเดียวกับยูนิกซ์ต่างๆ ลินุกซ์ก็มีโปรแกรมชนิดเซิร์ฟเวอร์ที่เรียกว่า daemons ซึ่งทำให้มีบริการสำหรับเครือข่าย (ต.ย. telnetd และ sshd ให้ผู้ใช้ล็อคอินเข้าใช้งานได้ lpd ให้บริการพิมพ์ httpd ให้บริการเว็บเพจ crond รันงานจัดการระบบที่ทำอยู่เสมออย่างอัตโนมัติ) แต่ละ daemon จะถูกปลุกให้ทำงานโดยอัตโนมัติเมื่อตอนเริ่มต้นระบบปฏิบัติการ และรอคอยการขอใช้งานจากผู้ใช้ (client)

- Application programs

  ลินุกซ์แต่ละ distribution มักจะมีโปรแกรมแอพพลิเคชันให้มาด้วย เช่น vi (editor), gcc (C compiler), g++ (C++ compiler), xfig (โปรแกรมวาดรูป), latex (ภาษาสำหรับจัดงานพิมพ์) และ soffice (StarOffice ซึ่งเป็น MS-Office clone)

  Redhat Linux ก็จะมี rpm (Redhat Package Manager) ซึ่งทำให้การติดตั้งและถอนการติดตั้งโปรแกรมทำได้อย่างง่าย ๆ

1.6 การเข้าใช้งาน (และออกจาก) ระบบปฏิบัติการยูนิกซ์

Text-based (TTY) terminal:

เมื่อคุณทำการติดต่อไปยังเครื่องคอมพิวเตอร์อื่นที่เป็นยูนิกซ์ด้วยเทอร์มินัลที่แสดงผลข้อความอย่างเดียว คุณจะพบ prompt:

```
login:
```

ที่ prompt นี้ พิมพ์ชื่อบัญชีผู้ใช้ แล้วกดปุ่ม Enter จำไว้ว่ายูนิกซ์มีความแตกต่างระหว่างตัวพิมพ์ใหญ่และตัวพิมพ์เล็ก (ต.ย. Will, WILL และ will นั้นเป็น login ต่างกัน)

หลังจากนั้นคุณจะได้ prompt สำหรับใส่รหัสผ่าน

```
login: will
password:
```

พิมพ์รหัสผ่านของคุณลงไปที่ prompt และกดปุ่ม Enter สังเกตว่ารหัสผ่านของคุณจะไม่ปรากฏขึ้นบนจอภาพในขณะที่คุณพิมพ์

ถ้าคุณพิมพ์ชื่อบัญชีผู้ใช้หรือรหัสผ่านผิด คุณจะได้รับข้อความบางอย่างแจ้งกลับมา และมันจะให้คุณใส่ใหม่โดยแสดง prompt `login:` นั้นอีกที ถ้าพิมพ์ถูกต้องคุณก็จะได้รับ shell prompt เช่น

```
$
```

เมื่อต้องการออกจาก shell ก็พิมพ์คำสั่ง “exit” ที่ shell prompt หรือถ้ามันไม่ทำงานก็พิมพ์คำสั่ง “logout” ถ้ายังไม่ทำงานอีก ให้กดปุ่ม Ctrl+d (กดปุ่ม Ctrl ค้างไว้ แล้วกดปุ่ม d และปล่อยพร้อมกัน)

Graphical terminals:

ถ้าคุณเข้าใช้เครื่องพีซีที่เป็นระบบยูนิกซ์ เช่นลินุกซ์ คุณมักจะพบการใช้งานในแบบกราฟิก ซึ่งจะมีหน้าต่างพร้อมฟิลด์ login และ password ก็ทำเช่นเดียวกับวิธีข้างบน

หลังจากเข้าใช้ได้แล้ว คุณก็จะพบกับ graphical window manager ซึ่งคล้ายกับ Microsoft Windows

ถ้าต้องการเข้าใช้งาน shell ก็ให้มองหาเมนูที่มีคำดังต่อไปนี้ “shell”, “xterm”, “console” หรือ “terminal emulator”

เมื่อต้องการออกจาก graphical window manager ให้มองหาเมนูที่มีคำว่า “Log out” หรือ “Exit”

1.7 การเปลี่ยนรหัสผ่าน

สิ่งหนึ่งที่คุณจะต้องทำเมื่อคุณเข้าใช้งานในครั้งแรกก็คือการเปลี่ยนรหัสผ่านของคุณ

คำสั่งยูนิกซ์ที่ใช้เพื่อเปลี่ยนรหัสผ่านของคุณคือ `passwd` :

```
$ passwd
```

ระบบจะมีแสดง prompt ให้คุณใส่รหัสผ่านอันเดิมที่กำลังใช้อยู่ แล้วตามด้วยรหัสผ่านใหม่ของคุณ เพื่อเป็นการลดข้อผิดพลาดจากการพิมพ์รหัสผ่านใหม่ มันจึงถามคุณให้ยืนยันรหัสผ่านใหม่นั้นอีกครั้ง

เมื่อเลือกรหัสผ่าน โปรดจำสิ่งต่าง ๆ ต่อไปนี้

- จุดอ่อนในความปลอดภัยของคอมพิวเตอร์คือรหัสผ่านของผู้ใช้ ดังนั้นเก็บรักษาไว้ให้ดี และหลีกเลี่ยงใช้คำศัพท์ในพจนานุกรม
- สร้างรหัสผ่านให้มีอย่างน้อย 7 ถึง 8 ตัวอักษร และใช้ตัวอักษรปนกับตัวเลข และเครื่องหมายต่างๆ

1.8 รูปแบบทั่วไปของคำสั่งยูนิกซ์

คำสั่งยูนิกซ์ คือ ชื่อของไฟล์ชนิด executable ที่เก็บอยู่ในระบบไฟล์ของยูนิกซ์ และส่วนมากชื่อไฟล์เหล่านี้จะเป็นอักษรตัวพิมพ์เล็ก

คำสั่งยูนิกซ์ประกอบด้วยชื่อคำสั่งตามด้วยอาร์กิวเมนต์ซึ่งแยกเป็น options และ targets ดังนี้

```
$ command -options targets
```

ตัวอย่างคำสั่งไม่มีอาร์กิวเมนต์

```
$ date
```

ตัวอย่างคำสั่งที่มี option

```
$ ls -l
```

ตัวอย่างคำสั่งที่มี target

```
$ cd newdir
```

ตัวอย่างคำสั่งที่มี option และ target

```
$ wc -l file1
```

ตัวอย่างคำสั่งที่ option หลาย ๆ option

```
$ ls -a -l -s
```

หรือเขียน option รวมกันได้

```
$ ls -als
```

ตัวอย่างคำสั่งที่มี target หลาย ๆ target

```
$ cat file1 file2 file3
```

เราสามารถเขียนคำสั่งหลายๆคำสั่งต่อเนื่องกันในหนึ่งบรรทัดโดยแยกกันด้วยเครื่องหมาย ; (semi colon)

```
$ cd newdir ; ls -l
```

และหากว่าไม่สามารถเขียนคำสั่งได้จบในหนึ่งบรรทัด ก็เชื่อมคำสั่งด้วยเครื่องหมาย \ (backslash)

```
$ cp /users/kelly/recipe kelly.recipe ; \
lpr -Pps3 #6m kelly.recipe
```

1.9 Terminal Control Keys

การกดปุ่มหลายปุ่มบนคีย์บอร์ด จะมีความหมายและมีผลกับการใช้เทอร์มินัล

ปุ่มคอนโทรล (CTRL) ใช้ร่วมกับปุ่มตัวอักษรอื่นโดยการกดปุ่มคอนโทรลค้างไว้แล้วจึงกดอีกปุ่มหนึ่ง เช่น CTRL-c หมายถึง กดปุ่ม CTRL ค้างไว้แล้วกดปุ่มตัวอักษร "c" และปล่อยพร้อมกัน

control keys ที่จำเป็นและใช้บ่อยคือ:

CTRL-u - ลบทุกตัวอักษรบนบรรทัดคำสั่ง

CTRL-c - หยุดคำสั่ง

CTRL-h - ลบตัวอักษรทางซ้าย (มันคือ backspace)

CTRL-z - ค้างคำสั่งนั้นไว้ก่อน หยุดชั่วคราว

CTRL-s - หยุดจอภาพที่กำลังแสดงผล

CTRL-q - สั่งให้จอภาพแสดงผลต่อไป

CTRL-d - ออกจากโปรแกรมหรือคำสั่ง (เป็นการส่งสัญญาณว่าจบข้อมูล)

แบบฝึกหัดท้ายบท

1. การติดต่อจากวินโดวส์ไปยังเครื่องชื่อ maliwan ซึ่งเป็นยูนิกซ์เซิร์ฟเวอร์ ใช้โปรแกรม secure shell client ชื่อ putty ติดต่อไป แล้วใส่ชื่อบัญชีผู้ใช้และรหัสผ่านที่ได้รับจากผู้สอน

   :----------:

   ดาวน์โหลดโปรแกรม putty ถ้าไม่มีอยู่ในเครื่อง

   เข้าโปรแกรมชื่อ putty แล้วติดต่อเครื่อง maliwan

   ```
   login as: sa32 ↵
   Sent username "sa32"
   sa32@maliwan.psu.ac.th's password: รหัสผ่าน ↵
   TERM = (vt100) ↵
   ```

   maliwan> รอพิมพ์คำสั่งตรงนี้

   :----------:

2. การติดต่อจากลินุกซ์ไปยังเครื่องชื่อ maliwan ซึ่งเป็นยูนิกซ์เซิร์ฟเวอร์ ใช้โปรแกรม secure shell client ชื่อ ssh ติดต่อไป แล้วใส่ชื่อบัญชีผู้ใช้และรหัสผ่านที่ได้รับจากผู้สอน

   ใส่ login และ password ของ demo (หากสร้างบัญชีผู้ใช้ชื่อ demo ไว้) เพื่อเข้าใช้เครื่องลินุกซ์

   :----------:

   ```
   login: demo ↵
   password: welcome ↵
   ```

ตอนนี้จะเข้าสู่ X manager ชื่อ gnome หรือ kde แล้วแต่ว่าเลือกตอนติดตั้งเป็นแบบใด

ออกสู่ shell โดยเลือกหาไอคอนชื่อ terminal

ใช้คำสั่ง ssh เพื่อเข้าใช้เครื่อง maliwan ดังนี้

bash$ **ssh sa32@maliwan.psu.ac.th**

ตอนนี้จะต้องพิมพ์คำว่า yes หากเป็นการติดต่อครั้งแรก จากนั้นใส่ password ของ sa32

:----------:

3. การเปลี่ยนรหัสผ่าน

:----------:

prompt> **passwd** ↵

```
passwd:  Changing password for sa32
Enter login password: ใส่รหัสอันที่ใช้อยู่ ↵
New password: ใส่รหัสที่คิดขึ้นใหม่ ↵
Re-enter new password: ยืนยันรหัสใหม่อีกครั้ง ↵
```

:----------:

กรณีเปลี่ยนสำเร็จ

เราจะเห็นข้อความนี้

```
passwd (SYSTEM): passwd successfully changed for sa32
```

prompt>

:----------:

----------:

กรณีเปลี่ยนไม่สำเร็จแบบที่ 1

```
Enter login password:
```
ใส่รหัสไม่เหมือนอันที่ใช้อยู่ ↵

เราจะเห็นข้อความนี้

```
passwd(SYSTEM): Sorry, wrong passwd
Permission denied
```

prompt>

:----------

----------:

กรณีเปลี่ยนไม่สำเร็จแบบที่ 2

```
Enter login password:
```
ใส่รหัสอันที่ใช้อยู่ ↵

```
New password:
```
ใส่รหัสที่คิดขึ้นใหม่ ↵

```
Re-enter new password:
```
ยืนยันรหัสใหม่อีกครั้งไม่เหมือน ↵

เราจะเห็นข้อความนี้

```
passwd(SYSTEM): They don't match; try again.
```

:----------

----------:

กรณีเปลี่ยนไม่สำเร็จแบบที่ 3

```
New password:
```
ใส่รหัสที่คิดขึ้นใหม่มีความยาวสั้นกว่า 6 ตัว ↵

```
passwd(SYSTEM): Password too short - must be at least 6 characters.
New password:
```

:----------

----------:

กรณีเปลี่ยนไม่สำเร็จแบบที่ 4

```
New password: ใส่รหัสที่คิดขึ้นใหม่มีความยาว 6 ตัวแต่ไม่มีเลขปน ↵
passwd(SYSTEM): The first 6 characters of the password
must contain at least two alphabetic characters and at least
one numeric or special character.
New password:
```

```
:----------------------------------------------------------------------------
----------------:
```

4. การใช้คำสั่ง ผู้เรียนป้อนคำสั่งเหล่านี้ที่ prompt และลองแปลความผลลัพธ์ (อย่ากลัวที่จะลองผิดลองถูก เพราะว่าผู้ใช้ทั่วไปจะไม่สามารถทำเครื่องยูนิกซ์พังได้)

```
maliwan> set prompt="prompt> " ↵
prompt>
prompt> echo hello world ↵
hello world
prompt> date ↵
Thu Jun 12 22:04:22 ICT 2003
prompt> hostname ↵
maliwan
prompt> arch ↵
sun4
prompt> uname -a ↵
SunOS maliwan 5.6 Generic_105181-16 sun4u sparc SUNW,Ultra-Enterprise
prompt> dmesg | more ↵
...
root on /sbus@3,0/SUNW,fas@3,8800000/sd@0,0:a fstype ufs
```

```
--More--
...
```

prompt> **uptime** ↵

```
 11:06pm  up 1 day(s), 21:55,  4 users, load average: 0.40, 0.43, 0.36
```

prompt> **who am i** ↵

```
sa32       pts/3        Jun 12 20:51    (wiboon.cc.psu.ac.th)
```

prompt> **who** ↵

```
root       console      Jun 11 09:19
s4125122   pts/1        Jun 12 21:30    (libra.psu.ac.th)
s4504082   pts/2        Jun 12 21:36    (libra.psu.ac.th)
sa32       pts/3        Jun 12 20:51    (wiboon.cc.psu.ac.th)
```

prompt> **id** ↵

```
uid=10032(sa32) gid=8000(trainee)
```

prompt> **last** ↵

จะมีจำนวนบรรทัดออกมามาก ให้กดปุ่ม Ctrl + C เพื่อหยุด

prompt> **last sa32** ↵

```
sa32      pts/3    wiboon.cc.psu.ac Thu Jun 12 20:51   still logged in
sa32      pts/3    wiboon.cc.psu.ac Thu Jun 12 20:30 - 20:51  (00:21)
sa32      pts/7    wiboon.cc.psu.ac Thu Jun 12 15:00   still logged in
sa32      pts/4    wiboon.cc.psu.ac Sat Jan 18 10:35 - down  (3+14:35)
wtmp begins Thu Sep 13 11:55
```

prompt> **finger** ↵

```
Login       Name           TTY     Idle When      Where
root     Super-User        console 1d   Wed 09:19
s4125122 *************** *pts/1         Thu 21:30 libra.psu.ac.th
sa32     Student_group_A  pts/3         Thu 20:51 wiboon.cc.psu.ac.th
```

prompt> **w** ↵

```
 11:16pm  up 1 day(s), 22:06,  3 users,  load average: 0.38, 0.42,
0.40
User     tty           login@  idle   JCPU   PCPU  what
root     console      Wed 9am 2days                bash
s4125122 pts/1         9:30pm            2      2  slirp ppp
```

```
sa32     pts/3         8:51pm     3     25         w
```

prompt> **who are we** ↵

```
root       console      Jun 11 09:19
s4125122   pts/1        Jun 12 21:30    (libra.psu.ac.th)
sa32       pts/3        Jun 12 20:51    (wiboon.cc.psu.ac.th)
```

prompt> **top** ↵

```
/usr/local/bin/top: Permission denied
```

prompt> **echo $SHELL** ↵

```
/bin/csh
```

prompt> **echo {con,pre}{sent,fer}{s,ed}** ↵

```
consents consented confers confered presents presented prefers
prefered
```

prompt> **man copy** ↵

```
No manual entry for copy.
```

prompt> **man ls** ↵

กดปุ่ม `q` เพื่อหยุด

prompt> **lost** ↵

```
lost: Command not found
```

prompt> **clear** ↵

prompt> **cal 2000** ↵

prompt> **cal 9 1752** ↵

สังเกตสิ่งผิดปกติ

prompt> **bc -l** ↵

พิมพ์ **10/7** ↵

`1.42857142857142857142` กดปุ่ม `Ctrl+d` เพื่อหยุด

prompt> **echo 5+4** ↵

```
5+4
```

prompt> **echo 5+4 | bc -l** ↵

```
9
```

prompt> **time sleep 5** ↵

```
0.0u 0.0s 0:05 0% 0+0k 0+0io 0pf+0w
```

prompt> **echo hello ; echo world** ↵

```
hello
world
```

prompt> **date ; time** ↵

```
Fri Jun 13 00:40:19 ICT 2003
2.0u 4.0s 39:32 0% 0+0k 0+0io 0pf+0w
```

prompt> **history** ↵

prompt> **!40** ↵

prompt> **!!** ↵

prompt> **Who** ↵

```
Who: Command not found
```

prompt> **which who** ↵

```
/usr/bin/who
```

prompt> **who** ↵

prompt> **/usr/bin/who** ↵

บทที่ 2 ระบบไฟล์

วัตถุประสงค์

เนื้อหาในบทนี้ (เวลาโดยประมาณ 3 ชั่วโมง)

- ระบบไฟล์และโครงสร้างไดเรกทอรีของยูนิกซ์
- คำสั่งจัดการไฟล์และไดเรกทอรี
- วิธีสร้างลิงค์ไฟล์แบบ hard link และ symbolic link
- การใช้ wildcard ในชื่อไฟล์
- สิทธิในไฟล์และไดเรกทอรี และคำสั่งในการเปลี่ยนสิทธิ
- การใช้งานแผ่นฟลอปปี้ดิสก์และซีดีรอม

2.1 ระบบไฟล์ของยูนิกซ์

ระบบปฏิบัติการยูนิกซ์ถูกสร้างด้วยหลักการของระบบไฟล์ ซึ่งถูกใช้เพื่อเก็บข้อมูลทั้งหมดที่ประกอบกันอย่างเป็นระบบระบบที่ว่านี้รวมถึงเคอร์เนลของระบบปฏิบัติการเอง ไฟล์ต่างๆที่เป็นคำสั่งที่ใช้ในระบบปฏิบัติการ ข้อมูลการทำคอนฟิก ไฟล์ทำงานชั่วคราว ข้อมูลผู้ใช้ และไฟล์พิเศษหลากหลายชนิดที่ใช้เพื่อควบคุมฮาร์ดแวร์และฟังก์ชันของระบบปฏิบัติการ

แต่ละรายการที่ถูกเก็บอยู่ในระบบไฟล์ยูนิกซ์นั้น จะจัดเข้าเป็นชนิดใดชนิดหนึ่งใน 4 ชนิดนี้

1. Ordinary files

   ไฟล์ธรรมดาสามารถเก็บข้อความ ข้อมูล หรือข้อมูลของโปรแกรม ไฟล์เหล่านี้ไม่สามารถเก็บไฟล์หรือไดเรกทอรีอื่นๆไว้ภายใน

2. Directories

   ไดเรกทอรีคือที่เก็บไฟล์และไดเรกทอรีต่างๆ

3. Devices

   เพื่อช่วยให้แอพพลิเคชันติดต่อกับอุปกรณ์ฮาร์ดแวร์ได้ง่าย ยูนิกซ์ทำให้มองดีไวซ์เหมือนเป็นไฟล์ธรรมดา มันมีอยู่ 2 ประเภทคือ block-oriented ซึ่งส่งข้อมูลเป็นบล็อก (เช่น

ฮาร์ดดิสก์) และ character-oriented ซึ่งส่งข้อมูลทีละไบต์ (เช่น โมเด็ม และเทอร์มินัล)

4. Links

ลิงค์คือตัวชี้ไปยังไฟล์อื่น มันมีอยู่ 2 ประเภทคือ hard link และ soft link

2.2 ชื่อไฟล์

ยูนิกซ์ไม่แบ่งชื่อไฟล์ออกเป็นส่วนชื่อและส่วนขยาย เช่นเดียวกับระบบปฏิบัติการอื่นๆ มันสามารถเป็นตัวอักษรใดก็ได้บนคีย์บอร์ด และมีความยาวได้ถึง 256 ตัวอักษร

- ยูนิกซ์อนุญาตให้ใช้ตัวอักษรได้เกือบทุกตัว ยกเว้นเครื่องหมาย “/” แต่แนะนำว่าควรหลีกเลี่ยง ช่องว่าง แท็บ การใส่ช่องว่างลงในชื่อไฟล์ก็จะทำให้ยากแก่การจัดการไฟล์ในภายหลัง ใช้เครื่องหมายขีดล่าง “_” แทนจะดีกว่า
- หลีกเลี่ยงตัวอักษรที่มีความหมายเฉพาะเมื่อใช้ในบรรทัดคำสั่ง shell เช่น

```
& * ? # ; ( ) | \ ' " ` [ ] { } < > $ - ! /
```

- ชื่อที่เป็นภาษาอังกฤษ อักษรตัวพิมพ์ใหญ่ กับ ตัวพิมพ์เล็ก จะถือว่าไม่ใช่ตัวเดียวกัน เช่น

```
NOVEMBER     November   november
```

- ไฟล์ซ่อน (hidden file) จะขึ้นต้นด้วยเครื่องหมาย . (dot) เช่น

```
.cshrc   .login   .mailrc   .mwmrc
```

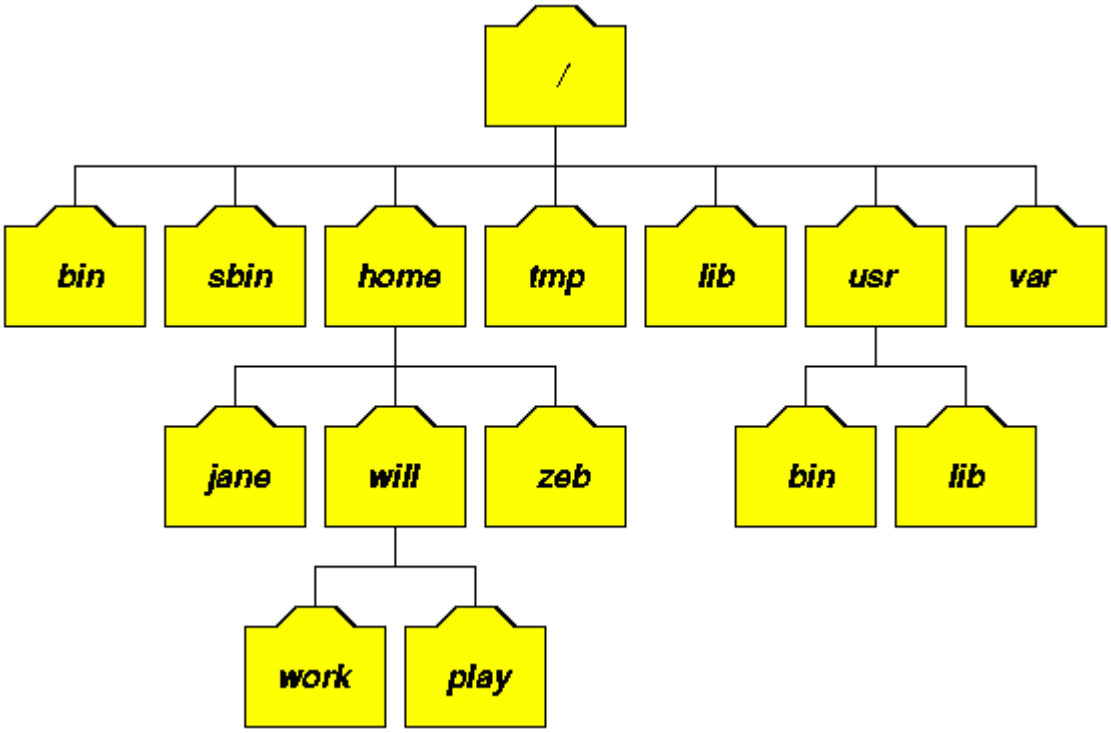
ตัวอย่างตัวอักษรพิเศษ ที่ใช้ในบรรทัดคำสั่ง

`|`	ไปป์ ส่งผ่านผลลัพธ์ของคำสั่งหนึ่งให้อีกคำสั่งหนึ่ง
`$`	อ้างถึงตัวแปรเชลล์
`{ }`	จัดกลุ่มคำสั่งภายใน function
`&`	รันคำสั่งเป็น background

2.3 โครงสร้างไดเรกทอรีของยูนิกซ์

ระบบไฟล์ของยูนิกซ์จัดวางรูปแบบคล้ายกับภาพต้นไม้กลับหัว มีราก “/” อยู่ส่วนบนสุด เพราะ

ว่ามันเป็นโครงสร้างต้นไม้ ดังนั้นไดเรกทอรีแต่ละอันจึงสามารถมีไดเรกทอรีลูกได้หลายอัน แต่ไดเรกทอรีลูกจะมีพ่อแม่เดียว รูปที่ 2.1 แสดงโครงสร้างที่กล่าวนี้



รูปที่ 2.1 ส่วนหนึ่งของระบบไฟล์แบบต้นไม้ของยูนิกซ์

เพื่อที่จะระบุตำแหน่งในโครงสร้างไดเรกทอรี เราต้องระบุเส้นทางแบบต้นไม้ เส้นทางไปยังตำแหน่งใด ๆ สามารถเขียนแบบเส้นทางสมบูรณ์เริ่มจากราก “/” หรือเขียนแบบเส้นทางสัมพันธ์เริ่มจากไดเรกทอรีที่กำลังใช้งานอยู่ในขณะนั้น ในการเขียนพาธ (path) ไดเรกทอรีแต่ละอันตลอดเส้นทางจากจุดเริ่มต้นไปถึงจุดหมายต้องเขียนไว้ในพาธ โดยคั่นด้วยเครื่องหมาย สแลช “/” เพื่อช่วยให้เราสามารถเขียนแบบสัมพันธ์ ยูนิกซ์จึงกำหนดเครื่องหมาย “.” แทนไดเรกทอรีปัจจุบัน และ “..” แทนไดเรกทอรีที่อยู่เหนือขึ้นไป ตัวอย่างเช่น การเขียนพาธแบบสมบูรณ์ไปยังไดเรกทอรี “play” คือ `/home/will/play` ในขณะเดียวกันการเขียนพาธแบบสัมพันธ์ไปยังไดเรกทอรีนี้จาก “zeb” คือ `../will/play`

ตารางที่ 2.1 ข้างล่างนี้แสดงไดเรกทอรีสำคัญบางส่วนที่คุณจะพบบนระบบปฏิบัติการยูนิกซ์และคำอธิบายสั้นๆ

Directory	Typical Contents
/	The "root" directory
/bin	Essential low-level system utilities
/usr/bin	Higher-level system utilities and application programs

Directory	Typical Contents
/sbin	Superuser system utilities (for performing system administration tasks)
/lib	Program libraries (collections of system calls that can be included in programs by a compiler) for low-level system utilities
/usr/lib	Program libraries for higher-level user programs
/tmp	Temporary file storage space (can be used by any user)
/home or /homes	User home directories containing personal file space for each user. Each directory is named after the login of the user.
/etc	UNIX system configuration and information files
/dev	Hardware devices
/proc	A pseudo-filesystem which is used as an interface to the kernel. Includes a sub-directory for each active program (or process).

ตารางที่ 2.1 ไดเรกทอรีสำคัญของยูนิกซ์

เมื่อคุณเข้าใช้งานยูนิกซ์ ไดเรกทอรีทำงานในขณะนั้นก็คือ home directory ของคุณ คุณสามารถอ้างถึง home directory ด้วยเครื่องหมาย “~” และอ้างถึง home directory ของผู้อื่นด้วย “~<login>” ดังนั้น `~will/play` ก็เป็นอีกวิธีหนึ่งที่ผู้ใช้ jane จะเขียนพาธแบบสมบูรณ์แทน `/home/will/play` ส่วนผู้ใช้ will ก็อาจจะเขียนเป็น `~/play`

2.4 คำสั่งจัดการไดเรกทอรีและไฟล์

เนื้อหาในตอนนี้จะอธิบายถึงคำสั่งชุดหนึ่งที่สำคัญมากในการจัดการไดเรกทอรีและไฟล์

- **pwd** (print [current] working directory)

  `pwd` แสดงพาธแบบสมบูรณ์ของไดเรกทอรีที่กำลังทำงาน ดังนั้น

  ```
  $ pwd
  /usr/bin
  ```

  บอกเราว่า `/usr/bin` คือไดเรกทอรีที่กำลังทำงาน

- **ls** (list directory)

  `ls` แสดงรายชื่อที่อยู่ภายในไดเรกทอรี ถ้าไม่ใส่ชื่อไดเรกทอรีต่อท้ายคำสั่ง ก็จะเป็นการแสดงรายชื่อที่อยู่ภายในไดเรกทอรีที่กำลังทำงาน ดังนั้นถ้าไดเรกทอรีที่กำลังทำงานคือ /

  ```
  $ ls
  bin   dev  home  mnt   share  usr  var
  boot  etc  lib   proc  sbin   tmp  vol
  ```

  จริง ๆ แล้ว `ls` ไม่ได้แสดงสิ่งที่อยู่ภายในไดเรกทอรีทั้งหมด ยังมีไฟล์และไดเรกทอรีที่ชื่อของ

มันขึ้นต้นด้วยเครื่องหมายจุด “.” ถูกซ่อนไว้ (รวมทั้งไดเรกทอรี “.” และ “..”) เหตุผลในเรื่องนี้ก็คือไฟล์ที่ขึ้นต้นด้วย “.” มักจะเป็นค่าคอนฟิกกูเรชันสำคัญ และไม่ควรถูกแก้ไขไม่ว่ากรณีใด ๆ แต่ถ้าคุณต้องการจะเห็นไฟล์ทั้งหมด ก็ใช้ `ls` ร่วมกับตัวเลือก `-a`

```
$ ls -a
```

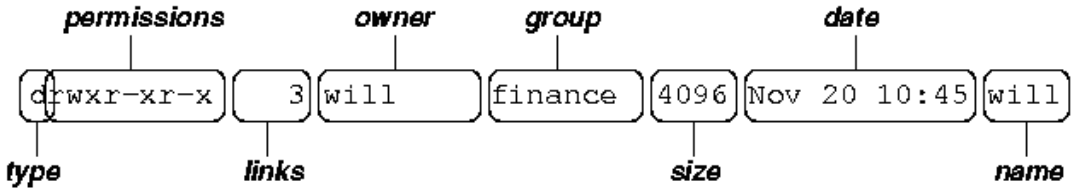
แต่ก็ยังไม่ได้ช่วยให้ข้อมูลอะไรมากนัก ไม่มีอะไรที่แสดงคุณสมบัติของไฟล์เช่น ขนาด ชนิด และความเป็นเจ้าของไฟล์ เลย แค่ชื่อไฟล์เท่านั้น เราสามารถใช้ตัวเลือก `-l` (long listing) เพื่อให้เห็นข้อมูลที่มากขึ้น ร่วมกับตัวเลือก `-a` ดังนี้

```
$ ls -a -l
```

หรือ

```
$ ls -al
```

แต่ละบรรทัดของผลลัพธ์ จะดูคล้ายข้างล่างนี้



เมื่อ:

- type จะเป็นตัวอักษร 1 ตัว “`d`” (directory), “`-`” (ordinary file), “`l`” (symbolic link), “`b`” (block-oriented device) หรือ “`c`” (character-oriented device)
- permissions จะเป็นตัวอักษรชุดหนึ่งบอกสิทธิการใช้ ประกอบด้วยตัวอักษร 9 ตัว มีสิทธิการใช้งานอยู่ 3 ชนิดให้กับแต่ละชนิดของผู้ใช้ 3 ชนิด สิทธิการใช้งาน 3 ชนิดคือ “`r`” (read), “`w`” (write), และ “`x`” (execute) และชนิดของผู้ใช้ 3 ชนิดคือ `user` คือ เจ้าของไฟล์, `group` คือผู้ใช้ในกลุ่ม, `other` คือผู้ใช้อื่นๆ ตัวอักษร “`r`”, “`w`”, “`x`” ปรากฏขึ้นแสดงว่ามีการกำหนดสิทธิ ถ้าเห็น “`-`” แสดงว่าไม่กำหนดสิทธิ
- links หมายถึงจำนวนลิงค์ที่ชี้มายังไฟล์/ไดเรกทอรีนี้ (ดูเรื่องลิงค์ในบทต่อไป)
- owner คือชื่อผู้ใช้ที่เป็นคนสร้างไฟล์นั้นขึ้นมา
- group บอกให้ทราบว่าถ้าเป็นผู้ใช้ในกลุ่มนี้ก็ใช้ไฟล์ได้ตามสิทธิที่ระบุในคอลัมน์

permissions

- size บอกความยาวของไฟล์ หรือจำนวนไบต์ที่ใช้โดยระบบปฏิบัติการในการเก็บรายชื่อไฟล์ที่อยู่ในไดเรกทอรี
- date คือวันที่แก้ไขไฟล์ครั้งหลังสุด (เขียน) ตัวเลือก -u จะแสดงเวลาเมื่อไฟล์ถูกใช้งาน (อ่าน)
- name คือชื่อไฟล์หรือชื่อไดเรกทอรี

`ls` ยังมีตัวเลือกอื่น ๆ อีก คุณสามารถอ่านได้จากคำสั่ง man ls

- **man**

man เป็นคู่มือผู้ใช้ออนไลน์ แบบย่อ และยังมีอีกแบบคือ `info` ถ้าได้รับการติดตั้งให้ใช้งาน

```
$ man ls (info ls)
```

- **cd** (change [current working] directory)

```
$ cd path
```

เปลี่ยนไดเรกทอรีทำงานไปยัง *path* (ทั้งแบบสมบรูณ์และแบบสัมพันธ์) พาธแบบสัมพันธ์ที่เราใช้บ่อยมากคือ “..”

เช่น

```
$ cd /usr/local    - change to /usr/local
$ cd doc/training  - change to doc/training in current directory
$ cd ..            - change to parent directory
$ cd ~/data        - change to data directory in home directory
$ cd ~joe          - change to user joe's home directory
```

เมื่อใช้คำสั่งโดยไม่ระบุไดเรกทอรีเป้าหมาย

```
$ cd
```

จะเป็นการรีเซตไดเรกทอรีทำงานกลับไปยัง home directory ของคุณ (เมื่อคุณหลงทาง)

ถ้าคุณเปลี่ยนเข้าไปไดเรกทอรีหนึ่ง และคุณต้องการกลับไปที่จุดเริ่มต้น ใช้คำสั่งนี้

```
$ cd -
```

- **mkdir** (make directory)

```
$ mkdir directory
```

สร้างไดเรกทอรีย่อยชื่อ *directory* ในไดเรกทอรีทำงานขณะนั้น คุณสามารถสร้างไดเรกทอรี

ย่อยได้ถ้าคุณได้มีสิทธิการเขียนที่ไดเรกทอรีนั้น

เช่น

```
$ mkdir /u/training/data
$ mkdir data2
```

- **rmdir** (remove directory)

```
$ rmdir directory
```

ลบไดเรกทอรีย่อยชื่อ ในไดเรกทอรีทำงานขณะนั้น คุณสามารถลบไดเรกทอรีย่อยนั้นได้ถ้ามันว่าง (ไม่มีไฟล์และไดเรกทอรีย่อยเหลืออยู่ในนั้นแล้ว)

เช่น

```
$ rmdir /u/training/data
```

- **cp** (copy)

cp ใช้สำหรับทำสำเนา (คัดลอก) ไฟล์หรือไดเรกทอรี ใช้คำสั่งนี้

```
$ cp source-file(s) destination
```

เมื่อ *source-file(s)* คือต้นทาง และ *destination* คือเป้าหมายปลายทาง คำสั่งนี้มันจะดูที่ว่าปลายทางเป็นไฟล์หรือไดเรกทอรี ถ้าปลายทางเป็นไฟล์ ต้นทางระบุชื่อไฟล์ได้ชื่อเดียว และคำสั่งนี้ก็จะสร้างไฟล์ชื่อ *destination* นั้นขึ้นมาซึ่งจะมีทุกอย่างเหมือนไฟล์ต้นทาง แต่ถ้าปลายทางเป็นไดเรกทอรี ต้นทางก็สามารถระบุชื่อไฟล์หลาย ๆ ไฟล์ได้ ไฟล์แต่ละไฟล์นั้นจะถูกคัดลอกไปใส่ในไดเรกทอรี ในเนื้อหาต่อไปจะพูดถึงการระบุชื่อไฟล์โดยใช้ตัวอักษร wildcard เพื่อให้ทำงานได้มีประสิทธิภาพขึ้น

การคัดลอกทั้งไดเรกทอรี ก็ทำได้ด้วยตัวเลือก -Rd (recursive) ดังนี้

```
$ cp -Rd source-file(s) destination
```

เช่น

```
$ cp sample.f sample2.f     - copies sample.f to sample2.f
$ cp -Rd dir1 dir2          - copies contents of directory dir1
                              to dir2
$ cp -i file.1 file.new     - prompts if file.new will be
                              overwritten
$ cp /etc/passwd ~          - copies file to your home directory
$ cp ~betty/index .         - copies the file "index" from user
                              betty's home directory to current
                              directory
```

- **mv** (move/rename)

`mv` ใช้เพื่อเปลี่ยนชื่อไฟล์/ไดเรกทอรี หรือเพื่อย้ายมันจากไดเรกทอรีหนึ่งไปอีกที่หนึ่ง โดยระบุต้นทาง และปลายทาง ด้วย

```
$ mv source destination
```

ถ้า *destination* เป็นชื่อไดเรกทอรี ก็จะเป็นการย้าย *source* ไปอยู่ในไดเรกทอรี ถ้าทั้ง *source* และ *destination* เป็นชื่อไฟล์ ก็จะเป็นการเปลี่ยนชื่อไฟล์จาก *source* เป็น *destination* ถ้าไปซ้ำกับชื่อไฟล์ที่มีอยู่แล้ว ก็จะเป็นการเขียน *source* ทับ *destination* (สามารถใช้ตัวเลือก -i เพื่อให้มีการถามให้ยืนยันการทำงาน)

เช่น

```
$ mv sample.f sample2.f     - moves sample.f to sample2.f
$ mv dir1 newdir/dir2       - moves contents of directory dir1
to                            newdir/dir2
$ mv -i file.1 file.new     - prompts if file.new will be
                              overwritten
```

- **rm** (remove/delete)

```
$ rm target-file(s)
```

ลบไฟล์ตามที่ระบุชื่อนั้น มันแทบจะไม่มีทางกู้ไฟล์กลับคืนมาเลยนอกจากเอาจากม้วนเทปที่ได้แบ็คอัพข้อมูลเก็บไว้ ซึ่งเป็นข้อแตกต่างจากระบบปฏิบัติการอื่น ๆ (ไม่มี recycle bin) ดังนั้นให้ระมัดระวังการใช้คำสั่งนี้ ถ้าคุณต้องการให้มีการถามย้ำก่อนลบ ก็ให้ใช้ตัวเลือก `-i` ดังนี้

```
$ rm -i myfile
rm: remove 'myfile'?
```

`rm` ยังใช้เพื่อลบไดเรกทอรีด้วย (รวมถึงไฟล์และไดเรกทอรีที่อยู่ข้างในด้วย) โดยใช้ตัวเลือก `-r` ถ้าไม่ต้องการให้มันถามย้ำว่าจะลบไฟล์/ไดเรกทอรีนั้นไหม ให้ใส่ตัวเลือก `-f` ด้วย แต่การใช้คำสั่งนี้ต้องเพิ่มความรอบครอบเป็นพิเศษ ลองดูตัวอย่างนี้ เมื่อผู้ดูแลระบบใช้คำสั่งลบไดเรกทอรีของ will แต่พิมพ์ผิดเป็น

```
$ rm -rf / home/will
```

แทนที่จะลบ home directory ของ will ก็จะเป็นการลบไดเรกทอรีราก (/) ก็เลยหมดเกลี้ยง

- **cat** (catenate/type)

```
$ cat target-file(s)
```

แสดงสิ่งที่เก็บอยู่ใน *target-file(s)* บนจอภาพทีละไฟล์

เช่น

```
$ cat hello.txt     - shows contents in hello.txt
$ cat -b myprog.c   - shows line numbers
$ cat file1 file2   - diplays entire file1 then file2
```

คุณสามารถใช้คำสั่งนี้เพื่อสร้างไฟล์จากคีย์บอร์ดซึ่งเป็นข้อมูลเข้า ดูตัวอย่าง (เครื่องหมาย > คือตัวที่ใช้ในการเปลี่ยนทิศทางผลลัพธ์ อธิบายละเอียดในบทต่อไป)

เช่น

```
$ cat > hello.txt
hello world!
[ctrl-d]

$ cat hello.txt
hello world!
$
```

- **more** (catenate with pause)

```
$ more target-file(s)
```

แสดงสิ่งที่เก็บอยู่ใน *target-file(s)* บนจอภาพทีละไฟล์ และหยุดหน้าจอไว้ให้ผู้ใช้อ่านเมื่อข้อมูลเต็มจอภาพ และรอให้ผู้ใช้กดคีย์ใด ๆ เพื่อดูส่วนที่เหลือต่อไป และใช้ค้นหาคำที่ต้องการได้ด้วย (กดคีย์ "/" และพิมพ์คำที่ค้นหา)

เช่น

```
$ more sample.f
```

นอกจากนี้ คุณยังใช้ more เพื่อใช้หยุดหน้าจอผลลัพธ์ของคำสั่งอีกคำสั่งที่มีข้อมูลมากกว่า 1 จอภาพดังเช่นตัวอย่างนี้ (เครื่องหมาย | คือตัวไปป์ อธิบายละเอียดในบทต่อไป)

```
$ ls -l | more
```

2.5 วิธีสร้างลิงค์ไฟล์ แบบ Hard link และ Soft link

คุณสามารถใช้คำสั่ง ln เพื่อสร้าง hard link ดังนี้

```
$ ln filename linkname
```

สร้างข้อมูลอีกอันสำหรับ *filename* โดยตั้งชื่อว่า *linkname* ตอนนี้ทั้ง *filename* และ *linkname* จะเป็นอันเดียวกัน ต่างก็มี link count มีค่าเป็น 2 ถ้าแก้ไข *filename* หรือ

linkname ผลลัพธ์จะกระทบถึงอีกไฟล์หนึ่ง (เพราะว่าจริง ๆ แล้วทั้ง 2 ต่างชี้ไปยังไฟล์เดียวกัน)

เช่น

```
$ ln results.1 last.run  - links filename "last.run" to the
real                       file results.1 in the current
                           directory.
$ ln ../Notes.jan notes  - links filename "notes" in current
                           directory to real file Notes.jan in
                           parent directory.
```

คุณสามารถใช้คำสั่ง `ln` เพื่อสร้าง soft link ดังนี้

```
$ ln -s filename linkname
```

เป็นการสร้างชอร์ตคัต (shortcut) ชื่อว่า *linkname* ชอร์ตคัตจะเป็นไฟล์ชนิดพิเศษ (“l”)

```
$ ln -s hello.txt bye.txt
$ ls -l bye.txt
lrwxrwxrwx   1 will finance 13 bye.txt -> hello.txt
$
```

link count ของไฟล์ต้นทางจะไม่เปลี่ยน

2.6 การใช้ wildcard ในชื่อไฟล์

เราสามารถอ้างถึงชื่อไฟล์หลายไฟล์ได้ด้วยตัวอักษรสำหรับเปรียบเทียบแบบที่เรียกว่า wildcard ซึ่งมีกฎดังนี้

- “`?`” ใช้เปรียบเทียบตัวอักษร 1 ตัวในตำแหน่งนั้นในชื่อไฟล์
- “`*`” ใช้เปรียบเทียบไฟล์ที่มีตัวอักษรกี่ตัวก็ได้ในชื่อไฟล์ เมื่อใช้ “*” อันเดียวก็จะแทนไฟล์ทุกไฟล์ ส่วน “*.*” ก็จะแทนไฟล์ทุกไฟล์ที่มี “.” อยู่ในชื่อไฟล์ด้วย
- ตัวอักษรที่อยู่ภายในวงเล็บเหลี่ยม (“`[`“ และ “`]`”) ก็ใช้เปรียบเทียบแบบกับชื่อไฟล์ที่มีตัวอักษรตัวใดตัวหนึ่งในชื่อไฟล์ ณ ตำแหน่งนั้น
- คำที่คั่นด้วยจุลภาคที่อยู่ภายในวงเล็บหยัก (“`{`“ และ “`}`”) ก็จะกระจายออกเท่ากับจำนวนที่อยู่ในวงเล็บหยัก

เช่น

1. `???` แทนทุกไฟล์ที่ประกอบด้วย 3 ตัวอักษร
2. `?ell?` แทนไฟล์ใด ๆ ที่มี 5 ตัวอักษร และมี `'ell'` อยู่ตรงกลาง

3. `he*` แทนไฟล์ใด ๆ ที่เริ่มต้นด้วย `'he'`

4. `[m-z]*[a-l]` แทนไฟล์ใด ๆ ที่เริ่มต้นด้วยตัวอักษร m ถึง z และจบด้วย a ถึง l

5. `{/usr,}{/bin,/lib}/file` กระจายเป็น `/usr/bin/file /usr/lib/file` และ `/lib/file`

สังเกตว่า shell จะทำการกระจายและเปรียบเทียบแบบในชื่อไฟล์ซึ่งเป็นอาร์กิวเมนต์ของคำสั่งก่อนที่จะรันคำสั่ง

ตัวอย่างการใช้ wildcard ในคำสั่งต่าง ๆ

```
$ cp *.txt chapt1 - copies all files with .txt suffix to
                    directory chapt
$ mv *.txt chapt1 - moves all files with .txt suffix to
                    directory chapt1
$ rm chap?.txt    - deletes all files with chap as the first
                    four characters of their name and with .txt
                    as the last four characters of their  name
$ rm -i *         - deletes all files in current directory but
                    asks first for each file
```

2.7 สิทธิการใช้ไฟล์และไดเรกทอรี

Permission	File	Directory
read	ผู้ใช้สามารถอ่านไฟล์ได้	ผู้ใช้สามารถดูรายชื่อไฟล์ในไดเรกทอรี
write	ผู้ใช้สามารถแก้ไขข้อมูลในไฟล์	ผู้ใช้สามารถสร้างไฟล์ใหม่และลบไฟล์ต่างๆในไดเรกทอรีได้
execute	ผู้ใช้สามารถใช้ชื่อไฟล์นั้นเป็นคำสั่งยูนิกซ์	ผู้ใช้สามารถเปลี่ยนเข้าไปในไดเรกทอรี แต่ไม่สามารถดูรายชื่อไฟล์นอกจากจะมีสิทธิ read ผู้ใช้สามารถอ่านไฟล์ได้ถ้ามีสิทธิ read ในไฟล์

ไฟล์ในระบบปฏิบัติการยูนิกซ์นั้นมีสิทธิอยู่ 3 อย่าง คือ read, write, execute และแบ่งประเภทของผู้ใช้ออกเป็น 3 กลุ่มคือ user/owner (u), group (g), others (o) การกำหนดสิทธิให้กับไฟล์และไดเรกทอรีจะมีความหมายแตกต่างกันดังตารางข้างบนนี้

ตัวอย่าง จากคำสั่ง ls -l

```
-rw-------  2 smith  staff 3287 Apr  8 12:10 file1
```

ผู้ใช้มีสิทธิ read และ write permission ส่วน Group และ others ไม่มี permissions

```
-rw-r--r--  2 smith  staff 13297 Apr  8 12:11 file2
```

ผู้ใช้มีสิทธิ read และ write permission ส่วน Group และ others สามารถอ่านได้อย่างเดียว

```
-rwxr-xr-x  2 smith  staff 4133 Apr  8 12:10 myprog
```

ผู้ใช้มีสิทธิ read, write และ execute permission ส่วน Group และ others มีสิทธิ read และ execute file.

```
drwxr-x---  2 smith  staff 1024 Jun 17 10:00 SCCS
```

อันนี้เป็นไดเรกทอรี ผู้ใช้มีสิทธิ read, write และ execute permission ส่วน Group มีสิทธิ read และ execute permission ในไดเรกทอรี ผู้ใช้อื่นๆนอกจากนั้นไม่มีสิทธิ

- **chmod** (change [file or directory] mode)

```
$ chmod options files
```

ผู้ที่เป็น owner หรือ superuser (root) สามารถเปลี่ยนสิทธิการใช้ไฟล์และไดเรกทอรี โดยใช้คำสั่ง chmod ซึ่งรับค่าตัวเลือกใน 2 รูปแบบ อันแรกกำหนดสิทธิด้วยเลขฐานแปด 3 ตัวเรียงกัน (เลขฐานแปดมีตัวเลข 0 ถึง 7) เลขแต่ละตัวแทนสิทธิการใช้สำหรับ user, group และ others ตามลำดับ ข้างล่างนี้แสดงความสัมพันธ์ของเลขฐานแปดกับสิทธิการใช้

```
---       0
--x       1
-w-       2
-wx       3
r--       4
r-x       5
rw-       6
rwx       7
```

ตัวอย่างเช่น คำสั่ง

```
$ chmod 600 private.txt
```

กำหนดสิทธิการใช้ไฟล์ `private.txt` เป็น `rw-------` (หมายถึงมีเพียงเจ้าของที่สามารถอ่านและเขียนไฟล์)

ตัวอย่างเพิ่มเติม

	user	group	others
chmod 640 file1	rw-	r--	---
chmod 754 file1	rwx	r-x	r--
chmod 664 file1	rw-	rw-	r--

เราสามารถใช้สัญลักษณ์ในการกำหนดสิทธิ ใช้สัญลักษณ์ `u` (user), `g` (group), `o` (other), `a` (all), `r` (read), `w` (write), `x` (execute), `+` (เพิ่มสิทธิ), `-` (ยกเลิกสิทธิ) และ `=` (เครื่องหมายกำหนดสิทธิ) ตัวอย่างเช่น

```
$ chmod ug=rw,o-rw,a-x *.txt
```

กำหนดสิทธิในทุกไฟล์ที่ลงท้าย `.txt` เป็น `rw-rw----` (นั่นคือ เจ้าของและผู้ใช้ในกลุ่มเดียวกันสามารถอ่านและเขียนไฟล์ ในขณะที่ผู้ใช้ทั่วไปไม่มีสิทธิใด ๆ ในไฟล์นี้เลย)

ตัวอย่างเพิ่มเติม

`chmod a+r sample.f`

เพิ่ม read permission สำหรับผู้ใช้ทุกคนในไฟล์ sample.f

`chmod o-r sample.f`

เอา read permission ออกจาก others ในไฟล์ sample.f

`chmod og+rx prog*`

เพิ่มสิทธิ read และ execute permissions สำหรับ group และ others ในไฟล์ทุกไฟล์ที่มีชื่อ "prog" นำหน้า

`chmod +w *`

เพิ่มสิทธิ write permission สำหรับผู้ใช้ ในไฟล์ทุกไฟล์ในไดเรกทอรีทำงานปัจจุบัน

`chmod` ยังมีตัวเลือก -R ให้ใช้เพื่อกำหนดสิทธิให้กับไดเรกทอรีรวมไปถึงทุกไฟล์และไดเรกทอรีข้างใน เช่น

```
$ chmod -R go+r play
```

จะให้สิทธิผู้ใช้ในกลุ่มเดียวกันและผู้ใช้ทั่วไปมีสิทธิ read ในไดเรกทอรี `play` และทุกไฟล์และไดเรกทอรีข้างใน `play`

- **`chgrp`** (change group)

```
$ chgrp group files
```

ใช้เพื่อเปลี่ยนความเป็นเจ้าของไฟล์ คำสั่งนี้ก็มีตัวเลือก -R ให้ใช้เช่นกัน คำสั่งนี้จะใช้โดยผู้ดูแลระบบบ่อยกว่าผู้ใช้

2.8 Default File Permissions

คำสั่ง umask ใช้เพื่อกำหนดค่า default file permissions โดยทั่วไป คำสั่ง umask นี้จะถูกนำไปใส่ไว้ในไฟล์เริ่มต้นการใช้งาน .profile, .cshrc หรือ .login เมื่อผู้ใช้งาน login เข้าใช้งานค่า umask ก็จะถูกกำหนดไว้เลย

คำสั่ง umask จะรับค่าการกำหนดเป็นเลขฐานแปดเท่านั้น ไม่เหมือนกับ chmod และสิ่งที่คำสั่ง umask ทำก็คือการเอาออก (mask out)

Octal number	Access	permissions given
0	rwx	read, write and execute
1	rw-	read and write
2	r-x	read and execute
3	r--	read only
4	-wx	write and execute
5	-w-	write only
6	--x	execute only
7	---	no permissions

ตัวอย่าง

```
$ umask 077
```

ลบ 077 ออกจากค่า default permission สำหรับไฟล์ (666) และไดเรกทอรี (777)

ได้ผลลัพธ์สำหรับไฟล์เป็น 600 `(rw-------)` และสำหรับไดเรกทอรีเป็น 700 `(rwx------)`

```
$ umask 002
```

ลบ 002 ออกจากค่า default permission สำหรับไฟล์ทำให้เป็น 664 `(rw-rw-r--)`

และสำหรับไดเรกทอรีทำให้เป็น 775 `(rwxrwxr-x)`

```
$ umask 022
```

ลบ 022 ออกจากค่า default permission สำหรับไฟล์ทำให้เป็น 644 `(rw-r--r--)`

และสำหรับไดเรกทอรีทำให้เป็น 755 `(rwxr-xr-x)`

2.9 การใช้งานแผ่นฟลอปปี้ดิสก์และซีดีรอม

- **mount, umount**

คำสั่ง mount จะเชื่อม filesystem ที่พบใน device ไปยัง filesystem tree ในทางกลับกันคำสั่ง umount ใช้ยกเลิกการเชื่อม (ต้องจำไว้เสมอว่าเมื่อจะเอาแผ่นฟลอปปี้หรือซีดีรอมออกได้ก็ต่อเมื่อทำคำสั่งนี้แล้ว)

เพื่อที่จะใช้แผ่นฟลอปปี้

```
$ mount /mnt/floppy
```

```
$ cd /mnt/floppy
```

```
$ ls
```

เพื่อสั่งให้ระบบปฏิบัติเก็บสิ่งที่มีการเปลี่ยนแปลงในฟลอปปี้เขียนกลับลงไปให้เรียบร้อย แล้วจึงยกเลิกการเชื่อม filesystem ในแผ่นฟลอปปี้ดิสก์จาก filesystem tree เราทำดังนี้

```
$ umount /mnt/floppy
```

ส่วนการใช้ซีดีรอม ก็ทำเช่นเดียวกับแผ่นฟลอปปี้ โดยเปลี่ยนจากคำว่า floppy เป็น cdrom ดังนี้

```
$ mount /mnt/cdrom
```

```
$ umount /mnt/cdrom
```

แบบฝึกหัดท้ายบท

เรื่อง ระบบไฟล์

1. ลองทำคำสั่งต่อไปนี้ตามลำดับ

prompt> **cd** ↵

prompt> **pwd** ↵ <-- ตอนนี้อยู่ที่ไหน

prompt> **ls -al** ↵

prompt> **cd .** ↵

prompt> **pwd** ↵ <-- ตอนนี้อยู่ที่ไหน

prompt> **cd ..** ↵

prompt> **pwd** ↵ <-- ตอนนี้อยู่ที่ไหน

prompt> **ls -al** ↵

prompt> **cd ..** ↵

prompt> **pwd** ↵ <-- ตอนนี้อยู่ที่ไหน

prompt> **ls -al** ↵

prompt> **cd ..** ↵

prompt> **pwd** ↵ <-- ตอนนี้อยู่ที่ไหน

prompt> **cd /etc** ↵

prompt> **ls -al | more** ↵

prompt> **cat passwd** ↵ ลองเปลี่ยนจากคำสั่ง **cat** เป็น **more**

prompt> **cd -** ↵

prompt> **pwd** ↵

2. สำรวจระบบไฟล์ใน `/bin, /sbin, /tmp` ด้วยคำสั่ง `cd` และ `ls -F` คุณเห็นอะไรบ้าง

prompt> **cd /bin** ↵

prompt> **ls -F** ↵

prompt> **cd /sbin** ↵

prompt> **ls -F** ↵

prompt> **cd /tmp** ↵

prompt> **ls -F** ↵

3. ทำขั้นตอนต่อไปนี้

3.1 เปลี่ยนไปยัง home directory ของผู้ใช้อื่น โดยใช้คำสั่ง

prompt> **cd ~username** ↵

3.2 เปลี่ยนกลับไปยัง home directory ของคุณ

prompt> **cd** ↵

3.3 สร้างไดเรกทอรีย่อยชื่อ `work` และ `play`

prompt> **mkdir work** ↵

prompt> **mkdir play** ↵

3.4 ลบไดเรกทอรีย่อยชื่อ `work`

prompt> **rmdir work** ↵

3.5 คัดลอกไฟล์ /etc/hosts ไปยัง home directory ของคุณ

prompt> **cp /etc/hosts ~** ↵

3.6 ย้ายไฟล์ hosts นั้นไปยังไดเรกทอรีย่อย play

prompt> **mv hosts play** ↵

4. สำรวจใน /dev คุณสามารถบอกได้ไหมว่ามีดีไวซ์อะไรบ้างที่ใช้ได้

prompt> **ls -lLR /dev** ↵

prompt> **ls -lL /dev/pts** ↵

prompt> **ls -lL /dev/null** ↵

5. ทำขั้นตอนต่อไปนี้

5.1 ต้องการรู้ชื่อดีไวซ์ tty (จอเทอร์มินัล) ที่คุณกำลังใช้งาน

prompt> **who am i** ↵

```
sa33       pts/0        Jun 20 16:50    (wiboon.cc.psu.ac.th)
```

5.2 ใช้คำสั่ง ls -lL ดูว่ามีชื่อดีไวซ์ pts/0 ดังกล่าวหรือไม่

prompt> **ls -lL /dev/pts/0** ↵

```
crw--w----   1 sa33     tty     24,   0 Jun 22 13:44 /dev/pts/0
```

5.3 สร้างไฟล์ชื่อ hello.txt โดยใช้วิธีคัดลอกตัวอักษรที่เกิดบนจอภาพลงไฟล์ ดังนี้

prompt> **cp /dev/pts/0 hello.txt** ↵

พิมพ์ข้อความ hello world กดปุ่ม Ctrl + d

prompt> **cat hello.txt** ↵

5.4 สร้างไฟล์ hello.txt ให้มีขนาดเป็น 0 byte

prompt> **cp /dev/null hello.txt** ↵

6. ทำขั้นตอนต่อไปนี้

6.1 เปลี่ยนเข้าไปที่ไดเรกทอรีย่อย play

prompt> **cd play** ↵

6.2 สร้างไฟล์ที่มีชื่อยาว ๆ ดังนี้

prompt> **cat > longfilename1** ↵

พิมพ์ข้อความ abcdef ↵

กดปุ่ม Ctrl + d

6.3 สร้าง symbolic link ชื่อ m1 ชี้ไปยังไฟล์ longfilename1

prompt> **ln -s longfilename1 m1 ; cat m1** ↵

6.4 สร้าง symbolic link ชื่อ terminal ที่จะชี้ไปยังดีไวซ์ tty ของคุณ

prompt> **ln -s /dev/pts/0 terminal** ↵

6.5 สร้างไฟล์ชื่อ world.txt ที่มีคำ “hello world” อยู่ภายใน ดังนี้

prompt> **cp terminal world.txt** ↵

พิมพ์ข้อความ hello world ↵

กดปุ่ม Ctrl + d

6.6 ใช้คำสั่ง cp คัดลอกไฟล์ world.txt ไปยังไฟล์ terminal มีอะไรเกิดขึ้น ดังนี้

prompt> **cp world.txt terminal** ↵

7. ทำขั้นตอนต่อไปนี้

7.1 กลับไปที่ home directory

prompt> **cd** ↵

7.2 คัดลอกไดเรกทอรี play เป็นไดเรกทอรีชื่อ work อย่าให้ symbolic link ที่สร้างไว้หายไป

prompt> **cp -Rd play work** ↵

8. ลบไดเรกทอรีชื่อ play และสิ่งที่อยู่ข้างในด้วยคำสั่งเดียว ไม่ต้องให้มีคำถามยืนยันการลบใดๆ

prompt> **rm -rf play** ↵

9. เปลี่ยนไปยังไดเรกทอรีที่คุณไม่ได้เป็นเจ้าของ และลองลบไฟล์

prompt> `cd ~username`

prompt> `rm filename`

เรื่อง สิทธิในไฟล์และไดเรกทอรี และคำสั่งในการเปลี่ยนสิทธิ

1. เขียนคำสั่งเพื่อกำหนดสิทธิให้ไฟล์เป็น `r--r--r--` ทั้ง 2 แบบ แล้วลองทดสอบกับไฟล์ที่อยู่ใน home directory ของคุณ

prompt> **chmod 444 hello.txt** ↵

prompt> **chmod ugo=r-- hello.txt** ↵

2. ทำขั้นตอนต่อไปนี้

2.1 เปลี่ยนสิทธิของ home directory ให้เป็นส่วนตัว

prompt> **chmod 700 ~** ↵

2.2 แล้วให้เพื่อนลองเข้ามายังไดเรกทอรีของคุณ

2.3 เปลี่ยนสิทธิกลับให้เหมือนเดิม

prompt> **chmod 755 ~** ↵

3. ทำขั้นตอนต่อไปนี้ เพื่อจัดทำโฮมเพจส่วนตัว

3.1 สร้างไดเรกทอรีชื่อ public_html

prompt> **cd** ↵

prompt> **mkdir public_html** ↵

3.2 กำหนดสิทธิในไฟล์และไดเรกทอรี

prompt> **chmod o+x ~** ↵

prompt> **chmod -R o+rx public_html** ↵

4. ทำขั้นตอนต่อไปนี้

4.1 กำหนดค่าสิทธิสำหรับการสร้างไฟล์เป็น 000

prompt> **umask 000** ↵

4.2 สร้างไฟล์ชื่อ world1.txt

prompt> **cat > world1.txt** ↵

พิมพ์ข้อความ hello world ↵

กดปุ่ม Ctrl+d เพื่อจบ

4.3 ตรวจสอบสิทธิของไฟล์ว่าเป็นอย่างไร

prompt> **ls -l** ↵

4.4 เปลี่ยนเป็น umask 022 แล้วสร้างไฟล์ชื่อ world2.txt

prompt> **umask 022** ↵

prompt> **cat > world2.txt** ↵

พิมพ์ข้อความ hi world ↵

กดปุ่ม Ctrl+d เพื่อจบ

4.5 เปลี่ยนสิทธิกลับเป็น 755 ทั้ง world1.txt และ world2.txt

prompt> **chmod 755 world?.txt** ↵

5. ทำขั้นตอนต่อไปนี้

5.1 คัดลอกไฟล์ /bin/sh มาไว้ใน home directory

prompt> **cp /bin/sh ~** ↵

5.2 ใช้คำสั่งต่อไปนี้

prompt> **chmod +s sh** ↵

5.3 ตรวจสอบสิทธิของไฟล์ sh ด้วยคำสั่ง ls

prompt> **ls -l** ↵

5.4 จากนั้นเปลี่ยนเข้าไปยังไดเรกทอรีของเพื่อนข้างเคียง (ผลัดกัน)

prompt> **cd ~sa33** ↵

5.5 รัน shell sh นั้นดังนี้

prompt> **./sh** ↵

5.6 รันคำสั่ง id

$ **id** ↵

5.7 เลิกออกจาก shell sh ที่รันอยู่นั้น

$ **exit** ↵

prompt>

5.8 ลบไฟล์ sh นั้นทิ้ง

prompt> **rm sh** ↵

5.9 คุณทราบไหมว่าผู้ดูแลระบบใช้ `chmod +s` เพื่อทำอะไร

บทที่ 3 เชลล์

วัตถุประสงค์

เนื้อหาในบทนี้ (เวลาโดยประมาณ 3 ชั่วโมง)

- เชลล์คืออะไร
- ความสามารถของเชลล์ทั่วไป
- วิธีเปลี่ยนไปใช้เชลล์ชนิดอื่นๆ
- ตัวแปรเชลล์คืออะไร มีวิธีกำหนดอย่างไร
- ไฟล์เริ่มต้นใช้งาน
- การเปลี่ยนทิศทาง input และ output
- การส่งผ่านข้อมูลด้วยไปป์ (pipe)
- การใช้ quotes ในบรรทัดคำสั่ง

3.1 เชลล์คืออะไร

เมื่อไรก็ตามที่คุณล็อคอินเข้าไปยังเครื่องยูนิกซ์ ขณะนั้นคุณกำลังอยู่ในเชลล์ การทำงานของคุณก็คือการสั่งให้เชลล์แปลความหมายคำสั่งและสร้างผลลัพธ์ให้ ซึ่งการทำงานทุกอย่างจะอยู่ภายใต้เชลล์

เชลล์คือส่วนที่ผู้ใช้ใช้เพื่อติดต่อกับระบบปฏิบัติการ มันทำหน้าที่เป็นตัวแปลคำสั่ง เพื่อบอกระบบปฏิบัติการว่าผู้ใช้ต้องการทำอะไร และแสดงผลลัพธ์ที่เกิดขึ้นไปบนจอภาพหรือควบคุมให้ผลลัพธ์ถูกเก็บลงไฟล์ก็ได้

เชลล์อะไรที่เราใช้

มีเชลล์ชื่อต่าง ๆ มากมายที่ใช้ในยูนิกซ์ แต่ที่ใช้กันมากคือ

- Bourne shell (sh) บอร์นเชลล์เป็นเชลล์ที่ใช้มาตั้งแต่แรกเริ่มมียูนิกซ์ และยังใช้กันอยู่ เขียนขึ้นโดย S.R. Bourne มีโครงสร้างไวยากรณ์ของคำสั่งคล้าย ๆกับภาษา Algol

- C shell (csh) ซีเชลล์ เป็นเชลล์ที่มีโครงสร้างไวยากรณ์ของคำสั่งคล้ายภาษาซี เขียนขึ้นโดย Bill Joy อยู่ที่ University of California, Berkley เหมาะสำหรับผู้ที่เคยเขียนภาษาซี ซีเชลล์มีเครื่องมือช่วยในการทำงานมากกว่า
บอร์นเชลล์

- Korn shell (ksh) คอร์นเชลล์ เขียนขึ้นโดย David Korn ของบริษัท AT&T ไม่มีในยูนิกซ์ทุกระบบ แต่ติดตั้งเพิ่มได้ คอร์นเชลล์ดึงเอาลักษณะเด่นของทั้งบอร์นเชลล์และซีเชลล์เข้ามาไว้ทำให้เพิ่มประสิทธิภาพการใช้งาน

- TC Shell (tcsh) ทีซีเชลล์ เป็น Public domain shell มีความสามารถเช่นเดียวกับซีเชลล์ ใช้สไตล์แบบ EMACS ในการแก้ไขคำสั่งภายในบรรทัด

- Bourne Again Shell (bash) บอร์นอะเกนเชลล์ เป็น Public domain shell เขียนขึ้นโดย Free Software Foundation เป็นการรวมซีเชลล์และคอร์นเชลล์เข้ามาไว้ด้วย ภาษาที่ใช้ในการเขียนโปรแกรมก็ใช้กันได้กับบอร์นเชลล์ จึงเป็นที่นิยมกันมาก โดยเฉพาะ Linux

ผู้ใช้จะได้รับเชลล์ใดในการใช้งานนั้นจะขึ้นอยู่กับผู้ดูแลระบบ แต่เรามีวิธีตรวจสอบว่าใช้เชลล์ใดดังนี้

prompt> **echo $SHELL**

แต่ละเชลล์จะมี prompt ของมันเอง ดังนี้

$ (dollar sign) - sh, ksh, bash

% (percent sign) - csh, tcsh

3.2 ความสามารถของเชลล์ทั่วไป

	Bourne sh	C csh	TC tcsh	Korn ksh	BASH bash
Programming language	Yes	Yes	Yes	Yes	Yes
Shell variables	Yes	Yes	Yes	Yes	Yes
Command alias	No	Yes	Yes	Yes	Yes
Command history	No	Yes	Yes	Yes	Yes
Filename completion	No	Yes*	Yes	Yes*	Yes
Command line editing	No	No	Yes	Yes*	Yes
Job control	No	Yes	Yes	Yes	Yes

* not the default setting for this shell

Command history: มีการเก็บคำสั่งที่เคยใช้ไว้ เพื่อนำกลับมาใช้ได้อีก

Event reexecution: มีวิธีในการสั่งให้คำสั่งที่เคยรันไปแล้ว รันใหม่ซ้ำ

Modifying previous events: ช่วยให้นำคำสั่งเดิมมาแก้ไขเพื่อทำเป็นคำสั่งใหม่

Aliases: มีคำสั่ง alias สำหรับสร้างคำสั่งใหม่ที่สั้นกว่าคำสั่งเดิม

Filename Completion: เชลล์จะเติมชื่อคำสั่งที่เหลือให้ในขณะที่ผู้ใช้พิมพ์ไปได้เพียง 2 – 3 ตัวอักษร

3.3 วิธีเปลี่ยนไปใช้เชลล์ชนิดอื่นๆ

ถ้าต้องการทราบว่ามีเชลล์ใดบ้างติดตั้งให้ใช้งาน ใช้คำสั่ง **`chsh -l`** หรือใช้คำสั่ง `which shell` เพื่อถามว่า shell นั้นเก็บอยู่ที่ไหนก็ได้ เช่น **`which bash`** เป็นต้น

ถ้าต้องการเปลี่ยนไปใช้เชลล์อื่น เช่น bash shell ก็พิมพ์ชื่อเชลล์ลงไป

```
$ bash
bash$
```

ถ้าต้องการเลิกและกลับไปใช้เชลล์เดิม พิมพ์คำสั่ง **`exit`**

3.4 ตัวแปรเชลล์คืออะไร

เชลล์ทุกเชลล์จะมีตัวแปรไว้ให้ใช้งานทั้งที่เป็นตัวแปรที่ระบบเตรียมไว้ให้ (predefined) และผู้ใช้กำหนดตัวแปรขึ้นเอง นิยมใช้ตัวแปรเมื่อเขียน shell scripts ตัวแปรจะมีหลายลักษณะดังนี้

local - ใช้ในเชลล์ที่กำลังทำงานอยู่เท่านั้น

global - ใช้ในเชลล์ที่กำลังทำงาน และรวมถึงโพรเซสลูกที่เกิดขึ้น

string - ใช้เป็นตัวอักษร

numeric - ใช้เป็นตัวเลข

arrays – เก็บค่ามากกว่า 1 ค่า

ตัวอย่าง

predefined variables เช่น SHELL, HOME, PATH, prompt เป็นต้น

user defined เช่น v1, v2, x, y เป็นต้น

3.5 มีวิธีกำหนดตัวแปรอย่างไร

แบบ Global,

The C Shell,

```
setenv NAME value
```

i.e.

% **setenv TERM=vt100**

The Bourne Shell,

```
NAME = value; export NAME
```

i.e.

$ **TERM=vt100; export TERM**

แบบ Local,

The C Shell,

```
set NAME = value
```

i.e.

% **set x = 10**

The Bourne Shell,

```
NAME = value
```

i.e.

$ **x=10**

3.6 ไฟล์เริ่มต้นใช้งาน

ในระบบยูนิกซ์ทั่วไป จะมีไฟล์เริ่มต้นใช้งานเมื่อผู้ใช้เข้าสู่ระบบอยู่ 2 อย่างคือ System-wide shell initialization files และ User customizations files

System-wide shell initialization files จะกำหนดโดยผู้ดูแลระบบ เพื่อใช้เป็นค่าเริ่มต้นให้กับผู้ใช้ทุกคนเหมือนกัน

ตัวอย่าง

```
/etc/environment
/etc/profile
/etc/cshrc
/etc/login
```

User customizations files นั้นจะกำหนดโดยผู้ใช้งานเอง เพื่อใช้ค่าเริ่มต้นตามที่ผู้ใช้กำหนด โดยไม่ใช้ค่าเริ่มต้นจาก System-wide shell initialization files ได้

ไฟล์ที่ใช้ทุกครั้งที่เริ่มต้นเชลล์ใหม่

```
.cshrc          - csh, tcsh
.tcshrc         - tcsh
.kshrc          - ksh
.bashrc         - bash
```

และไฟล์ที่ใช้ในระหว่าง interactive login

```
.login          - csh, tcsh
.profile        - sh, ksh, bash
.bash_profile   - bash (alternative 1)
.bash_login     - bash (alternative 2)
```

ยกตัวอย่างเช่น C shell จะใช้ไฟล์ `.cshrc` และ `.login` ตามลำดับ

แต่อย่างไรก็ตามผู้ดูแลระบบอาจไม่เตรียมไฟล์ User customizations files เหล่านี้ไว้ให้ก็ได้

3.7 การเปลี่ยนทิศทาง input และ output

ผลลัพธ์ที่เกิดขึ้นจากโปรแกรมหรือคำสั่งใด ๆ จะถูกส่งออกไปแสดงที่จอภาพ ส่วนข้อมูลเข้าก็ได้มาจากคีย์บอร์ด

ถ้าพูดในเชิงเทคนิค เรากล่าวได้ว่าโพรเซสจะเขียนผลลัพธ์ไปที่ standard output และรับข้อมูลเข้าไปประมวลผลทาง standard input และยังมีผลลัพธ์อีกอย่างหนึ่งเรียกว่า standard error ที่ซึ่งโพรเซสจะเขียน errror messages ซึ่งก็คือจอภาพ

ตัวอย่างต่อไปนี้ เป็นวิธีการที่ใช้กับ bash shell

เราใช้โอเปอเรเตอร์ **>** เพื่อเปลี่ยนทิศทางผลลัพธ์ standard output ลงไฟล์:

```
bash$ echo hello > file1

bash$ cat file1
hello
```

นอกจากนี้เรายังใช้โอเปอเรเตอร์ **>** เพื่อสร้างไฟล์ด้วยคำสั่ง cat:

```
bash$ cat > file2
hello world
good bye (กดปุ่ม Ctrl+d เพื่อจบ)
```

ถ้าหากว่า file1 มีอยู่แล้วและเราใช้โอเปอเรเตอร์ > ก็จะเป็นการเขียนทับไฟล์ ข้อมูลเดิมที่มีอยู่จะเสียไป ดังนั้นถ้าต้องการเขียนข้อความต่อท้ายไฟล์ให้ใช้โอเปอเรเตอร์ **>>** แทน ดังนี้

```
bash$ echo bye >> file1

bash$ cat file1
hello
bye
```

ในยูนิกซ์ เราใช้เลข 0 แทน standard input เลข 1 แทน standard output และเลข 2 แทน standard errror ดังนั้น ถ้าเราต้องการเก็บ error messages ก็ทำได้โดยใส่เลข 2 นำหน้าโอเปอเรเตอร์ > ดังนี้

```
bash$ cat nonexistent 2>errors

bash$ cat errors
cat: nonexistent: No such file or directory
```

เราสามารถส่ง output และ error ลงไปในไฟล์ได้ 2 วิธีคือ

แยกคนละไฟล์ ดังนี้

```
bash$ cat file1 none 1>files 2>error.log
```

รวมไว้ในไฟล์เดียวกัน ดังนี้

```
bash$ cat file1 none 1>output 2>output
```

ซึ่งเขียนได้อีกแบบ

```
bash$ cat file1 none > output 2>&1
```

เราสามารถเปลี่ยนจากการรับข้อมูลเข้าทางคีย์บอร์ดเป็นรับข้อมูลเข้าจากไฟล์ได้ โดยใช้โอเปอเรเตอร์ < ดังนี้

```
bash$ cat < file1
hello
bye
```

ลองดูอีกตัวอย่าง เป็นการนำข้อความใน file1 ไปเป็นข้อความที่จะพิมพ์ลงในอีเมลล์ ตัวอย่างนี้ส่งไปถึงผู้รับ user@oursite.domain โดยมี subject คือ -s “This is a test”

```
bash$ mail -s "This is a test" user@oursite.domain < file1
```

เราสามารถใช้ทั้งการเปลี่ยนทิศทางข้อมูลเข้าและเปลี่ยนทิศทางผลลัพธ์ภายในบรรทัดคำสั่งเดียวกันได้ แต่ระวังอย่างใช้ชื่อไฟล์เดียวกัน ดังนี้

```
bash$ cat < file1 > file1
```

แบบนี้มันจะทำให้ file1 เสีย เพราะว่าสิ่งแรกที่เชลล์ทำเมื่อมันตรวจพบโอเปอเรเตอร์ > มันจะสร้างไฟล์ว่างเพื่อรอรับผลลัพธ์

3.8 การส่งผ่านข้อมูลด้วยไปป์ (pipe)

โอเปอเรเตอร์ไปป์ | ใช้เพื่อสร้างคำสั่งหลายๆคำสั่งเชื่อมเข้าด้วยกัน โดยที่เป็นการนำผลลัพธ์ของคำสั่งที่อยู่ทางซ้ายของโอเปอเรเตอร์ | ส่งต่อให้กับคำสั่งที่อยู่ทางขวา หากไม่ใช้วิธีนี้ก็ต้องใช้วิธีที่กล่าวไปแล้วคือเก็บผลลัพธ์ลงไฟล์ก่อนแล้วนำไฟล์เป็นข้อมูลเข้าให้กับคำสั่งถัดไป ซึ่งจะใช้ถึง 2 คำสั่ง จะเสียเวลาในการสร้างไฟล์ด้วย แต่การเขียนคำสั่งโดยใช้ไปป์ ใช้บรรทัดเดียว และไม่เสียเวลาสร้างไฟล์

ตัวอย่าง

ต้องการหยุดดูทีละหน้าเมื่อผลลัพธ์ของคำสั่ง ls มีความยาวเกินหน้าจอ

```
$ ls -l | more
```

ต้องการนับจำนวนผู้ใช้งานในขณะนั้น โดยเรารู้ว่าคำสั่ง who จะให้ผลลัพธ์เป็นบรรทัดหลายบรรทัด แต่ละบรรทัดคือชื่อผู้ใช้และเวลาที่เข้าใช้งาน และเรารู้ว่าคำสั่ง wc -l จะใช้นับจำนวนบรรทัดในไฟล์ข้อมูลได้

```
$ who | wc -l
```

มันมีประโยชน์สำหรับการนำคำสั่งหลายๆ คำสั่งมาเชื่อมกันเพื่อทำงานที่ซับซ้อน เช่น

```
$ who | cut -f1 | sort | uniq
```

บรรทัดนี้จะสร้าง 4 โพรเซส (คือ who, cut, sort และ uniq) ซึ่งทำงานต่อเนื่องกันไป เมื่อคำสั่งเริ่มทำงาน ผลลัพธ์ของโพรเซส who จะถูกส่งผ่านไปให้โพรเซส cut ซึ่งจะส่งต่อไปให้โพรเซส sort และสุดท้ายส่งผ่านไปให้โพรเซส uniq คำสั่ง uniq แสดงผลลัพธ์บนจอภาพ

3.9 การใช้ quotes ในบรรทัดคำสั่ง

ตอนนี้เรารู้แล้วว่าตัวอักษรพิเศษบางตัวถูกแปลความเป็นอย่างอื่นโดย shell ในการผ่านค่าอาร์กิวเมนต์ที่มีตัวอักษรพิเศษเหล่านี้ให้กับคำสั่งโดยไม่ต้องการให้ shell แปลความหมายเลย เราจำเป็นต้องใช้เครื่องหมาย quote มาช่วย ซึ่งมีอยู่ 3 ระดับ

1. แทรกเครื่องหมาย “\” ไว้หน้าตัวอักษรพิเศษนั้น

เช่น

```
$ echo Can you use command echo \$SHELL \?
Can you use command echo $SHELL ?
```

2. ใช้เครื่องหมาย double quotes (") ล้อมอาร์กิวเมนต์เพื่อป้องกันตัวอักษรเกือบทั้งหมด ไม่ให้มีการขยายความ

เช่น

```
$ echo "Can you use command echo $SHELL ?"
Can you use command echo /bin/csh ?
```

3. ใช้เครื่องหมาย single quotes (') ล้อมอาร์กิวเมนต์เพื่อป้องกันตัวอักษรทั้งหมด ไม่ให้มีการขยายความ

เช่น

```
$ echo 'Can you use command echo $SHELL ?'
Can you use command echo $SHELL ?
```

นอกจากนี้ ยังมี quote ชนิดที่สี่ คือเครื่องหมาย single backward quotes (`) ใช้สำหรับส่งผ่านผลลัพธ์ของคำสั่งหนึ่งให้เป็นข้อมูลเข้าของอีกคำสั่ง ตัวอย่างเช่น

```
$ hostname
rose
$ echo this machine is called `hostname`
this machine is called rose
```

คุณสามารถกดปุ่ม Alt + NUM 96 เพื่อให้ได้เครื่องหมาย backquote (`)

แบบฝึกหัดท้ายบท

เรื่อง ตัวแปรเชลล์

1. เมื่อ login เข้าใช้งานได้แล้ว เราจะทราบชื่อ Shell ที่เราได้รับจากผู้ดูแลระบบได้

prompt> **echo $SHELL** ↵

2. เปลี่ยน shell จากปัจจุบันซึ่งเป็น csh shell ไปเป็น bash shell แล้วลองใช้ปุ่มลูกศรขึ้น ลง เพื่อเลือกคำสั่งที่ใช้แล้ว มาแก้ไขเพื่อสั่งใหม่ การแก้ไขให้กดปุ่มลูกศรซ้าย ขวา

prompt> **bash** ↵

3. กำหนดค่าให้ตัวแปร x เป็น 10 และ y เป็น 20

prompt> **x=10 ; y=20** ↵

4. ใช้คำสั่ง echo แสดงค่าของ x และ y

prompt> **echo x = $x , y = $y** ↵

เรื่อง การเปลี่ยนทิศทาง *input* และ *output*

1. เขียนคำสั่งอย่างไรเพื่อเก็บคำแนะนำการใช้คำสั่ง cp ลงในไฟล์ชื่อ cp.manpages

prompt> **man cp > cp.manpages** ↵

2. สร้างไฟล์ชื่อ exercise.1 เพื่อเก็บข้อความ “This is an exercise.”

prompt> **cat > exercise.1** ↵

```
This is an exercise. ↵
```

กดปุ่ม Ctrl+d เพื่อจบ

3. ต้องการเพิ่มข้อความ “Good luck” ต่อท้ายไฟล์ exercise.1

prompt> **cat >> exercise.1** ↵

```
Good luck ↵
```

กดปุ่ม Ctrl+d เพื่อจบ

4. ใช้คำสั่ง echo สร้างไฟล์ชื่อ unsorted มีจำนวน 4 บรรทัด ดังนี้

prompt> **echo cherry > unsorted** ↵

prompt> **echo apple >> unsorted** ↵

prompt> **echo lemon >> unsorted** ↵

prompt> **echo banana >> unsorted** ↵

5. เรียงลำดับข้อมูลในไฟล์ unsorted ด้วยคำสั่ง sort แล้วเก็บไว้ในไฟล์ชื่อ sorted

prompt> **sort < unsorted > sorted** ↵

6. ใช้คำสั่ง ls -l เพื่อเก็บรายชื่อไฟล์ที่อยู่ใน home directory ของคุณไว้ในไฟล์ชื่อ homedir.files

prompt> **ls -l > homedir.files** ↵

7. แล้วใช้คำสั่งต่อไปนี้เพื่อส่งอีเมล์ถึงผู้สอน

prompt> **mail -s "$USER dir" sa33@maliwan.psu.ac.th <** ↵

homedir.files ↵

เรื่อง การใช้ *quotes* ในบรรทัดคำสั่ง

1. จะสร้างไฟล์ชื่อ $SHELL ได้อย่างไร และจะลบไฟล์นั้นได้อย่างไร

prompt> **cat > \$SHELL** ↵

prompt> **rm \$SHELL** ↵

2. จะสร้างไฟล์ชื่อ This is a long file name ได้อย่างไร และจะลบไฟล์นั้นได้อย่างไร

prompt> **cat > "my file"** ↵

prompt> **rm "my file"** ↵

3. คุณคิดว่า 2 คำสั่งนี้ ให้ผลลัพธ์เหมือนกันหรือไม่

prompt> **echo "Command 'pwd' will print current directory like `pwd`"** ↵

และ

prompt> **echo 'Command "pwd" will print current directory like `pwd`'** ↵

บทที่ 4 คำสั่งดำเนินการไฟล์

วัตถุประสงค์

เนื้อหาในบทนี้ (เวลาโดยประมาณ 2 ชั่วโมง)

- วิธีค้นหาข้อมูลที่อยู่ในไฟล์
- คำสั่งค้นหาไฟล์
- วิธีค้นหาข้อมูลที่ต้องการในไฟล์ต่างๆ
- วิธีเรียงลำดับข้อมูลในไฟล์
- คำสั่งสำหรับการบีบอัดไฟล์และการแบ็คอัพข้อมูล

4.1 วิธีค้นหาข้อมูลในไฟล์

นอกจากคำสั่ง cat ที่เรารู้จักแล้ว ยังมีอีกหลายคำสั่งที่ใช้ในการจัดการข้อมูลหรือเนื้อหาที่อยู่ในไฟล์

- **file** *filename(s)*

  `file` วิเคราะห์สิ่งที่อยู่ในไฟล์แล้วแจ้งเป็นข้อความบอกชนิดของไฟล์:

  ตัวอย่าง

```
$ file myprog.c letter.txt webpage.html
myprog.c: C program text
letter.txt: English text
webpage.html: HTML document text
```

- **head**, **tail** *filename*

  `head` และ `tail` แสดงจำนวนบรรทัดตอนบนสุดและตอนล่างสุดของไฟล์ตามลำดับ ค่า default คือ 10 บรรทัด เราสามารถกำหนดจำนวนบรรทัดที่ต้องการได้

  ตัวอย่าง

  ต้องการแสดงข้อมูล 5 บรรทัดแรก

```
$ head -5 /etc/passwd
```

ต้องการแสดงข้อมูล 20 บรรทัดสุดท้าย

```
$ tail -20 /etc/passwd
```

ตัวอย่างเพิ่มเติม

```
head sample.f        - display first 10 lines (default)
head -5 sample.f     - display first 5 lines
head -25 sample.f    - display first 25 lines
tail sample.f        - display last 10 lines (default)
tail -5 sample.f     - display last 5 lines
tail -5c sample.f    - display last 5 characters
tail -25 sample.f    - display last 25 lines
```

tail มี -f option เพื่อใช้ในการดูข้อมูลจำนวนบรรทัดท้ายสุดอย่างต่อเนื่อง อันนี้ใช้ในการเฝ้าดู log files

```
$ tail -f /var/log/messages
```

```
$ tail -f /var/log/syslog
```

4.2 คำสั่งค้นหาไฟล์

- **find**

  ใช้ค้นหาตำแหน่งที่ไฟล์เก็บอยู่

  ```
  find directory -name targetfile -print
  ```

  find จะค้นหาชื่อไฟล์ตามที่ระบุใน targetfile เริ่มต้นค้นหาที่ directory และพิมพ์ออกจอภาพ

  ตัวอย่าง

  ```
  $ find /home -name "*.txt" -print
  ```

  จะค้นหาไฟล์ทุกไฟล์ที่มีชื่อจบด้วย ".txt" ทุกไดเรกทอรีที่อยู่ใน /home และส่งผลลัพธ์ที่

ค้นหาได้ไปที่จอภาพ

find สามารถค้นหาได้หลายวิธี ไม่เพียงแค่การค้นหาโดยชื่อ มันยังใช้ค้นหาชนิดของไฟล์ก็ได้ (เช่นชนิดไฟล์ -type f ชนิดไดเรกทอรี -type d) หรือค้นหาโดย permissions (เช่น -perm o=r สำหรับทุกไฟล์และไดเรกทอรีที่เปิดสิทธิอ่านให้ทุกคน), หรือค้นหาโดย size (-size) เป็นต้น

คุณยังสามารถรันคำสั่งได้ทันทีที่ค้นหาได้ชื่อไฟล์ เช่น

```
$ find . -name "*.txt" -exec wc -l '{}' ';'
```

เป็นการนับจำนวนบรรทัดในทุกๆไฟล์ที่อยู่ใน current directory เครื่องหมาย '{}' จะถูกใช้เพื่อแทนชื่อไฟล์ที่ค้นหาพบ และเครื่องหมาย ';' ใช้เพื่อจบ -exec

- **which** *command*

ถ้าคุณสามารถรันโปรแกรมหรือยูทิลิตี้ระบบโดยพิมพ์ชื่อของมันที่ shell prompt แล้วละก็ คุณสามารถหาว่ามันเก็บอยู่ที่ไหนบนดิสก์

ตัวอย่าง

```
$ which ls
/bin/ls
```

4.3 วิธีค้นหาข้อมูลที่ต้องการในไฟล์ต่างๆ

- **grep** (General Regular Expression Print)

```
grep options pattern files
```

grep ค้นหาในไฟล์ที่ระบุ (ถ้าไม่ระบุก็จะค้นหาจาก standard input) แสดงจำนวนบรรทัดที่ตรงกับรูปแบบที่จอภาพ

ตัวอย่าง

```
$ grep hello *.txt
```

ค้นหาคำว่า hello ในไฟล์ที่มีชื่อลงท้าย .txt

grep มี options ที่น่าใช้ดังนี้

-c พิมพ์จำนวนบรรทัดที่นับได้จากการค้นหา

-i ไม่สนใจตัวอักษรว่าจะเป็นพิมพ์ใหญ่หรือพิมพ์เล็ก

-v พิมพ์บรรทัดที่ไม่ตรงกับรูปแบบที่ระบุ

-n พิมพ์หมายเลขบรรทัดหน้าบรรทัดที่ค้นหาได้

ดังนั้น

```
$ grep -vi hello *.txt
```

ค้นหาในทุกไฟล์ *.txt แสดงบรรทัดที่ไม่มีคำว่า hello ทุกแบบ (Hello, HELLO, or hELlO)

ตัวอย่างที่ซับซ้อน

```
$ cat hello.txt | grep "dog" | grep -v "cat"
```

คำสั่งนี้จะค้นหาทุกบรรทัดที่มีคำว่า "dog" แต่ไม่มีคำว่า "cat".

คุณสามารถใช้ grep ร่วมกับ back quotes ดังนี้

```
$ grep hello `find . -name "*.txt" -print`
```

จะเป็นการค้นหาทุกไฟล์ที่อยู่ภายในไดเรกทอรีทรีตั้งแต่ current directory ที่มีคำว่า "hello" อยู่ในไฟล์

regular expression

- ตัวอักษรและตัวเลขจะเรียกว่า regular expressions
- ตัวอักษรที่มีความหมายพิเศษในเชลล์ให้ใส่ quoted ด้วยเครื่องหมาย backslash (\)
- ตัวอักษรชุดหนึ่งที่อยู่ภายใน '[' และ ']' จะใช้จับคู่กับตัวอักษรตัวใดก็ได้ที่อยู่ในชุด ถ้าตัวอักษรตัวแรกมีเครื่องหมาย caret ^ นำหน้า มันจะใช้เพื่อจับคู่กับตัวอักษรอื่น ๆ ที่ไม่อยู่ในชุด เราสามารถใช้เครื่องหมาย dash (-) ระหว่างตัวอักษรตัวแรกและตัวสุดท้าย ดังนั้น [0-9] จับคู่กับตัวเลขใดๆและ [^0-9] จับคู่กับตัวอักษรอื่นๆที่ไม่ใช่ตัวเลข
- เครื่องหมาย caret ^ และเครื่องหมาย dollar sign $ เป็นตัวอักษรพิเศษจับคู่กับตัวอักษรตัวแรกสุดหน้าบรรทัด และตัวสุดท้ายในบรรทัดตามลำดับ เครื่องหมาย dot . จับคู่กับตัวอักษรใด ๆณ ตำแหน่งนั้น ดังนั้น

```
$ grep ^..[l-z]$ hello.txt
```

จับคู่กับบรรทัดใด ๆ ในไฟล์ hello.txt ที่ขึ้นต้นด้วยลำดับตัวอักษร 3 ตัวและจบด้วยตัวอักษรตั้งแต่ l ถึง z

4.4 วิธีเรียงลำดับข้อมูลในไฟล์

มีคำสั่งอยู่ 2 คำสั่งที่ใช้ในการเรียงลำดับข้อมูลในไฟล์:

- **sort** *filenames*

  sort เรียงลำดับข้อมูลในทุกบรรทัดเรียงแบบ ตัวอักษร (alphabetically) หรือตัวเลข (numerically) ถ้าระบุ -n option ผลลัพธ์จะถูกแสดงบนจอภาพ และอาจจะถูกเก็บไว้ในไฟล์ก็ได้หากใช้วิธีเปลี่ยนทิศทาง output

ดังนั้น

```
$ sort input1.txt input2.txt > output.txt
```

ให้ผลลัพธ์เป็นบรรทัดที่เรียงลำดับแล้วจากไฟล์ input1.txt และ input2.txt เก็บไว้ในไฟล์ output.txt

- **uniq** *filename*

  uniq ลบบรรทัดที่อยู่ติดกันที่เหมือนกันให้เหลือเพียง 1 บรรทัด คำสั่งนี้มีประโยชน์เมื่อใช้ร่วมกับคำสั่ง sort

  ```
  $ sort input.txt | uniq > output.txt
  ```

4.5 คำสั่งแบ็คอัพข้อมูล

- **tar** (tape archiver)

  tar แบ็คอัพทั้งไดเรกทอรีและไฟล์ทั้งหมดลงไปบน tape device หรือลงไปในไฟล์ 1 ไฟล์ เรียกว่า archive

  ไฟล์ archive คือไฟล์ที่เก็บไฟล์อื่น ๆ พร้อมทั้งข้อมูลเช่นชื่อไฟล์ ชื่อเจ้าของไฟล์ เวลาของไฟล์ และสิทธิการใช้งานไฟล์

  tar ไม่ได้ทำการบีบอัดไฟล์ใด ๆ

  เพื่อที่จะสร้างไฟล์ archive

  ```
  tar -cvf archivename filenames
  ```

  เมื่อ archivename จะใช้ส่วนขยายไฟล์เป็น .tar (แต่ไม่จำเป็นต้องใส่ก็ได้)

  option `c` หมายถึง create

  `v` หมายถึง verbose (output filenames as they are archived)

  `f` หมายถึง file

  ตัวอย่าง

  ```
  $ tar -cvf play.tar play
  ```

เพื่อดูสิ่งที่เก็บอยู่ใน tar archive

```
tar -tvf archivename
```

ตัวอย่าง

```
$ tar -tvf play.tar
```

เพื่อนำ tar archive กลับมาใช้

```
tar -xvf archivename
```

ตัวอย่าง

```
$ tar -xvf play.tar
```

4.6 คำสั่งบีบอัดไฟล์

- **compress, gzip**

compress และ gzip เป็นยูทิลิตี้สำหรับ compressing และ decompressing ไฟล์ (อาจจะเป็นไฟล์ archive หรือไม่ก็ได้).

เพื่อที่จะบีบอัดไฟล์

```
compress filename
```

หรือ

```
gzip filename
```

ทั้ง 2 คำสั่งนี้ จะทำให้ filename ถูกลบและแทนที่ด้วยไฟล์บีบอัดชื่อใหม่เป็น filename.Z หรือ filename.gz

ตัวอย่าง

```
$ compress play.tar
```

จะได้ไฟล์บีบอัดชื่อ `play.tar.Z`

หรือ

```
$ gzip play.tar
```

จะได้ไฟล์บีบอัดชื่อ `play.tar.gz`

เพื่อทำให้เป็นไฟล์ปกติตามเดิม

```
compress -d filename
```

หรือ

```
gzip -d filename
```

ตัวอย่าง

```
$ compress -d play.tar.Z
```

หรือ

```
$ gzip -d play.tar.gz
```

แบบฝึกหัดท้ายบท

1. ทำขั้นตอนต่อไปนี้

1.1 ผู้สอนสร้างไฟล์ sawadee.txt ใน home directory โดยใช้คำสั่งดังนี้ (ผู้สอนใช้ login ชื่อ sa33)

prompt> **cat -u > sawadee.txt** ↵

แล้วพิมพ์ข้อความ 1 บรรทัด

1.2 ผู้เรียนใช้คำสั่ง **tail -f ~sa33/sawadee.txt** ↵

1.3 ผู้สอนพิมพ์ข้อความอีก 2 บรรทัด ผู้เรียนสังเกตดูบนจอของตน

2. ใช้คำสั่ง find เพื่อแสดงชื่อไฟล์ที่อยู่ใน /export/home

prompt> **find /export/home -print** ↵

3. ใช้คำสั่ง find เพื่อแสดงชื่อไฟล์ที่อยู่ใน /export/home โดยไม่ให้มีข้อความแจ้ง error

prompt> **find /export/home -print 2>home.err** ↵

4. ใช้คำสั่ง find เพื่อแสดงชื่อไฟล์ที่อยู่ใน /export/mail ที่มีขนาดไฟล์ใหญ่กว่า 200000 Bytes

prompt> **find /export/mail -size +200000c -print** ↵

5. ใช้คำสั่งเพื่อค้นหารายชื่อไฟล์ใน /export/home แล้วให้แสดงว่าแต่ละไฟล์เป็นชนิดใด ทำได้ 2 วิธี

prompt> **find . -print -exec file '{}' ';'** ↵ หรือ

prompt> **file `find /export/home -print` ↵**

6. ใช้คำสั่ง grep เพื่อดึงบรรทัดที่เป็น login ของคุณออกมา ทำได้ดังนี้

prompt> **grep $user /etc/passwd ↵**

7. ใช้คำสั่ง find, grep และ sort ร่วมกันในคำสั่ง เพื่อแสดงรายชื่อไฟล์ที่มีคำว่า hello อยู่ภายใน

prompt> **grep -i "hello" `find ~ -print` | sort ↵**

8. แปลความหมายของบรรทัดข้างล่างนี้

```
grep ^s4[4-6][1-3][0-9] /etc/passwd | cut -d: -f1 | sort
```

บทที่ 5 โพรเซส

วัตถุประสงค์

เนื้อหาในบทนี้ (เวลาโดยประมาณ 1 ชั่วโมง)

- หลักการของโพรเซส
- การควบคุมโพรเซส

5.1 หลักการของโพรเซส

โพรเซส (process) คือโปรแกรมที่อยู่ระหว่างการทำงาน

ทุกครั้งที่คุณสั่งให้ system utility หรือ application program ทำงานจาก shell ก็จะเกิดมีโพรเซส "child" เกิดขึ้น 1 หรือมากกว่าหนึ่งโพรเซสถูกสร้างจาก shell ในทันที

ทุกๆโพรเซสของยูนิกซ์จะแยกแยะได้จากสิ่งที่เรียกว่า a unique process identifier หรือ PID

โพรเซสที่สำคัญซึ่งเกิดขึ้นทุกครั้งคือโพรเซสชื่อ init มันเป็นโพรเซสแรกสุดที่เกิดขึ้นเมื่อระบบยูนิกซ์เริ่มต้นขึ้น มันมี PID เท่ากับ 1 แล้วโพรเซสอื่นๆทั้งหมดหลังจากนั้นก็อยู่ภายใต้โพรเซส init นี้

โอเปอเรเตอร์ไปป์ ('|') ใช้เพื่อสร้างโพรเซสที่ทำงานต่อเนื่องกันไปโดยส่งผ่านข้อมูลจากโพรเซสหนึ่งไปสู่อีกโพรเซสหนึ่งโดยตรง

5.2 การควบคุมโพรเซส

เชลล์ส่วนมากจะมี job control facilities ซึ่งทำให้คุณควบคุม job ต่าง ๆ ที่รันอยู่ ณ เวลาเดียวกันนั้นได้

ถ้าสมมติว่า ในขณะที่คุณกำลังแก้ไขไฟล์แล้วต้องการหยุดชั่วขณะเพื่อไปทำอะไรสักอย่าง โดยใช้ job control คุณจึงสามารถหยุดแก้ไขไว้ก่อน แล้วกลับสู่ shell prompt และเริ่มต้นงานอันอื่น หลังจากเสร็จงานนั้นแล้ว คุณก็สลับกลับไปแก้ไขไฟล์ที่ทำค้างอยู่นั้นต่อไปได้

คุณสามารถรันงานได้ทั้งใน foreground และ background

มันสามารถมีเพียง 1 งาน (job) ใน foreground ณ เวลาใด ๆ

สำหรับ foreground job เป็นการทำงานที่คุณสามารถรับข้อมูลจากคีย์บอร์ดและส่งผลลัพธ์ไปบนจอภาพได้แบบ interactive ส่วน background job นั้นไม่สามารถรับข้อมูลจากคีย์บอร์ดได้ แต่จะทำงานเงียบๆโดยไม่ต้องการข้อมูลจากผู้ใช้

เราสามารถหยุด foreground job ไว้ชั่วขณะโดยกดปุ่ม `Ctrl-z` และจะเรียก job นั้นว่า suspended job

เราสามารถสั่งให้ suspended job นั้นทำงานต่อไปใน foreground หรือ background โดยใช้คำสั่ง "`fg`" หรือ "`bg`" ตามลำดับ

โปรดสังเกตว่าการหยุดชั่วขณะนั้นไม่เหมือนกับการหยุดการทำงานโดยกดปุ่ม Ctrl-c ซึ่ง job นั้นจะตายไปเลยไม่สามารถทำงานต่อได้

เราสามารถรันคำสั่งเป็น Background jobs ได้โดยตรงจากบรรทัดคำสั่ง โดยเพิ่มตัวอักษร '&' หลังคำสั่ง

ตัวอย่าง

```
$ find / -name "*.txt" -print 1>output 2>/dev/null &
[1] 27501
$
```

สังเกตที่ [1] ที่ได้รับจากเชลล์แสดงถึงหมายเลข job ของ background process และเลข 27501 คือ PID ของโพรเซส

เพื่อที่จะดูว่ามี jobs ใดทำงานอยู่บ้าง

```
$ jobs
[1]+  Running  find / -name "*.txt" -print 1>output 2>/dev/null &
$
```

สังเกตว่าถ้าคุณมี job มากกว่า 1 job คุณสามารถอ้างถึง job นั้นด้วย `%n` เมื่อ n คือ หมายเลข job

ดังนั้น

```
$ fg %3
```

จึงเป็นการสั่งให้ job หมายเลข 3 กลับมาทำงานใน foreground

เมื่อต้องการรู้หมายเลขโพรเซสที่เป็นของเชลล์และ job ที่รัน ใช้คำสั่ง ps ดังนี้

```
$ ps
PID   TTY      TIME     CMD
17717 pts/10   00:00:00 bash
27501 pts/10   00:00:01 find
27502 pts/10   00:00:00 ps
```

ดังนั้น PID ของเชลล์ (bash) คือ 17717, PID ของ find คือ 27501 และ PID ของ ps คือ 27502

เมื่อต้องการหยุดโพรเซสหรือ job ให้ใช้คำสั่ง kill ซึ่งมีวิธีใช้คำสั่ง kill นี้ 2 วิธี โดย PID หรือโดย job number

ดังนั้น

```
$ kill %1
```

หรือ

```
$ kill 27501
```

จึงเป็นการหยุดโพรเซสของ find

จริง ๆ แล้วคำสั่ง kill นั้นเพียงแต่ส่ง signal ไปยังโพรเซสเพื่อขอทำการ shutdown และเลิก (the SIGTERM signal), ดังนั้นมันจึงอาจเป็นไปได้ว่าไม่สามารถใช้ได้ทุกกรณี

เราอาจใช้ -9 option (the SIGKILL signal) เพื่อหยุดโพรเซสแทน

```
$ kill -9 27501
```

นอกจากนี้คุณยังสามารถดูโพรเซสอื่นๆที่รันอยู่ในเครื่องนี้ได้ด้วยคำสั่งนี้

```
$ ps -fae
```

หรือ `ps -aux` บนเครื่องที่เป็น BSD

ส่วน ps -aeH แสดงข้อมูลโพรเซสตามลำดับชั้นแบบ hierarchy (รวมโพรเซส init ด้วย)

แบบฝึกหัดท้ายบท

1. พิมพ์คำสั่ง **sleep 15** ↵ แล้วพิมพ์คำสั่ง **ls** ต่อทันที (ทำได้ทันทีหรือไม่)

2. พิมพ์คำสั่ง **sleep 15 &** ↵ แล้วพิมพ์คำสั่ง **ls** ต่อทันที (ทำได้ทันทีหรือไม่)

3. ทำขั้นตอนต่อไปนี้

3.1 พิมพ์คำสั่ง **sleep 15** ↵ แล้วหยุดด้วยกดปุ่ม **Ctrl-z**

3.2 สั่งให้มันทำงานต่อไปใน background ด้วยคำสั่ง **bg** ↵

3.3 พิมพ์คำสั่ง **jobs** ↵

3.4 พิมพ์คำสั่ง **ps** ↵

3.5 พิมพ์คำสั่ง **fg** ↵ เพื่อให้คำสั่งกลับมาทำงานใน foreground

4. ทำขั้นตอนต่อไปนี้

4.1 พิมพ์คำสั่ง **sleep 100 &** ↵

4.2 พิมพ์คำสั่ง **sleep 200 &** ↵

4.3 พิมพ์คำสั่ง **sleep 300 &** ↵

4.4 พิมพ์คำสั่ง **jobs** ↵

5. ทำขั้นตอนต่อไปนี้

5.1 พิมพ์คำสั่ง **fg %2** ↵

5.2 กดปุ่ม **Ctrl-z**

5.3 พิมพ์คำสั่ง **bg %2** ↵

6. ทำขั้นตอนต่อไปนี้

6.1 พิมพ์คำสั่ง **ps -f** ↵ แล้วจดเลข PID ของ job 3

6.2 พิมพ์คำสั่ง **kill %1** ↵

6.3 พิมพ์คำสั่ง **kill PID** ↵

6.4 พิมพ์คำสั่ง **jobs** ↵

7. เปรียบเทียบคำสั่งนี้

7.1 รันคำสั่ง **sleep 300 &** ↵ แล้ว log out ออกไป

7.2 กลับเข้ามาใหม่ แล้วใช้คำสั่ง **ps -f** ↵ ยังเห็นโพรเซสอยู่หรือไม่

7.3 รันคำสั่ง **nohup sleep 300 &** ↵ แล้ว log out ออกไป

7.4 กลับเข้ามาใหม่ แล้วใช้คำสั่ง **ps -ef** ↵ ยังเห็นโพรเซสอยู่หรือไม่

7.5 สร้างไฟล์ run1 ด้วยคำสั่ง pico ดังนี้

prompt> **pico run1** ↵

ภายในเก็บข้อความข้างล่างนี้

```
sleep 60
echo Hello
sleep 60
```

```
echo Hi
sleep 60
echo Bye
```

7.6 เปลี่ยนสิทธิไฟล์ให้เป็น executable

prompt> **chmod u+x run1** ↵

7.7 รันคำสั่ง **./run1** ↵

7.8 ถ้าต้องการเก็บผลลัพธ์จากการรันคำสั่ง run1 ต้องใช้คำสั่งอย่างไร

7.9 ถ้าต้องการเก็บผลลัพธ์จากการรันคำสั่ง run1 แล้ว logout ออกไป แต่ให้โพรเซสนี้ทำงานต่อไป ต้องใช้คำสั่งอย่างไร

บทที่ 6 ยูทิลิตี้ของระบบ

วัตถุประสงค์

เนื้อหาในบทนี้ (เวลาโดยประมาณ 6 ชั่วโมง)

- การติดต่อไปยังเครื่องระยะไกล
- ยูทิลิตี้สำหรับตรวจสอบเครือข่าย
- การถ่ายโอนไฟล์
- การติดต่อระหว่างผู้ใช้
- ยูทิลิตี้ด้านอีเมล์
- ยูนิกซ์เอดิเตอร์

6.1 การติดต่อไปยังเครื่องระยะไกล

- **ssh** *machinename* (secure shell)

ssh เป็นโปรแกรมที่มาแทนที่คำสั่ง telnet เนื่องจาก telnet ขาดคุณสมบัติด้านความปลอดภัยของข้อมูลในระหว่างการส่งแพกเกจไปบนสายเคเบิล ssh จะมีการเข้ารหัส encrypted ข้อมูลก่อนส่งไปยังอีกเครื่อง

ssh ไม่ได้เป็น standard system utility แต่อย่างไรก็ตามมันก็ถูกจัดให้เป็น de facto standard คุณสามารถอ่านเพิ่มเติมได้ที่ http://www.ssh.org/

ssh clients ก็มีให้ใช้งานบนเครื่อง Windows เช่นกัน โปรแกรมที่ดีตัวหนึ่งชื่อว่า putty

ตัวอย่าง

```
$ ssh sa33@maliwan.psu.ac.th
```

จะเป็นการติดต่อไปยังเครื่อง maliwan เพื่อขอเข้าใช้ในชื่อผู้ใช้ sa33

ในครั้งแรกที่ติดต่อนั้น โปรแกรมจะสร้าง fingerprint ของทั้งสองเครื่องที่จะติดต่อกัน และจะถามให้ผู้ใช้ยืนยันว่าถูกต้อง ให้ตอบว่า yes (เขียนเต็ม)

6.2 ยูทิลิตี้สำหรับตรวจสอบเครือข่าย

- **ping** *machinename*

ยูทิลิตี้ ping มีประโยชน์สำหรับการเช็คระยะเวลาในการตอบสนองระหว่างเครื่อง 2 เครื่อง เช่น

```
$ ping www.uni.net.th
```

จะวัดระยะเวลาในการตอบว่ามี delay ระหว่างเครื่องที่กำลังใช้กับเว็บเซิร์ฟเวอร์ที่ UNINET และ ping ยังใช้เพื่อเช็คว่า เครื่องที่เราต้องการติดต่อด้วยยังมีอยู่หรือไม่

- **traceroute** *machinename*

traceroute แสดงเส้นทางทั้งหมดเพื่อไปยังเครื่องที่อยู่ไกล และรวมถึงเวลาที่ล้าช้าในการผ่านเครื่องแต่ละเครื่องตลอดเส้นทาง มันใช้เพื่อติดตามดูว่าจุดไหนที่เน็ตเวอร์คใช้ไม่ได้ เช่น

```
$ traceroute www.uni.net.th
```

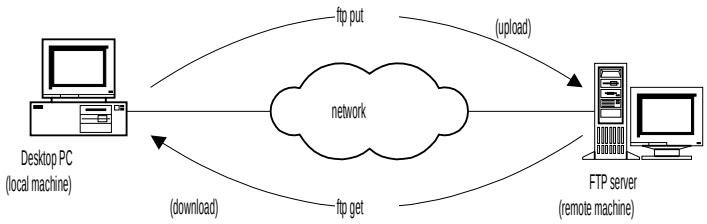
จะเป็นการตรวจสอบเส้นทางจากเครื่องที่กำลังใช้ไปจนถึงเว็บเซิร์ฟเวอร์ที่ UNINET

6.3 การถ่ายโอนไฟล์

- **ftp** *machinename* (file transfer protocol)

ftp เป็นวิธีถ่ายโอนไฟล์แบบหนึ่งแต่ไม่มีความปลอดภัย เมื่อคุณติดต่อไปยังเครื่องหนึ่งด้วย ftp คุณจะถูกถาม ชื่อผู้ใช้ และ รหัสผ่าน ถ้าคุณมีบัญชีบนเครื่องนั้น คุณก็สามารถใช้งานได้ หรือคุณอาจใช้ชื่อผู้ใช้เป็น "ftp" หรือ "anonymous" แทนก็ได้ หากว่าเครื่องนั้นเป็น ftp server ที่บริการให้ดาวน์โหลดไฟล์ได้

เมื่อคุณล็อคอินผ่าน FTP คุณสามารถดูรายชื่อไฟล์ (dir), รับไฟล์ (get และ mget) และส่งไฟล์ (put และ mput) และคำแนะนำ (help) จะบอกว่ามีคำสั่งอะไรบ้าง เมื่อต้องการเลิกก็พิมพ์ quit เพื่อกลับสู่ shell prompt



FTP site ที่ให้บริการจะแยกได้เป็น 2 ลักษณะคือ

1. Authorized FTP site คือเครื่อง server ที่บริการเฉพาะผู้ใช้งานที่มีบัญชีผู้ใช้อยู่ในเครื่องนั้น เช่น mail server ชื่อ ratree ก็ให้บริการ FTP เพื่อให้ผู้ใช้สามารถ download และ upload ไฟล์ ได้

ตัวอย่างเช่น

prompt> **ftp maliwan.psu.ac.th**

username: **sa33**

password: ใส่รหัสผ่านของ **sa33**

คำสั่ง get ใช้เพื่อดาวน์โหลด

ftp> **get local.login**

หรือใช้คำสั่ง mget * เพื่อเลือกชื่อไฟล์ที่ดาวน์โหลด

ftp> **mget ***

คำสั่ง put ใช้เพื่ออัปโหลด

ftp> **put my.doc**

หรือใช้คำสั่ง mput * เพื่อเลือกชื่อไฟล์ที่จะอัปโหลด

ftp> **mput ***

ftp> **quit**

prompt>

2. Anonymous FTP site คือเครื่อง server ที่บริการทุกคน แม้ว่าจะไม่มีบัญชีผู้ใช้อยู่ในเครื่องนั้นก็ตาม โดยใช้ชื่อบัญชีผู้ใช้ (user name) ว่า Anonymous และใส่ password ด้วย e-mail address ใด ๆ ก็ได้ FTP server แบบนี้โดยทั่วไปมักจะอนุญาตให้ download ได้อย่างเดียว และให้บริการเก็บโปรแกรมชนิดที่แจกฟรี (freeware) หรือทดลองใช้ (evaluatation) เป็นต้น

ตัวอย่างเช่น

prompt> **ftp ftp.psu.ac.th**

username: **anonymous**

password: **anonymous@ftp.psu.ac.th**

ftp> **cd pub/windows-utils**

ftp> **get lftp13.zip**

หรือใช้คำสั่ง mget * เพื่อเลือกชื่อไฟล์

ftp> **mget ***

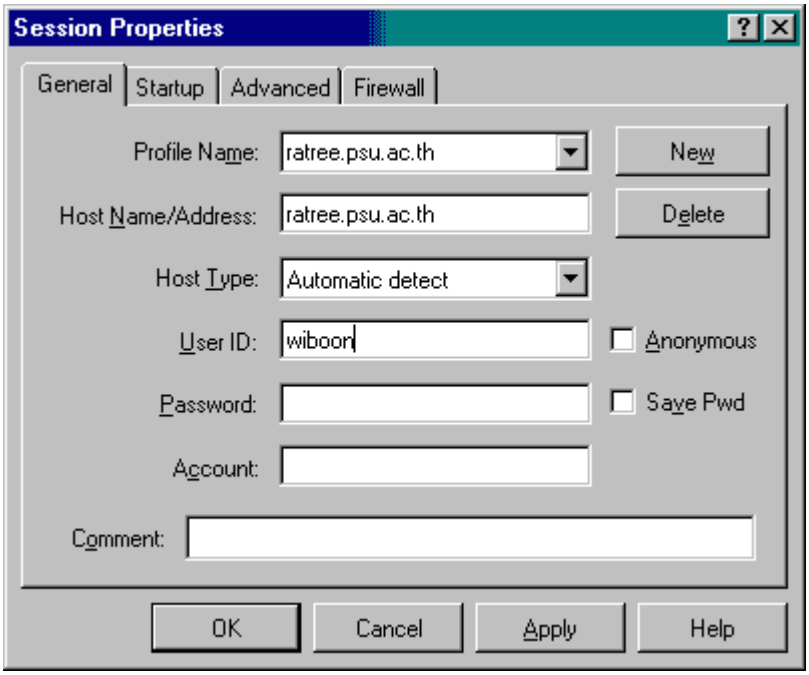
ftp> **quit**

prompt>

คำสั่งที่ใช้มากอีกคำสั่งคือ **lcd** ใช้เพื่อกำหนดไดเรกทอรีที่จะเก็บไฟล์ที่ดาวน์โหลดมา เช่น ถ้าใช้วินโดวส์ **lcd c:\download** ถ้าใช้ลินุกซ์ **lcd ~/download** เป็นต้น

นอกจากนี้ยังมีการใช้งานแบบกราฟิก (Windows Graphical User Interface) คือสั่งทำงานด้วยการใช้ mouse คลิกและลากไฟล์จากช่องคอลัมน์ฝั่ง remote ไปยังคอลัมน์ฝั่ง local อย่างง่ายดาย ดังรูป

ขั้นตอนการติดต่อด้วยโปรแกรม WS_FTP สำหรับวินโดวส์



การกำหนดค่าต่าง ๆ ที่ใช้ในการติดต่อ ดังนี้

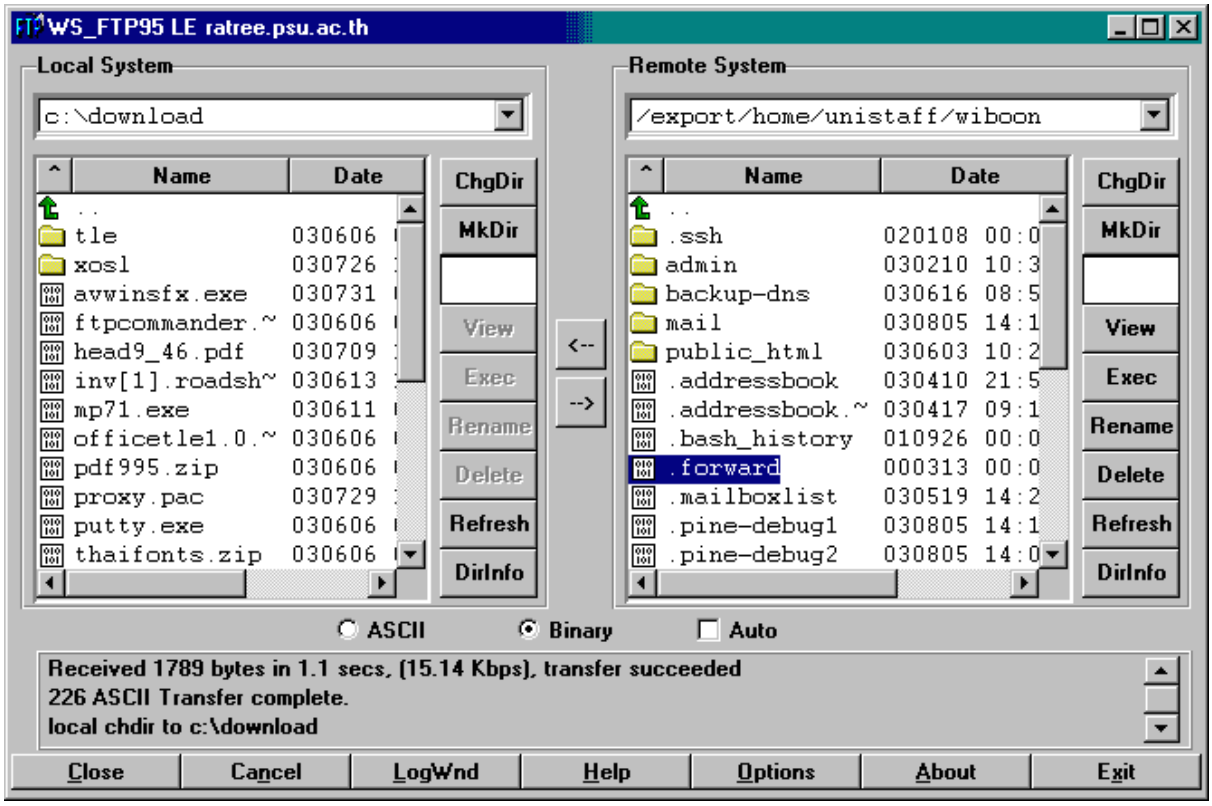
Profile Name: คือ ส่วนที่ใส่ชื่อคำอธิบายประกอบ Site เช่น ratree.psu.ac.th

Host Name/Address: คือ ส่วนที่ใส่ชื่อ site ที่ต้องการถ่ายโอนข้อมูล เช่น ratree.psu.ac.th

User ID: คือ ส่วนที่ใส่ Username ของผู้ใช้งาน

Password: คือ พิมพ์รหัสผ่าน password ของผู้ใช้งาน

ขั้นตอนการถ่ายโอนไฟล์



คลิกชื่อไฟล์หรือไดเรกทอรี แล้วคลิกที่รูปลูกศร (ชี้ซ้าย เป็นการดาวน์โหลด ชี้ขวา เป็นการอัปโหลด)

- **scp** *sourcefiles destination* (secure copy)

scp เป็นวิธีถ่ายโอนไฟล์แบบหนึ่งที่มีความปลอดภัย มันทำงานคล้ายกับคำสั่ง cp นอกจากว่ามันรับค่า user และ machine เช่นเดียวกับ file ตัวอย่างเช่น

```
$ scp sa33@maliwan.psu.ac.th:~/hello.txt
```

จะคัดลอกไฟล์ hello.txt จากผู้ใช้ชื่อ sa33 ที่เครื่อง maliwan.psu.ac.th ไปเก็บไว้ใน current directory (.) บนเครื่องที่ใช้งาน

- **wget** *URL*

wget เป็นวิธีการไปเอาไฟล์มาจากเว็บเซิร์ฟเวอร์ (โดยใช้โปรโตคอล HTTP)

wget เป็นการใช้งานแบบ non-interactive หมายความว่ามันทำงานอยู่ใน background ในขณะที่ผู้ใช้ไม่ได้ล็อคอิน

เนื้อหาที่เอามาด้วย wget ถูกเก็บอยู่ในรูป HTML text (ซึ่งสามารถอ่านดูได้ในภายหลังด้วย web browser)

ตัวอย่าง

```
$ wget http://www.yahoo.com
```

จะไปเอาเว็บเพจของ yahoo มาเก็บไว้ที่ current directory

6.4 การติดต่อระหว่างผู้ใช้

- **write**

write ใช้ได้กับผู้ใช้ที่อยู่บนเครื่องเดียวกันที่ต้องการส่งข้อความคุยกัน คุณจะต้องระบุชื่อผู้ใช้หรือบางทีอาจต้องระบุเทอร์มินัลที่ใช้ด้วย

```
$ write sa33 pts/2
hello sa33
.
```

ข้อความทั้งหมดจะถูกส่งไปก็ต่อเมื่อคุณใส่เครื่องหมาย . (dot) การกลับไปสู่ shell prompt ให้กดปุ่ม Ctrl-d

- **talk**

talk ใช้งานแบบ interactive คุยกันได้ระหว่างเครื่อง 2 เครื่อง

```
$ talk sa33@maliwan.psu.ac.th
```

จะเป็นการขอคุยกับผู้ใช้ sa33@maliwan.psu.ac.th เมื่อ sa33@maliwan.psu.ac.th พิมพ์คำสั่ง talk กับมา ก็เป็นการสร้างคอนเนคชันในการคุยกัน

- **mesg**

mesg n ปฏิเสธการสร้าง connection , mesg y อนุญาตให้มีการสร้าง connection

6.5 ยูทิลิตี้ด้านอีเมล์

- **mail**

  mail เป็นยูทิลิตี้มาตรฐานบน UNIX สำหรับส่งและรับอีเมล์

  คุณสามารถใช้คำสั่ง mail เพื่อส่งอีเมล์ได้โดยตรงจากบรรทัดคำสั่ง ตัวอย่างเช่น

  ```
  $ mail -s "Hi" sa33@maliwan.psu.ac.th < message.txt
  ```

  จะส่งข้อความ message.txt ไปยังผู้รับ sa33@maliwan.psu.ac.th ด้วย subject "Hi"

- **pine**

  pine ใช้งานได้ง่ายกว่า mail แต่เป็น non-standard บางทีต้องติดตั้งโปรแกรมเพิ่มเอง pine จะมีคำแนะนำการใช้งานอยู่ในโปรแกรม

6.6 ยูนิกซ์เอดิเตอร์

- **vi** *filename*

  vi (อ่านว่า วี-ไอ, ย่อมาจาก visual) เป็นเท็กซ์เอดิเตอร์ชนิด display-oriented

การเริ่มต้นใช้ vi

```
$ vi filename
```

เมื่อ filename คือชื่อไฟล์ที่คุณจะเขียน ถ้ามันไม่มีอยู่ vi จะสร้างไฟล์ว่างเปล่าไว้ให้

การใช้งาน

vi มี 2 สถานะ: สถานะคำสั่ง (command mode) และสถานะป้อนข้อความ (input mode)

ในสถานะคำสั่ง ตัวอักษรที่คุณกด จะหมายถึงการกระทำ (เช่น ย้ายเคอร์เซอร์, ตัดหรือคัดลอกข้อความ, ฯลฯ) ในสถานะป้อนข้อความ ตัวอักษรที่คุณพิมพ์จะแทรกหรือเขียนทับข้อความเดิม

เมื่อคุณเริ่มต้น vi, มันอยู่ในสถานะคำสั่ง คุณเปลี่ยนเป็นสถานะป้อนข้อความได้โดยกดปุ่ม **i**

(insert) คุณเปลี่ยนกลับเป็นสถานะคำสั่งด้วยกดปุ่ม **ESC** (the escape key)

ในสถานะคำสั่ง คุณสามารถเคลื่อนที่เคอร์เซอร์ไปทั่วเอกสารด้วยกดปุ่ม **h**, **j**, **k** และ **l** เคลื่อนที่ไปซ้าย, ลง, ขึ้น และ ขวา ตามลำดับ (และถ้าเทอร์มินัลของคุณใช้ได้ดี คุณก็จะใช้ปุ่มลูกศรได้)

การลบ

เมื่อจะลบข้อความ ให้ย้ายเคอร์เซอร์ไปบนตัวอักษรนั้น และแน่ใจว่าอยู่ในสถานะคำสั่ง

กด

x เพื่อลบ 1 ตัว

dw เพื่อลบ 1 คำถัดไป, **d4w** เพื่อลบ 4 คำถัดไป,

dd เพื่อลบ 1 บรรทัดถัดไป, **4dd** เพื่อลบ 4 บรรทัดถัดไป

d$ เพื่อลบไปจนจบบรรทัด หรือ **dG** เพื่อลบไปจนจบเอกสาร

ถ้าคุณลบพลาดมากไป ให้กดปุ่ม **u** จะเป็นการเรียกกลับคืน (undo)

การย้ายและคัดลอกข้อความ

เมื่อต้องการย้ายข้อความ, ลบข้อความด้วยคำสั่งเช่น **5dd** เพื่อลบ 5 บรรทัด จากนั้นย้ายเคอร์เซอร์ไปที่จะนำมันมาวาง แล้วกดปุ่ม **p**

เมื่อต้องการคัดลอกข้อความ, ลอกข้อความด้วยคำสั่งเช่น **5yy** เพื่อคัดลอก 5 บรรทัด จากนั้นย้ายเคอร์เซอร์ไปที่จะนำมันมาวาง แล้วกดปุ่ม **p**

การค้นหาคำ

เมื่ออยู่ในสถานะคำสั่ง คุณสามารถค้นหาคำโดยระบุ regular expressions

การค้นหาไปข้างหน้า พิมพ์เครื่องหมาย **/** (slash) จากนั้นใส่ regular expression แล้ว

กดปุ่ม Enter

การค้นหาย้อนกลับ พิมพ์เครื่องหมาย **?** แทน **/**

การค้นหาคำต่อไปให้กดปุ่ม **n**

การแทนที่คำ

การแทนที่คำทุกคำที่ค้นหาได้ตาม pattern1 ด้วย pattern2, พิมพ์ **:**
%s/pattern1/pattern2/g

แทนการใช้เครื่องหมาย % ด้วยช่วงบรรทัด (เช่น **1,17**) เพื่อกำหนดช่วงที่ต้องการเท่านั้น

การเลิก

การบันทึก, พิมพ์ **:w**

การบันทึกและเลิก, พิมพ์ **:wq**

การเลิกโดยไม่บันทึก, พิมพ์ **:q!**

แบบฝึกหัดท้ายบท

1. ใช้คำสั่ง ssh ขอเข้าใช้งานในชื่อผู้ใช้ sa32@maliwan.psu.ac.th

prompt> **ssh sa32@maliwan.psu.ac.th** ↵

2. ใช้คำสั่ง ping ตรวจสอบว่า www.uni.net.th ให้บริการได้อยู่

prompt> **/usr/sbin/ping www.uni.net.th** ↵

3. ใช้คำสั่ง traceroute ตรวจสอบดูว่าเส้นทางไปยัง www.google.com ผ่านสถานที่ใดบ้าง

prompt> **/usr/local/bin/traceroute www.google.com** ↵

4. ใช้คำสั่ง ftp เข้าไปยังเครื่อง ftp.psu.ac.th แล้วดาวน์โหลดไฟล์

prompt> **ftp ftp.psu.ac.th** ↵

username: **anonymous** ↵

password: **anonymous@ftp.psu.ac.th** ↵

ftp> **cd pub/unix-util** ↵

ftp> **get txt2html-1.28.tar.gz** ↵

ftp> **quit** ↵

การนำ *txt2html-1.28.tar.gz* ไปใช้ *(ถ้าต้องการทำ)*

prompt> gzip -d txt2html-1.28.tar.gz

prompt> tar -xvf txt2html-1.28.tar

prompt> cd txt2html-1.28

prompt> perl txt2html.pl README > readme.html

prompt> cat readme.html

5. ใช้คำสั่ง scp เพื่อคัดลอกไฟล์ important จากผู้ใช้ sa33

prompt> **scp sa33@maliwan.psu.ac.th:~/important .** ↵

6. ใช้คำสั่ง wget ดึงข้อมูลเว็บเพจ และ zip file ดังนี้

prompt> **wget http://ratree.psu.ac.th/index.html** ↵

prompt> **wget ftp://ftp.psu.ac.th/pub/unix-util/txt2html-1.28.tar.gz** ↵

7. ใช้คำสั่ง write ส่งข้อความไปให้เพื่อนข้าง ๆ

ลองใช้คำสั่ง mesg n แล้วให้เพื่อนส่งข้อความมา

8. ส่งเมล์ให้ผู้สอน โดยมีข้อมูลชื่อไฟล์ทั้งหมดใน home

prompt> **find ~ -print > myhome** ↵

prompt> **mail -s "my home" sa33@maliwan.psu.ac.th < myhome** ↵

9. ทำขั้นตอนต่อไปนี้

9.1 ใช้คำสั่ง wget เพื่อดาวน์โหลดไฟล์ตัวอย่างนำมาแก้ไขด้วย vi

wget http://ratree.psu.ac.th/~wiboon/download/unix/a.sh ↵

9.2 แก้ไขด้วยเอดิเตอร์ vi ให้ได้ดังนี้

```
##!/bin/sh
echo -n "Enter 2 digits B.C. year: "
read yr
yr=$yr
echo "**************************************"
echo "Listing for login and uid only year $yr"
echo "**************************************"
grep "^s$yr" /etc/passwd | cut -d: -f1,3
echo "**************************************"
```

9.3 ถ้าแก้ไขไฟล์ a.sh ถูกต้อง เมื่อรันด้วยคำสั่งนี้ ./a.sh จะได้ผลลัพธ์คืออะไร

บรรณานุกรม

1. Introduction to UNIX, Dr William J. Knottenbelt, Department of Computing, Imperial College London, South Kensington Campus, London, September 2001,
Website http://www.doc.ic.ac.uk/~wjk/UnixIntro/

2. Introduction to UNIX, Blaise Barney, the Maui High Performance Computing Center, March 1995, Website
http://www.mhpcc.edu/training/vitecbids/UnixIntro/