ลินุกซ์ ๑๐๑

แก่นความรู้ลินุกซ์

อิสระของการใช้งาน, การพัฒนา, การศึกษา, และการแจกจ่าย

หนังสือเล่มนี้เขียนด้วยบรรณาธิกรณ์ `emacs`, เรียงพิมพ์ด้วย LaTeX, ภาพประกอบใช้โปรแกรม `xfig`, `inkscape` และ `dia`. ระบบปฏิบัติการที่ใช้เตรียมหนังสือเล่มนี้ได้แก่ระบบปฏิบัติการลินุกซ์

คำนำ

"For the things we have to learn before we can do them,
we learn by doing them."
Aristotle

ลินุกซ์ไม่ใช่สิ่งใหม่ เป็นสิ่งที่มีมานานมากกว่า 10 ปี และพัฒนาต่อไปเรื่อย ๆ. คนที่ใช้ลินุกซ์อยู่แล้วก็มีมาก, คนที่กำลังเริ่มใช้ลินุกซ์ก็มีไม่น้อย. แต่ถ้าจะถามหาหนังสือลินุกซ์ภาษาไทยที่อธิบายแก่นความรู้โดยรวมแล้ว อาจจะตอบได้ว่ายังไม่มี.

หนังสือเล่มนี้ต้องการเป็นสื่อถ่ายทอดแก่นความรู้ที่เกี่ยวกับลินุกซ์. แก่นความรู้นี้หมายถึงความรู้พื้นฐานที่สามารถขยายผลต่อไปเรื่อย ๆ. แก่นความรู้นี้อาจจะล้าสมัยในบางมุมแต่เป็นความรู้ที่ไม่ตาย. แก่นความรู้นี้ไม่ใช่สิ่งที่เกิดจากการท่องจำแต่เป็นสิ่งที่เกิดจากการเข้าใจและใช้งานจริง. ใครที่ต้องการจะเรียนรู้ต้องใช้เวลาและความอดทน. แต่เมื่อเรียนรู้และเข้าใจแล้วก็จะสามารถแก้ปัญหาที่ไม่เคยเจอมาก่อนได้ และต่อยอดความรู้ไปได้เรื่อย ๆ. หนังสือเล่มนี้จะไม่สอนให้คลิกโน่นคลิกนี่. วันนี้โปรแกรมนี้อาจจะคลิกตรงนี้, วันหน้าโปรแกรมเดียวกันอาจจะคลิกที่อื่นก็ได้. สิ่งที่ควรรู้คือ "ทำไม" ต้องทำอย่างนั้น, ต้องทำอย่างนี้. แต่ก็ไม่ได้หมายความว่าเราไม่ต้องจำอะไรเลย. สิ่งที่สำคัญคือเราต้องจำในสิ่งที่จำเป็น, จำเท่าที่พอดี. เหมือนกับเราต้องจำว่า $2 \times 3 = 6$ เพื่อที่จะใช้คำนวณโจทย์ที่ยากขึ้นต่อไป.

ถึงแม้ว่าผมใช้ลินุกซ์และเขียนหนังสือเกี่ยวกับลินุกซ์แต่ผมก็ไม่สามารถบอกได้ว่าลินุกซ์ดีไปเสียทุกอย่าง. บางอย่างเราต้องยอมรับความไม่สมบูรณ์ของสรรพสิ่ง ซึ่งก็เป็นความจริงสำหรับระบบปฏิบัติการอื่น ๆด้วย. สิ่งที่สำคัญคือผู้ใช้ต้องรู้จักเลือก, รู้จักใช้ให้เหมาะกับประเภทงานและความถนัดตามวิจารณญาณของตัวเอง.

สุดท้ายนี้ต้องขอขอบพระคุณทุกท่านที่มีส่วนเกี่ยวข้องทำให้หนังสือเล่มนี้ออกมาเป็นรูปเล่มได้ และหวังว่าหนังสือเล่มนี้จะมีประโยชน์ต่อสังคมและการพัฒนาประเทศไม่มากก็น้อย.

พูลลาภ วีระธนาบุตร

สารบัญ

สารบัญรูป

สารบัญตาราง

บทที่ 1
รู้จักกับลินุกซ์

```
From:  tovalds@klaava.Helsinki.Fi (Linus Benedict Torvalds)
To:  Newsgroups:  comp.os.minix
Subject:  What would you like to see most in minix?
Summary:  small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.Fi>

Hello everybody out there using minix-I'm doing a (free) operating system (just a hobby, won't be
big and professional like gnu) for 386 (486) AT clones.  This has been brewing since april, and
is starting to get ready.  I'd like any feedback on things people like/dislike in minix, as my OS
resembles it somewhat (same physical layout of the file-system (due to practical reasons) among
otherthings).

I've currently ported bash (1.08) and gcc (1.40), and things seems to work.  This implies
that I'll get something practical within a few months, and I'd like to know what features most
people would want.  Any suggestions are welcome, but I won't promise I'll implement them:-)

                                                                    Linus (torvalds@kruuna.helsinki.fi)

PS. Yes-it's free of any minix code, and it has a multi-threaded fs.  It is NOT portable (uses
386 task switching etc.), and it probably never will support anything other than AT-harddisks, as
that's all I have:-(.
```

เรื่องราวของ*ลินุกซ์ (Linux)* เริ่มมาจากข้อความข้างบนที่ส่งถึง Minix *นิวส์กรุป (newsgroup)* เมื่อปี ค.ศ.1991. ข้อความฉบับนี้เขียนโดยนักศึกษาชาวฟินแลนด์ที่มีชื่อว่า Linus Benedict Torvalds เพื่อแนะนำ *ระบบปฏิบัติการ (Operating System, OS)* ที่เขาสร้างขึ้น. ใครจะคาดคิดบ้างว่าในเวลาต่อมา, ระบบปฏิบัติการที่เขาสร้างขึ้นที่เรียกว่า *Linux* จะเป็นระบบปฏิบัติการที่ใช้กันอย่างแพร่หลายในปัจจุบันและเป็นคู่แข่งที่น่ากลัวของระบบปฏิบัติการอื่น ๆ.

คำว่า “ลินุกซ์” เป็นศัพท์ที่บัญญัติโดยราชบัณฑิตยสถาน. ชาวต่างชาติบ้างก็อ่านออกเสียงว่า “ไลนักซ์”. ทางญี่ปุ่นอ่านออกเสียงว่า “ลินักซ์”.

Minix ▶
ระบบปฏิบัติการคล้ายเหมือนยูนิกซ์ที่สร้างขึ้นโดยศาสตราจารย์ Andrew Tanenbaum เพื่อการเรียนการสอนเกี่ยวกับระบบปฏิบัติการ

1.1 Linux คืออะไร?

“Linux” หรือที่เรียกในภาษาไทยว่า “ลินุกซ์”,เป็นระบบปฏิบัติการที่สร้างโดยนาย Linux Benedict Torvalds ชาวฟินแลนด์ขณะที่เขาเป็นนักศึกษาอยู่ที่มหาวิทยาลัย Helsin-

ki. หลังจากที่เขาเผยแพร่*รหัสต้นฉบับ (source code)* ทางอินเทอร์เน็ตแล้วทำให้มี*นักพัฒนาซอฟต์แวร์ (software developer)* ซึ่งส่วนมากเป็นอาสาสมัครอยู่ทั่วโลกช่วยกันพัฒนาตัวระบบปฏิบัติการเอง, รวมถึงโปรแกรมต่าง ๆที่ใช้ในระบบด้วย. กล่าวได้ว่าระบบปฏิบัติการลินุกซ์เป็นระบบปฏิบัติการที่เกิดจากอินเทอร์เน็ตและพัฒนาบนอินเทอร์เน็ต.

multi-users ▶
มัลติยูสเซอร์. ความสามารถของระบบปฏิบัติการที่ให้ผู้ใช้หลายคนทำงานได้ในเวลาเดียวกัน

multi-tasks ▶
มัลติทาส์ก. ความสามารถในการแบ่งเวลาการทำงานของหน่วยประมวลผลเมื่อมีงานหลายอย่างที่ต้องทำ. เป็นผลให้ดูเหมือนว่าหน่วยประมวลผลข้อมูลสามารถทำงานหลายอย่างได้ในเวลาเดียวกัน.

ลินุกซ์เป็นระบบปฏิบัติการแบบ*มัลติยูสเซอร์ (multi-users)* และ *มัลติทาส์ก (multi-tasks)* ที่ใช้ได้กับระบบคอมพิวเตอร์หลายประเภท. ระบบคอมพิวเตอร์ที่ลินุกซ์ใช้กันอย่างแพร่หลายได้แก่ระบบ*คอมพิวเตอร์ส่วนบุคคล (Personal Computer, PC)* ที่ใช้*หน่วยประมวลผล (Processor)* ที่มา*สถาปัตยกรรม (architechture)* แบบ CISC (Complex Instruction Set Computer) เช่นหน่วยประมวลผลตระกูล Intel, AMD เป็นต้น. ลินุกซ์ยังสามารถใช้งานกับระบบคอมพิวเตอร์ที่ใช้หน่วยประมวลผลอื่น ๆได้ด้วยเช่น MIPS, SPARC, PowerPC ฯลฯ, โดยแก้ไขรหัสต้นฉบับของลินุกซ์บางส่วน.

ลินุกซ์เป็นระบบปฏิบัติการที่คล้ายเหมือนกับระบบปฏิบัติการ*ยูนิกซ์ (UNIX)*. กล่าวคือ, ลินุกซ์ออกแบบโดยใช้*แนวความคิด (concept)* ของระบบปฏิบัติการยูนิกซ์เป็นต้นแบบ. แต่ลินุกซ์เริ่มสร้างขึ้นโดยนาย Linus Torvalds เพียงลำพังโดยไม่ได้ลอกเลียนหรือสืบทอดรหัสต้นฉบับจากยูนิกซ์แต่อย่างใด. ลินุกซ์ถูกสร้างขึ้นมาให้ทำงานทุกอย่างที่ระบบปฏิบัติการยูนิกซ์ทำได้โดยยึดมาตรฐาน *POSIX (Portable Operating System Interface based on UNIX)*. ซึ่งหมายความว่าโปรแกรมที่ใช้บนลินุกซ์มี*ความเข้ากันได้ (compatibility)* กับระบบปฏิบัติการยูนิกซ์อื่น ๆในระดับรหัสต้นแบบ. กล่าวคือสามารถนำรหัสต้นฉบับของโปรแกรมที่ใช้ในลินุกซ์ไป*คอมไพล์ (compile)* บนระบบปฏิบัติการยูนิกซ์อื่น ๆและ*กระทำการ (execute)* ได้โดยไม่ต้องแก้ไขรหัสต้นฉบับหรือแก้ไขเพียงเล็กน้อยเท่านั้น. ในทางกลับกัน, ผู้ใช้สามารถนำรหัสต้นฉบับของโปรแกรมที่ใช้ในระบบปฏิบัติการยูนิกซ์อื่น ๆมาคอมไพล์แล้วกระทำการบนระบบปฏิบัติการลินุกซ์ได้เช่นเดียวกัน.

POSIX ▶
ข้อกำหนดมาตรฐานว่าระบบปฏิบัติการที่สามารถใช้งานได้กับฮาร์ดแวร์ต่างระบบนั้นต้องมีรายละเอียดคุณสมบัติอย่างไร. มาตรฐานนี้กำหนดโดยองค์กร The Institute of Electrical and Electronic Engineers (IEEE).

การที่นำรหัสต้นฉบับของโปรแกรมบนระบบหนึ่งไปแก้ไขและคอมไพล์บนระบบอื่นเรียกว่าการ porting. โปรแกรมที่มีคุณสมบัตินี้เรียกได้ว่า portable หรือมีความสามารถในการ port (portability).

เมื่อกล่าวถึงคำว่า “ลินุกซ์” เดี่ยว ๆ, เราสามารถตีความได้หลายความหมาย. ในความหมายเฉพาะเรื่อง, ลินุกซ์หมายถึงแก่นของระบบปฏิบัติการที่เรียกว่า*เคอร์เนล (kernel)* หรือเรียกให้ชัดเจนว่า *ลินุกซ์เคอร์เนล (linux kernel)* ซึ่งเป็นโปรแกรมพิเศษ, ทำหน้าแบ่งเวลาทำงานของหน่วยประมวลผลให้*โปรเซส (process)* ต่าง ๆ, จัดการการใช้*หน่วยความจำ (memory)*, ควบคุมการทำงานของ*ดีไวส์ (device)* ไม่ว่าจะเป็น*ฮาร์ดดิกส์ (hard disk)*, *I/O Port* ฯลฯ. ในความหมายโดยรวมหรือความหมายทั่วไป, ลินุกซ์หมายถึง *Operating Enviroment* ซึ่งหมายถึงเคอร์เนลและกลุ่มของซอฟต์แวร์ต่างที่นำมารวมกันให้เป็นระบบ. ตัวอย่างโปรแกรมต่าง ๆที่ใช้กับระบบปฏิบัติการลินุกซ์ได้แก่ *ระบบ X วินโดว์ (X window system)*, *ระบบเดกส์ท็อป (desktop environment)*, *คอมไพเลอร์ (compiler)*, *อินเทอร์พรีเตอร์ (interpreter)*, *บรรณาธิกรณ์ (editor)* เป็นต้น. สำหรับบุคคลทั่วไปถ้าพูดถึง “ลินุกซ์” จะหมายถึง Operating Environment แต่สำหรับนักพัฒนาซอฟต์แวร์หรือโปรแกรมเมอร์, “ลินุกซ์” อาจจะมีความหมายลึกกว่านั้นซึ่งเจาะจงถึงเคอร์เนล.

1.2 ประวัติความเป็นมาของลินุกซ์

ระบบปฏิบัติการลินุกซ์เริ่มสร้างในปี ค.ศ. 1991 โดยนักศึกษามหาวิทยาลัยชาวฟินแลนด์ที่ชื่อ Linus Torvalds [2]. Linus เป็นคนที่สนใจและคลุกคลีกับคอมพิวเตอร์มาตั้งแต่

เด็ก. เมื่อเขาเป็นนักศึกษาปีหนึ่งที่มหาวิทยาลัย Helsinki, เขาได้รู้จักกับระบบปฏิบัติการยูนิกซ์ซึ่งแตกต่างจากระบบปฏิบัติการที่ใช้กับคอมพิวเตอร์ส่วนบุคคล. การที่เขาได้รู้จักกับยูนิกซ์นี้เองเป็นจุดเริ่มต้นทำให้เขาสนใจศึกษาการทำงานของระบบปฏิบัติการ.

หนังสือ *Operating Systems: Design and Implementation* เป็นหนังสือที่เขาอ่านประกอบการเรียนเกี่ยวกับเรื่องระบบปฏิบัติการ. หนังสือเล่มนี้เขียนโดย Andrew Tanenbaum ซึ่งเป็นอาจารย์มหาวิทยาลัยในประเทศเนเธอร์แลนด์. Andrew Tanenbaum นอกจากจะเขียนหนังสือเกี่ยวกับระบบปฏิบัติการแล้วเขายังสร้างระบบปฏิบัติการขนาดเล็กคล้ายยูนิกซ์ที่มีชื่อว่า *มินิกซ์ (Minix)* สำหรับการเรียนการสอนระบบปฏิบัติการอีกด้วย. หนังสือเล่มนี้เป็นแรงบันดาลใจและเป็นชนวนความคิดให้ Linus สร้างระบบปฏิบัติการด้วยตัวเอง.

ในปี 1991, Linux ซื้อคอมพิวเตอร์ส่วนบุคคลมาใช้งาน. คอมพิวเตอร์ที่เขาซื้อเป็นคอมพิวเตอร์ส่วนบุคคลที่ใช้หน่วยประมวลผลของ Intel รุ่น 80386 และระบบปฏิบัติการที่มาพร้อมกับเครื่องคอมพิวเตอร์นั้นได้แก่ MS DOS. นอกจากนั้นเขาสั่งซื้อระบบปฏิบัติการมินิกซ์ต่างหากเพื่อมาติดตั้งในคอมพิวเตอร์เครื่องใหม่ของเขาแทน MS DOS สำหรับศึกษาการทำงานของระบบปฏิบัติการ. เพราะว่ามินิกซ์เป็นระบบปฏิบัติเพื่อการศึกษาไม่ใช่เพื่อการใช้งาน, ระบบบางอย่างในมินิกซ์ใช้งานได้ไม่ดีนักและสิ่งที่เขาไม่ชอบคือ*เทอร์มินอลเอมูเลเตอร์ (terminal emulator)* ของมินิกซ์. เขาจึงตัดสินใจเริ่มสร้างโปรแกรมเทอร์มินอลเอมูเลเตอร์ด้วย*ภาษาแอสเซมบลี (assembly language)* เอง.

terminal emulator ▶ *เทอร์มินอลเอมูเลเตอร์.* โปรแกรมที่ใช้ในการแสดงผลในรูปของตัวอักษรผ่านทางหน้าจอโดยการป้อนข้อมูลเข้าผ่านทางคีย์บอร์ด

เพื่อที่จะอ่านนิวส์กรุปของมหาวิทยาลัยผ่าน*โมเด็ม (Modem)* จากที่บ้าน, เขาต้องสร้างเทอร์มินัลเอมูเลเตอร์ที่ทำงาน 2 อย่างพร้อม ๆกันคือ อ่านข้อมูลจากโมเด็มแล้วแสดงผลทางหน้าจอ, และรับข้อมูลจากคีย์บอร์ดแล้วส่งต่อให้โมเด็ม. การทำงานสองอย่างในเวลาเดียวกันใช้หลักการที่เรียกว่า *task switching* ซึ่งเป็นพื้นฐานของระบบปฏิบัติการ. โปรแกรมเทอร์มินอลเอมูเลเตอร์ที่เขาสร้างเก็บบันทึกอยู่ในแผ่น*ฟล็อปปี้ดิกส์ (floppy disk)* เพราะฉะนั้นเวลาที่เขาจะใช้โปรแกรมต้อง*บูต (boot)* โปรแกรมจากแผ่นฟล็อปปี้ดิกส์โดยตรง. ปัญหาที่ต้องแก้มีมากขึ้นเมื่อเขาต้องการดาวน์โหลดไฟล์ผ่านทางโมเด็มเก็บบันทึกลงฮาร์ดดิกส์. ปัญหานี้ทำให้เขาต้องศึกษาเกี่ยวกับ *file system* และเขาไม่ท้อถอย. Linus เขียน*ไดร์เวอร์ (driver)* ของฮาร์ดดิกส์และทำให้เทอร์มินอลเอมูเลเตอร์เก็บข้อมูลลงฮาร์ดดิกส์จนได้. ในที่สุดเทอร์มินอลเอมูเลเตอร์ที่เป็นโปรแกรมเล็ก ๆ, เดิมเขียนขึ้นเพื่ออ่านนิวส์กรุปก็เริ่มเปลี่ยนเป็นระบบปฏิบัติการทีละเล็กละน้อย.

ระบบปฏิบัติการเริ่มเป็นรูปเป็นร่างชัดเจนเมื่อ Linus ทำการ*พอร์ต (port)* *เชลล์ (shell)* ซึ่งเป็นตัวโปรแกรมรับคำสั่งจากคีย์บอร์ดส่งต่อให้ระบบปฏิบัติการ. ระบบปฏิบัติเริ่มสมบูรณ์เมื่อเขาพอร์ต*คอมไพเลอร์ภาษา C (C compiler)* สำเร็จ. นั่นหมายความว่าเขาสามารถนำรหัสต้นฉบับของโปรแกรมอื่นมาคอมไพล์และใช้ได้กับระบบปฏิบัติการที่เขาสร้างไว้. ความต้องการ, ความพยายาม, และการไม่ยอมแพ้ของนาย Linus นี่เองที่ทำให้เกิดระบบปฏิบัติการที่เรียกว่า “ลินุกซ์” ในวันนี้.

port ▶ *พอร์ต*(กริยา). การดัดแปลงต้นฉบับโปรแกรม, คอมไพล์ให้ใช้บนระบบปฏิบัติการที่แตกต่างกันหรือสถาปัตยกรรมคอมพิวเตอร์ที่แตกต่างกันได้

หลังจากที่ Linus เปิดเผยระบบปฏิบัติการที่เขาสร้างผ่านทางอินเทอร์เน็ต. นักพัฒนาซอฟต์แวร์จากทั่วโลกที่สนใจเริ่มพอร์ตโปรแกรมต่าง ๆที่ใช้ในยูนิกซ์ให้ใช้ได้กับลินุกซ์เช่น *ระบบ X วินโดว์ (X window system)*, ยูทิลิตี้โปรแกรมต่าง ๆที่จาก Free Software Foundation (GNU) และอื่น ๆ. นอกจากการพอร์ตโปรแกรมที่มีอยู่แล้ว, โปรแกรมบางอย่างสร้างขึ้นสำหรับใช้บน ลินุกซ์ปัจจุบันพอร์ตไปใช้ในระบบปฏิบัติการยูนิกซ์ด้วยเช่น

X window system ▶ *X วินโดว์.* ระบบการแสดงผลกราฟฟิกส์ผ่านทางจอภาพในรูปแบบของหน้าต่างหลายบาน. เป็นโครงการของ MIT ที่พัฒนาต่อมาจาก W windows system ของ Stanford. ระบบ X วินโดว์ที่ใช้ในลินุกซ์เป็นโครงการของ Xfree86 ซึ่งพัฒนา X เซิฟร์เวอร์สำหรับวีดีโอการ์ดต่าง ๆ. สังเกตุว่าเขียนว่า X window ไม่ใช่ X windows.

GNOME เป็นต้น. สิ่งที่สำคัญอีกอย่างคือการพอร์ตตัวระบบปฏิบัติการลินุกซ์เองให้ใช้ได้กับ*สถาปัตยกรรมคอมพิวเตอร์ (computer architecture)* อื่น ๆ เช่น Atari, Alpha, SPARC เป็นต้น.

1.3 กฎหมายกับซอฟต์แวร์

เนื่องจากผู้เขียนไม่ใช่ผู้เชี่ยวชาญด้านกฎหมาย, เนื้อหาที่เกี่ยวกับลิขสิทธิ์, ใบอนุญาต และสิทธิบัตรที่แนะนำนี้อาจจะไม่ใช่ข้อมูลที่เที่ยงตรง. สำหรับผู้ที่สนใจกรุณาหาหนังสือหรือเว็บไซต์ [3] ที่เกี่ยวกับกฎหมายอ่านประกอบ.

หนังสือเล่มนี้เป็นหนังสือเกี่ยวกับลินุกซ์ไม่ใช่หนังสือที่เกี่ยวกับกฎหมาย. แต่หลีกเลี่ยงไม่ได้ที่จะต้องกล่าวถึงเพราะซอฟต์แวร์ไม่ว่าจะเป็นลินุกซ์เคอร์เนลหรือโปรแกรมต่าง ๆที่ผู้ใช้ใช้อยู่ถึงแม้ว่าจะ “ฟรี” แต่ไม่ได้หมายความว่าเราจะทำอะไรก็ได้กับซอฟต์แวร์เหล่านั้น.

1.3.1 ลิขสิทธิ์

ลิขสิทธิ์ (copyright) คือสิทธิ์ที่ผู้สร้างสรรค์พึงจะได้จากผลงานที่สร้าง. ผลงานในที่นี้ได้แก่ ซอฟต์แวร์, หนังสือ, ภาพยนต์, เพลง เป็นต้น. ผู้ถือสิทธิ์มีสิทธิ์ในผลงานของตัวเองเช่น สิทธิ์ในการขาย, สิทธิ์ห้ามทำสำเนา, สิทธิ์ในการแจกจ่าย ฯลฯ. ซอฟต์แวร์ลิขสิทธิ์คือซอฟต์แวร์ที่ได้รับการคุ้มครองโดยกฎหมายลิขสิทธิ์. การที่ผู้อื่นที่ไม่ใช่เจ้าของผลงานนั้น ๆจะใช้ต้องได้รับอนุญาตจากผู้ถือสิทธิ์ก่อนที่จะใช้ผลงานนั้น. ซอฟต์แวร์ที่ไม่มีลิขสิทธิ์ได้แก่ซอฟต์แวร์ที่ไม่มีการคุ้มครองใด ๆทางกฎหมาย เช่นซอฟต์แวร์ที่ประกาศเป็น*สาธารณะสมบัติ (public domain)*.

ตัวอย่างซอฟต์แวร์เช่นลินุกซ์เคอร์เนลเป็นซอฟต์แวร์ที่มีลิขสิทธิ์และผู้ที่ถือครองลิขสิทธิ์ได้แก่นาย Linus Torvalds ซึ่งเป็นผู้สร้าง. ตามหลักแล้วไม่ว่าจะเป็นใครก็ตามที่ต้องการใช้ลินุกซ์ต้องได้รับอนุญาตจากนาย Linus ก่อนจึงจะใช้ได้. แต่ในความเป็นจริงผู้ใช้สามารถใช้ลินุกซ์ได้โดยไม่ต้องขออนุญาตเพราะนาย Linus ได้ให้อนุญาตแล้วโดยใช้*ใบอนุญาต (license)* ที่เรียกว่า GNU General Public License ต่อลินุกซ์เคอร์เนลที่เขาสร้างขึ้น.

1.3.2 ใบอนุญาต

ตั้งแต่เริ่มต้นเรื่องราวของคอมพิวเตอร์, ซอฟต์แวร์คือสินค้าแต่เป็นสินค้าที่ต่างจากสินค้าทั่วไปตรงที่ซอฟต์แวร์เป็นสินค้าเชิงนามธรรมมากกว่ารูปธรรม. ตามธรรมชาติของซอฟต์แวร์, ซอฟต์แวร์เป็นข้อมูลที่เก็บอยู่ในสื่อกลางต่าง ๆได้. สามารถทำสำเนาส่งต่อให้คนอื่นได้. ใช้ได้กับคอมพิวเตอร์ไม่เจาะจงว่าต้องเป็นเครื่องใด ๆ. ธรรมชาติของซอฟต์แวร์นี้เองเป็นปัญหาสำหรับผู้ผลิตซอฟต์แวร์เชิงพาณิชย์. กล่าวคือถ้าผู้ผลิตซอฟต์แวร์ไม่ทำสัญญากับผู้ที่ได้ซอฟต์แวร์, ผู้ที่ได้ซอฟต์แวร์นั้นสามารถทำสำเนาแจกจ่าย, หรือใช้กับคอมพิวเตอร์ได้ไม่เลือก. เป็นผลให้ผู้ผลิตซอฟต์แวร์ไม่สามารถดำเนินธุรกิจด้วยการขายซอฟต์แวร์ได้. นี่เองเป็นที่มาของ*ใบอนุญาต (license)*.

โดยปรกติแล้วซอฟต์แวร์จะมีใบอนุญาตในการใช้งานกำกับมาด้วย. ใบอนุญาตได้แก่ข้อตกลงระหว่างผู้สร้าง, เจ้าของ, หรือผู้จำหน่ายซอฟต์แวร์กับผู้ที่ได้รับซอฟต์แวร์นั้น ๆ ซึ่งเนื้อหาของข้อตกลงจะแตกต่างกันออกไปตามกรณี. โดยปรกติ, ในอนุญาตจะจำกัดสิทธิ์ต่าง ๆที่พึงกระทำได้กับซอฟต์แวร์เช่นห้ามทำสำเนาแจก, ห้ามติดตั้งซอฟต์แวร์ลงใน

เครื่องคอมพิวเตอร์เกินหนึ่งเครื่องเป็นต้น. แน่นอนว่าในทางปฏิบัติ, ผู้ที่ได้ซอฟต์แวร์นั้นมาสามารถกระทำสิ่งเหล่านี้ได้แต่จะเป็นการละเมิดข้อตกลงและถือว่ามีความผิด, อาจถูกดำเนินคดีตามกฎหมายต่อไป.

GNU General Public License

ลินุกซ์เคอร์เนลให้อิสระแก่ผู้ใช้ครอบคลุมเรื่องการทำสำเนา (copying), การแจกจ่าย (distribution) และการแก้ไข (modification). ใบอนุญาตที่ลินุกซ์เลือกใช้ได้แก่ *GNU General Public License (GNU GPL)*. ใบอนุญาตแบบ GPL นี้สร้างขึ้นโดยองค์กรที่เรียกว่า *Free Software Foundation* ก่อตั้งและดำเนินการโดยนาย Richard Stallman.

Free Software Foundation เป็นองค์กรอิสระไม่หวังผลกำไรซึ่งก่อตั้งโดย Richard Stallman เมื่อปีค.ศ. 1982. จุดประสงค์ขององค์กรนี้คือสร้างระบบยูนิกซ์ที่เสรี. ซอฟต์แวร์ที่มีชื่อเสียงที่พัฒนาโดยองค์กรนี้ได้แก่ GNU C compiler, GNU Emacs ฯลฯ. Free Software Foundation มีชื่อเรียกทั่วไปว่า GNU เป็นคำย่อแบบ recursive ของคำว่า "GNU is Not Unix"

ใบอนุญาตแบบ GPL นี้เองที่ทำให้ลินุกซ์ต่างจากระบบปฏิบัติการทั่ว ๆไป. โดยปรกติแล้วระบบปฏิบัติการจะพัฒนาโดยองค์กรหรือบริษัทในกลุ่มเล็ก ๆ. ไม่มีการเปิดเผยรหัสต้นฉบับแก่สาธารณะ, ผู้ใช้ไม่สามารถทำสำเนาหรือแจกจ่ายให้คนอื่นต่อได้. ระบบปฏิบัติการลินุกซ์มีความคิดที่แตกต่างกับสิ่งที่กล่าวมาอย่างสิ้นเชิง. ลินุกซ์เป็นระบบปฏิบัติการที่ฟรี. ผู้ที่ต้องการใช้ลินุกซ์ไม่มีความจำเป็นที่ต้องเสียเงินซื้อ, สามารถ*ดาวน์โหลด (download)* จากอินเทอร์เน็ต [4]. และผู้ใช้มีสิทธิ์ที่จะแจกจ่ายต่อไปได้โดยไม่ผิดกฎหมายตราบที่ผู้ใช้ยังปฏิบัติตามใบอนุญาต GPL ระบุไว้. คำว่า "ฟรี" ในที่นี้ไม่ได้หมายความว่าราคาเป็น "ศูนย์" แต่หมายถึงความเป็น "อิสระเสรี" ที่ผู้ใช้กระทำได้. การที่ลินุกซ์ใช้ใบอนุญาตแบบ GPL มีผลดีคือให้อิสระเสรีแก่ผู้ใช้. เปิดโอกาสให้นักพัฒนาซอฟต์แวร์สามารถศึกษารหัสต้นฉบับ, และพัฒนาปรับปรุงแก้ไขโปรแกรมต่อไปให้ดียิ่งขึ้น. GPL ยังระบุถึง*งานสืบทอด (derived work)* ของซอฟต์แวร์ที่มีใบอนุญาตเป็นแบบ GPL ด้วยว่างานสืบทอดจะใช้ใบอนุญาตแบบ GPL โดยปริยาย. กล่าวคือถ้าเป็นงานที่สืบทอดมาจากงาน GPL งานนั้นต้องเป็น GPL ด้วย. บุคคลใด ๆมีอิสระทำสำเนา, แจกจ่าย, แก้ไขรหัสต้นฉบับของงานสืบทอดได้. เราสามารถกล่าวได้ว่าเหตุที่ลินุกซ์ใช้กันอย่างแพร่หลายและมีการแก้ไขปรับปรุงให้ทันสมัยทันเหตุการณ์เป็นผลของการที่ลินุกซ์ใช้ใบอนุญาต GPL นี่เอง.

GNU Lesser General Public License

เดิมคือใบอนุญาต GNU Library General Public License.

สำหรับงานสืบทอดที่ทำต่อหรือใช้รหัสต้นฉบับของซอฟต์แวร์ที่มีใบอนุญาตแบบ GPL, ตัวใบอนุญาตระบุไว้ว่างานสืบทอดจะต้องใช้ใบอนุญาตแบบ GPL ไปโดยปริยาย [5]. หมายความว่าโปรแกรมเชิงพาณิชย์ที่สืบทอดหรือใช้รหัสต้นฉบับจากซอฟต์แวร์ที่มีใบอนุญาตแบบ GPL ต้องมีใบอนุญาตเป็นแบบ GPL ไปด้วย. ในกรณีอาจเป็นผลเสียในเชิงพาณิชย์เพราะซอฟต์แวร์ที่ผลิตออกมาจะกลายเป็นซอฟต์แวร์ที่สามารถทำสำเนา, แจกจ่ายได้, ขอรหัสต้นฉบับได้ไปโดยปริยาย. เหตุนี้เองทาง FSF จึงออกในอนุญาตที่เรียกว่า *GNU Lesser General Public License (LGPL)* ที่ลดหย่อนสิทธิ์ของผู้ใช้เล็กน้อยตรงที่เปิดโอกาสให้ซอฟต์แวร์เชิงพาณิชย์สามารถใช้รหัสต้นฉบับของซอฟต์แวร์ซึ่งรวมไปถึง*ไลบรารี (library)* ที่มีใบอนุญาตแบบ GPL ได้โดยที่งานสืบทอดนั้นยังคงใช้ใบอนุญาตของตัวเองที่ไม่ใช่ GPL ก็ได้.

สรุปได้ว่า LGPL ให้อิสระต่อผู้ใช้น้อยลง (lesser) เมื่อเทียบกับใบอนุญาตแบบ GPL. ในทางตรงกันข้ามเป็นการเปิดโอการให้ซอฟต์แวร์เชิงพาณิชย์ที่ถือเป็นงานสืบทอดใช้

ตารางที่ 1.1: ตัวอย่างประเภทของใบอนุญาตต่าง ๆแบ่งตามความเสรี

ประเภทของใบอนุญาต	ตัวอย่าง
GPL-Compatible Free Software Licenses	GPL, LGPL, Public Domain, X11 License, BSD (modified)
GPL-Incompatible, Free Software Licenses	BSD (original), Open Software License, Apache License, Mozilla Public License (MPL), Q Public License (QPL), PHP License
Non-Free Software Licenses	Open Public License, Aladdin Free Public License, Microsoft's Shared Source License

ซอฟต์แวร์หรือไลบรารีที่มีใบอนุญาตเป็น GPL ได้โดยยังคงใบอนุญาตของตัวเอง.

การตีความงานสืบทอดว่าคืออะไรยังไม่ค่อยกระจ่างมากนัก, คงต้องยกเป็นเรื่องของกฎหมาย. ตัวอย่างงานสืบทอดที่ชัดเจนได้แก่การแก้ไข, เพิ่มเติม, ปรับแต่งงานที่เป็นแบบ GPL ถือว่าเป็นงานสืบทอดอย่างเห็นได้ชัด. ซอฟต์แวร์ที่ใช้ไลบรารีที่ไม่ได้สร้างเองก็ถือว่าเป็นงานสืบทอดเช่นกัน. ดังนั้นไลบรารีที่เป็นมาตรฐานและใช้กันมากของ FSF มันจะใช้ใบอนุญาตแบบ LGPL เช่นไลบรารี C ของ GNU เป็นต้น.

การคอมไพล์โปรแกรมเช่นโปรแกรมที่เขียนด้วยภาษา C จะใช้ไลบรารี C โดยปริยาย. แต่เนื่องจากไลบรารี C ของ GNU มีใบอนุญาตเป็น LGPL เพราะฉะนั้นโปรแกรมที่ใช้ไลบรารีนี้ไม่จำเป็นต้องเป็น GPL ก็ได้.

ใบอนุญาตแบบอื่น ๆ

ในขณะที่ลินุกซ์เคอร์เนลใช้ใบอนุญาตแบบ GPL, โปรแกรมต่าง ๆที่นำมารวมเข้าเป็นระบบใช้งานไม่จำเป็นต้องใช้ใบอนุญาตแบบ GPL ก็ได้. โปรแกรมที่ใช้กับลินุกซ์เคอร์อาจจะใช้ใบอนุญาตแบบอื่น ๆ [6] ตามตัวอย่างที่แสดงในตารางที่ 1.1.

1.3.3 สิทธิบัตร

สิทธิบัตร (patent) คือหนังสือสำคัญที่ออกให้เพื่อคุ้มครองการประดิษฐ์ หรือการออกแบบผลิตภัณฑ์. การประดิษฐ์นี้รวมถึงกรรมวิธีซึ่งได้แก่ขั้นตอนวิธี (algorithm) ด้วย. หมายความว่าถ้ามีใครไปจดสิทธิบัตรขั้นตอนวิธีใด ๆก่อนแล้วผู้พัฒนาซอฟต์แวร์ใช้ขั้นตอนเดียวกันในการสร้างซอฟต์แวร์ไม่ว่าจะคิดเองหรือไม่ต้องขออนุญาตหรือจ่ายค่าตอบแทนกับผู้ที่จดสิทธิบัตรนั้นก่อน.

ปัญหาของสิทธิบัตรกับการพัฒนาซอฟต์แวร์เสรีมีหลายกรณี. สมมติว่าบริษัทบางแห่งจดสิทธิบัตรไว้และนักพัฒนาซอฟต์แวร์เสรีสร้างซอฟต์แวร์อันหนึ่งซึ่งไปตรงกันเนื้อหาของสิทธิบัตรนั้น. บริษัทที่เป็นเจ้าของสิทธิบัตรอาจจะรู้ว่าซอฟต์แวร์ที่นักพัฒนาซอฟต์แวร์นั้นสร้างใช้วิธีการเดียวกับที่บริษัทจดสิทธิบัตรไว้ แต่บริษัทไม่ฟ้องร้องทันที. บริษัทอาจจะเก็บเรื่องเงียบและถ้าซอฟต์แวร์ที่พาดพิงเนื้อหาสิทธิบัตรของตัวเองเกิดได้รับความนิยมมีคนใช้มาก, บริษัทค่อยเริ่มฟ้องร้องหรือเรียกเก็บค่าตอบแทนจากผู้ใช้หรือผู้สร้างซอฟต์แวร์นั้น ๆก็ได้. วิธีนี้เองเป็นวิธีทำรายได้อย่างหนึ่งซึ่งไม่เป็นธรรมเท่าไรนักแต่ก็ยังมีคนทำโดย

เฉพาะในสหรัฐอเมริกาเพราะสำนักงานจดสิทธิบัตรได้รายได้จากการจดสิทธิบัตร. ถ้ามีคนจดสิทธิบัตรมากเท่าไรก็ได้รายได้มากเท่านั้น.

ปัญหาเกี่ยวกับสิทธิบัตรอีกอย่างได้แก่การจดสิทธิบัตรกรรมวิธีในการกระทำอย่างใดอย่างหนึ่งที่ใคร ๆก็คิดได้. ตัวอย่างเช่น Amazon.com ร้านขายหนังสือบนอินเทอร์เน็ตรายใหญ่ได้จดสิทธิบัตรเกี่ยวกับขั้นตอนการซื้อของบนอินเทอร์เน็ตโดยกดปุ่มหนึ่งที (one-click purchasing) [7] ซึ่งขั้นตอนนี้เป็นสิ่งที่จำเป็นในการซื้อขายของการอินเทอร์เน็ตและใคร ๆก็คิดได้.

1.4 ซอฟต์แวร์เสรี

ต้องทำความเข้าใจอีกเล็กน้อยว่า GPL คือใบอนุญาตไม่ใช่ซอฟต์แวร์. *free software* [8, 9] หรือภาษาไทยใช้คำว่า*ซอฟต์แวร์เสรี*, ฟรีในที่นี้คืออิสระเสรีไม่เกี่ยวกับราคา. กล่าวคือผู้ใช้มีอิสระที่จะกระทำการ (run), ทำสำเนา (copy), แจกจ่าย (distribute), ศึกษา, ปรับปรุงเปลี่ยนแปลงซอฟต์แวร์. แบ่งความเป็นอิสระได้หลายระดับคือ:

คำแปรภาษาไทยที่สื่อความหมายคำว่า free sotfware ได้แก่คำว่า*ซอฟต์แวร์เสรี*. ไม่ควรใช้ทับศัพท์ว่าฟรีซอฟต์แวร์เพราะเกิดความสับสนได้ง่าย. เสรีในที่นี้ยังหมายถึงเสรีที่มีขอบเขต. ไม่ใช่เสรีที่อยากจะทำอะไรก็ทำได้.

- อิสระระดับที่ 0: อิสระที่ผู้ใช้สามารถใช้ซอฟต์แวร์ไม่ว่าในกรณีหรือโอกาสใด ๆ
- อิสระระดับที่ 1: อิสระในการศึกษาการทำงานของซอฟต์แวร์, เปลี่ยนแปลงแก้ไขซอฟต์แวร์ตามที่ต้องการ. หมายความว่ารหัสต้นฉบับเปิดเผยต่อผู้ที่ต้องการ.
- อิสระระดับที่ 2: อิสระในการแจกจ่ายซอฟต์แวร์ให้ผู้อื่น ๆ.
- อิสระระดับที่ 3: อิสระในการปรับปรุงซอฟต์แวร์ให้ดีขึ้น, แจกจ่ายสู่สาธารณะชนได้เพื่อให้ได้รับประโยชน์ของการปรับปรุงร่วมกัน. การที่จะบรรลุจุดประสงค์นี้, รหัสต้นฉบับต้องเปิดเผยด้วย.

ซอฟต์แวร์ที่จะเรียกได้ว่าเป็นซอฟต์แวร์เสรี (free software) ต้องมีองค์ประกอบข้างต้นทั้งหมดครบ. อีกประการหนึ่งคือ free software ไม่ได้หมายความว่าไม่มีราคา. อาจจะมีการขายเป็นสินค้าก็ได้และเป็นซอฟต์แวร์เสรี (free software) ได้ถ้ามีคุณสมบัติทั้งหมดที่กล่าวไปแล้ว.

1.5 โอเพนซอร์ส

ถึงแม้ว่าได้มีการกำหนดความหมายของคำว่า free software ไปแล้วก็ตาม, คำว่า “ฟรี” ในภาษาอังกฤษมักเป็นสาเหตุให้บุคคลทั่วไปสับสนได้ง่าย. ตัวอย่างเช่นโปรแกรมบางอย่างแจกฟรีแต่ผู้ใช้ไม่สามารถดูหรือศึกษารหัสต้นฉบับได้. บางโปรแกรมเรียกกันว่า “ฟรี” เพราะเป็น*สาธารณะสมบัติ (pulic domain)*. Public domain หมายถึงการไม่มี*ลิขสิทธิ์ (copyright)* คุ้มครอง, ผู้ใช้สามารถทำอะไรก็ได้ซึ่งบางกรณีไม่ใช่ผลดีเสมอไป. บางโปรแกรม “ฟรี” แต่ฟรีเฉพาะบุคคลบางกลุ่มหรือการใช้ที่ไม่ใช่การพาณิชย์. ความสับสนต่าง ๆเหล่านี้ทำให้เกิดการใช้คำว่า*โอเพนซอร์ส (open source)* แทนคำว่า free software.

ซอฟต์แวร์ที่จะเรียกได้ว่าเป็นโอเพนซอร์สนั้นต้องประกอบด้วยข้อกำหนดหลายอย่างที่กำหนดโดย Open Source Initiative [10]. ตัวอย่างเช่นซอฟต์แวร์โอเพนซอร์สต้องแจกจ่ายได้เสรี, ไม่กีดกันบุคคลใดบุคคลหนึ่งไม่ว่าจะเป็นเชื้อชาติหรืออื่นๆ. สามารถเอาไปใช้ในงานใดก็ได้โดยไม่กีดกัน. เปิดเผยรหัสต้นฉบับเป็นต้น. ผู้อ่านสามารถอ่านคำจัดกัดความของโอเพนซอร์สโดยละเอียดเพิ่มเติมได้จากเว็บไซต์ของ Open Source Initiative.

Open Source Initiati
องค์กรณ์ที่ไม่หวังผล
ประสงค์ช่วยส่งเสริม
ซอฟต์แวร์แบบโอเพ
ออกใบรับรองซอฟต์
ให้รางวัล เป็นต้น.

1.6 ธรรมชาติและคุณลักษณะของลินุกซ์

เปิดเผยและให้อิสระแก่ผู้ใช้

เนื่องจากลินุกซ์ใช้ใบอนุญาตแบบ GPL ผู้ใช้จึงมีสิทธิ์ที่จะทำสำเนา, ศึกษารหัสต้นฉบับ, แก้ไขรหัสต้นฉบับตามที่ต้องการตราบเท่าที่รหัสต้นฉบับที่เปลี่ยนแปลงเปิดเผยต่อสาธารณะ. ลินุกซ์สามารถดาว์นโหลดได้จากอินเทอร์เน็ต, หรือก๊อบปี้แจกจ่ายซื้อขายได้ในรูปแบบของสื่อต่างๆเช่น CDROM เป็นต้น. ในกรณีที่ไม่สะดวกดาว์นโหลดจากอินเทอร์เน็ต, ผู้ใช้สามารถเลือกซื้อ CDROM ดิสทริบิวชันที่ทำสำเร็จแล้วจากผู้ผลิตดิสทริบิวชันโดยตรง.

พึ่งตนเองก่อนขอความช่วยเหลือจากผู้อื่น

การแก้ปัญหาหรือข้อสงสัยที่เกิดจากลินุกซ์โดยพื้นฐานแล้วผู้ใช้ต้องพึ่งตนเองเป็นหลัก. เนื่องจากลินุกซ์สร้างจากกลุ่มคนบนอินเทอร์เน็ตไม่ใช่สินค้าที่สร้างพัฒนาโดยบริษัทใดบริษัทหนึ่ง, จึงเป็นการยากที่จะรับประกันหรือสนับสนุนการใช้งาน (support) ได้อย่างเต็มที่. ผู้ใช้ควรจะตระหนักอยู่เสมอว่าต้องพึ่งตนเองก่อนเวลาเจอปัญหาหรือข้อสงสัยในการใช้งาน. ผู้ใช้อาจจะต้องหาข้อมูลจากหนังสือหรืออินเทอร์เน็ต, คู่มือการใช้งานเอง. ถ้าไม่สามารถแก้ปัญหาได้หรือหาคำตอบไม่เจอก็อาจจะถามผู้รู้หรือผู้ใช้งานคนอื่นทาง mailing list, webboard, newsgroup ฯลฯ ประกอบกับข้อมูลที่ตนเองหาไว้.

ประสิทธิภาพ

ลินุกซ์เป็นระบบปฏิบัติการที่*เสถียร (stable)* และมี *ดาว์นไทม์ (downtime)* ต่ำ. รหัสต้นฉบับของระบบปฏิบัติการได้รับการตรวจสอบและทดลองอย่างดีก่อนที่จะ*รีลีส (release)* ในแต่ละเวอร์ชั่น. แน่นอนว่าไม่มีอะไรที่สมบูรณ์แบบ, เคอร์เนลหรือโปรแกรมใช้งานบางอย่างอาจจะมีช่องโหว่หลังจากออกประกาศใช้. โดยทั่วไปช่องโหว่หรือข้อผิดพลาดเหล่านี้จะถูกรายงานต่อผู้พัฒนาและแก้ไขในรุ่นถัดไป. ถ้าเป็นช่องโหว่ที่เกี่ยวกับความปลอดภัยของระบบอย่างร้ายแรงก็อาจจะได้รับการแก้ไขอย่างรวดเร็วแล้วแต่กรณี.

ใช้งานได้หลายระดับ

ลินุกซ์สามารถใช้งานได้กับคอมพิวเตอร์หลายระดับตั้งแต่งานเดกสท็อปที่เน้น GUI, จนถึงระดับเซิฟเวอร์ที่เน้นความเสถียรและความสะดวกในการปรับแต่งทาง *CUI (Command Line User Interface)*. ลินุกซ์ไม่เพียงแค่ใช้ได้กับคอมพิวเตอร์ส่วน

บุคคลเท่านั้น, ลินุกซ์ยังสามารถใช้ใน*ระบบเอ็มเบ็ด (embedded system)* ซึ่งหน่วยความจำมีขนาดเล็กและจำกัด.

การใช้งานร่วมกับระบบปฏิบัติการอื่น ๆ

ลินุกซ์สามารถอ่านข้อมูลที่บันทึกในดิกส์โดยระบบปฏิบัติการเช่น MS-DOS, Microsoft Windows, SVR4, OS/2, Mac, Solaris ได้. ทางด้านเน็ตเวอร์ก, ลินุกซ์สนับสนุน*เน็ตเวิร์กเลเยอร์ (network layer)* หลายประเภทเช่น *อีเทอร์เน็ต (Ethernet)*, FDDI, Token ring ฯลฯ. สามารถรัน*โปรแกรมไบนารี (Binary Program)* ของ DOS หรือ MicroSoft Windows ได้โดยผ่านเอมูเลเตอร์เช่น VMWare.

ยึดมาตรฐาน

ลินุกซ์เป็นระบบปฏิบัติการที่ร่วมสร้างโดยคนหลายบนอินเทอร์เน็ต, เพราะฉะนั้นมาตรฐานต่าง ๆเป็นสิ่งจำเป็นเพื่อให้การพัฒนาระบบดำเนินไปในแนวเดียวกัน. มาตรฐานดังกล่าวเป็นที่เปิดเผยและยอมรับกันทั่วไปเช่น POSIX, *เน็ตเวอร์กโปรโตคอล (network protocol)* ต่าง ๆเป็นต้น.

protocol ►
โปรโตคอล. คือข้อตกลง, วิธีการในการกระทำการอย่างใดอย่างหนึ่ง. เช่น HTTP protol เป็นข้อตกลง, วิธี

1.7 ดิสทริบิวชัน

จากที่กล่าวไปแล้วข้างต้นว่าลินุกซ์ในความหมายทั่วไปหมายถึงการนำเคอร์เนลและโปรแกรมต่าง ๆมารวมกันเป็นระบบ. การนำเคอร์เนลและโปรแกรมต่าง ๆมารวมกันแล้วแจกจ่าย (distribute) เพื่อความสะดวกของผู้ใช้นี้เรียกว่า*ดิสทริบิวชัน (distribution)*. ดิสทริบิวชันบางค่ายก็เป็นเชิงพาณิชย์, บางค่ายก็สร้างโดยอาสาสมัครโดยไม่หวังผลประโยชน์. ถึงแม้บางค่ายจะเป็นดิสทริบิวชันเชิงพาณิชย์ก็ตาม, โดยส่วนใหญ่แล้วผู้ใช้สามารถดาว์นโหลดได้โดยไม่เสียค่าใช้จ่ายให้ดิสทริบิวชันเหล่านั้น.

ดิสทริบิวชั่นแต่ละค่ายมีความแตกต่างกันในรายละเอียด เช่นรูปร่างหน้าตาเดกสท็อป, ความยากง่ายของ*โปรแกรมติดตั้ง (installer)*, *ระบบจัดการแพ็กเกจ (package management)*, เพิ่มเติมซอฟต์แวร์เชิงพาณิชย์แถมเป็นต้น. แต่สิ่งที่ทุกค่ายเหมือนกันคือหัวใจของดิสทริบิวชันใช้ลินุกซ์เคอร์เนลเป็นระบบปฏิบัติพื้นฐาน, รวบรวมซอฟต์แวร์เสรีต่าง ๆให้เป็นระบบใช้งาน.

หลังจากที่นาย Linus ประกาศระบบปฏิบัติการที่เขาสร้างขึ้นผ่านทางนิวส์กรุปแล้ว, ในราวปี ค.ศ.1993 Soft Landing Software (SLS) โดยนาย Peter McDonald ก็เริ่มสร้างดิสทริบิวชันเป็นเจ้าแรก ๆ. ดิสทริบิวชันอื่นในช่วงต้น ๆค่ายอื่นได้แก่ Yggdrasil ซึ่งทั้ง SLS และ Yggdrasil ไม่เป็นที่นิยมในปัจจุบัน. ดิสทริบิวชันที่เรียกว่าประสบความสำเร็จและเป็นที่รู้จักกันอย่างแพร่หลายในปัจจุบันได้แก่ Slackware, Red Hat, Debian เป็นต้น.

จากประวัติศาสตร์ที่ผู้เขียนค้นคว้ามาไม่สามารถระบุได้แน่ชัดว่าดิสทริบิวชันค่ายแรกเริ่มเมื่อใดและใครเป็นผู้สร้าง

1.7.1 Slackware

SLS เป็นดิสทริบิวชันค่ายแรก ๆแต่ก็ยังเป็นเรื่องยากสำหรับผู้ใช้ทั่วไปที่ติดตั้งและเริ่มต้นใช้ลินุกซ์. นาย Patrick Volkerding เป็นผู้ที่แก้จุดอ่อนเหล่านี้และสร้างดิสทริบิวชันที่มีชื่อว่า *Slackware* ขึ้นมาในราวเดือนมิถุนายน ค.ศ.1993. ในเวลาต่อมาดิสทริบิวชัน

นี้ก็กลายเป็นต้นแบบดิสทริบิวชั่นของค่ายอื่น ๆต่อมา. Slackware เน้นความเรียบง่ายของระบบและคำนึงถึงการใช้งาน. โปรแกรมต่าง ๆที่รวมอยู่ในดิสทริบิวชั่นนี้แพ็กอย่างง่าย ๆ ด้วยโปรแกรม `tar` และ `gzip`.

1.7.2 Red Hat

ในราวปี ค.ศ.1994 [11], Red Hat เป็นดิสทริบิวชันเชิงพาณิชย์แรก ๆที่โปรแกรมติดตั้งใช้ง่ายและเป็น *Graphical User Interface (GUI)*. ด้วยเหตุนี้เองที่ทำให้ Red Hat เป็นที่นิยมกันอย่างแพร่หลาย. Red Hat มีจุดเด่นที่*การจัดการแพ็กเกจ (Package Management)*, โปรแกรมติดตั้งและ*โปรแกรมช่วยควบคุมระบบ (System Management Utilities)*.

Red Hat ใช้โปรแกรมจัดการแพ็กเกจที่เรียกว่า *RPM (Red Hat Package Management)*. ผู้ใช้สามารถติดตั้งโปรแกรมที่มาในรูปของไฟล์ `.rpm`. โปรแกรม `rpm` จะจัดการติดตั้ง*ไบนารีไฟล์ (binary file)*, ไฟล์ที่เกี่ยวข้องกับตัวโปรแกรม, คู่มือการใช้งานที่อยู่ในไฟล์ `.rpm` ไปไว้ในไดเรกทอรี่ที่เหมาะสม. หลักจากที่ติดตั้งไฟล์ต่าง ๆแล้ว `rpm` จะ*ปรับแต่ง (configure)* ตัวโปรแกรมให้ให้เข้ากับระบบคอมพิวเตอร์และบันทึกข้อมูลที่เกี่ยวกับโปรแกรมนั้น ๆใน*ฐานข้อมูลการจัดการแพ็กเกจ (Package Management Database)*. ผู้ใช้สามารถ*อันอินสตอลล์ (uninstall)* โปรแกรมที่ไม่ต้องการออกจากระบบได้อย่างง่ายดายด้วยโปรแกรม `rpm` เช่นกัน. ระบบจัดการแพ็กเกจจะลบโปรแกรมและไฟล์ต่าง ๆที่เกี่ยวข้องให้โดยอัตโนมัติ. โปรแกรมติดตั้งที่ใช้เป็น GUI ซึ่งสะดวกต่อผู้ที่เริ่มใช้หรือติดตั้งลินุกซ์. หลังจากที่ติดตั้งลินุกซ์แล้ว, ผู้ใช้สามารถปรับแต่งระบบได้ตามที่ต้องการโดยอาศัยยูทิลิตี้โปรแกรมต่าง ๆ.

Red Hat เป็นดิสทริบิวเชิงพาณิชย์แต่ในขณะเดียวกันอนุญาติให้ผู้ใช้หรือใครก็ได้ดาว์นโหลดทั้งรหัสต้นฉบับ, โปรแกรมติดตั้ง, และแพ็กเกจไบนารีต่าง ๆ และมีบริการแก้ไขแพ็กเกจที่มีข้อบกพร่องแจกจ่ายด้วย. กล่าวคือผู้ที่ต้องใช้ Red Hat ไม่จำเป็นต้องซื้อแผ่นซีดีจาก Red Hat ก็สามารถใช้ Red Hat ได้, ถ้า Red Hat ที่ใช้อยู่มีช่องโหว่เกี่ยวกับความปลอดภัยก็สามารถอัปเดทแพ็กเกจได้โดยไม่เสียค่าใช้จ่าย.

เมื่อปลายปี ค.ศ. 2003 ทาง Red Hat [12] ได้ประกาศการหมดอายุ (end of life) ของ Red Hat Linux 7.1, 7.2, 7.3 และ 8.0. การประกาศมีเนื้อหาเกี่ยวกับการหยุดให้บริการอัปเดท, หยุดการให้บริการ, หยุดการสร้างแพ็กเกจแก้ไขข้อบกพร่องของ Red Hat Linux รุ่นที่ต่ำกว่า 8.0 ภายในสิ้นปี ค.ศ. 2003. และ Red Hat Linux 9 จะหมดอายุขัยภายในเดือนเมษายนปี ค.ศ. 2004. ผู้ใช้ที่ต้องการใช้และบริการอัปเดทต่าง ๆของ Red Hat ต้องใช้ Red Hat Enterprise Linux ซึ่งจะไม่แจกจ่ายแพ็กเกจไบนารีหรือโปรแกรมติดตั้งต่าง ๆ ทางอินเทอร์เน็ต. Red Hat หันมาเน้นธุรกิจทางด้านการให้บริการมากกว่าการขายตัวแพ็กเกจ. ผู้ที่ต้องการใช้ Red Hat Enterprise Linux ต้องเข้าระบบบริการของ Red Hat ซึ่งจะมีหลายระดับตั้งแต่บริการแพ็กเกจแก้ไขโปรแกรมหากมีข้อบกพร่อง, ติดต่อสอบถามทางโทรศัพท์, หรือบริการถึงที่ ฯลฯ [13].

นอกจาก Red Hat Enterprise Linux แล้ว Red Hat ยังเป็นแกนนำสนับสนุนพัฒนาดิสทริบิวชันใหม่ในชื่อ Fedora [14] ซึ่งเปิดกว้างให้ผู้สนใจช่วยกันพัฒนาเป็นดิสทริบิวชันเสรีแทน Red Hat Linux ที่หมดอายุไป.

1.7.3 Debian GNU/Linux

Debian เป็นดิสทริบิวชันที่พัฒนาโดยอาสาสมัครจากทั่วโลกมีชื่อเป็นทางการว่า *Debian GNU/Linux*. Debian ถือกำเนิดโดยนาย Ian Murdock เมื่อวันที่ 16 สิงหาคม ค.ศ.1993. นาย Ian ต้องการที่จะทำให้ดิสทริบิวชันที่เขาเริ่มเปิดเผยและมีอุดมการณ์ตามแบบลินุกซ์และ GNU. การสร้างดิสทริบิวชันนี้ได้รับทุนจาก Free Software Foundation เป็นเวลาหนึ่งปีตั้งแต่เดือนพฤศจิกายนปี ค.ศ.1994 ถึงเดือนพฤศจิกายนปีถัดไป [15]. ชื่อของ Debian (Deb-ian) มาจากชื่อของภรรยาเขาที่มีชื่อว่า Deborah และชื่อของเขาเอง Ian รวมกันเป็นชื่อดิสทริบิวชัน.

จุดเด่นของ Debian คือเป็นดิสทริบิวชันที่ไม่หวังผลกำไร, เปิดเผยและให้โอกาสกับนักพัฒนาทุกคนที่สนใจร่วมพัฒนาดิสทริบิวชัน. มีโปรแกรมอำนวยความสะดวกที่เกี่ยวกับการจัดการแพ็กเกจดีเยี่ยมได้แก่ Advanced Package Management หรือเรียกสั้น ๆว่า APT. สิ่งที่เป็นอุปสรรคสำหรับผู้ใช้ใหม่ได้แก่การติดตั้งซึ่งโปรแกรมติดตั้งจะเป็นเมนูแบบเท็กซ์โหมดและในขณะติดตั้งจะมีการถามคำถามเกี่ยวกับการปรับแต่งโปรแกรมต่าง ๆที่เลือกติดตั้งด้วย. ถ้าผู้ใช้ที่ไม่คุ้นเคยกับลินุกซ์อาจจะไม่เข้าใจทำให้ไม่สามารถตอบคำถามได้เหมาะสม. อย่างไรก็ตามผู้ที่ใช้ Debian มักจะติดตั้งตัวระบบปฏิบัติการเพียงครั้งเดียวและใช้ระบบจัดการแพ็กเกจ APT อัปเกรดระบบปฏิบัติการให้ทันสมัยอยู่เสมอ. ผู้ใช้ไม่จำเป็นต้องติดตั้งระบบทั้งหมดใหม่เมื่อ Debian ประกาศออกตัวรุ่นล่าสุด.

โปรแกรมจัดการแพ็กเกจหลักของระบบจัดการแพ็กเกจ APT ได้แก่ `apt-get`, `apt-cache` ฯลฯ.

เมื่อติดตั้ง Debian แล้วผู้ใช้สามารถเลือกประเภทของ Debian ตามการใช้งานได้สามแบบได้แก่

Stable

เป็นระบบที่รวบรวมแพ็กเกจซอฟต์แวร์ต่าง ๆที่เสถียรและมีการรับรองเป็นทางการจากทีม Debian หากมีจุดบกพร่องด้านความปลอดภัยหรือข้อผิดพลาดอื่น ๆ, จะมีการออกแพ็กเกจอัปเดทแก้ไขให้. คำว่า*เสถียร (stable)* ในที่นี้หมายถึงซอฟต์แวร์แพ็กเกจต่าง ๆที่อยู่ในระบบได้รับการทดสอบ, ตรวจสอบอย่างถี่ถ้วนก่อนจะรวมแพ็กเกจนั้น ๆรวมเป็นระบบ. ทำให้แพ็กเกจที่อยู่ในระบบ stable มีข้อบกพร่องหรือ*บั๊ก (bug)* น้อยที่สุด. ถ้ามีการพบข้อผิดพลาดอย่างร้ายแรงเช่นข้อผิดพลาดด้านความปลอดภัยของตัวซอฟต์แวร์ก็จะมีการออกแพ็กเกจอัปเกรดที่ได้รับการแก้ไขแล้วโดยที่เลขรุ่นหลักของซอฟต์แวร์ที่ใช้ยังเป็นรุ่นเดิม, ไม่ใช่การเปลี่ยนแพ็กเกจไปใช้รุ่นใหม่. ในกรณีนี้มีข้อดีที่ว่าการใช้งานของซอฟต์แวร์ต่าง ๆไม่ว่าจะเป็นวิธีการใช้หรือการปรับแต่งยังเหมือนเดิมเพราะเป็นซอฟต์แวร์รุ่นเดิมที่เคยใช้. ดังนั้นระบบแบบ stable จึงเหมาะสำหรับการใช้งานแบบเซิร์ฟเวอร์.

bug ▶ *บั๊ก, ข้อบกพร่อง.* ข้อบกพร่องของซอฟต์แวร์หลังจากที่เจ้าของซอฟต์แวร์นั้นประกาศออกตัวซอฟต์แวร์นั้น

ในทางกลับกัน, ซอฟต์แวร์ที่เสถียรมีความหมายเป็นนัยว่าซอฟต์แวร์นั้นเป็นรุ่นค่อนข้างเก่า. ซอฟต์แวร์ที่พึ่งออกเผยแพร่ถึงแม้จะมีคุณสมบัติใหม่ ๆน่าใช้แต่อาจจะมีข้อบกพร่องซึ่งยังค้นไม่พบอยู่มากจึงถือว่าไม่เสถียร. ด้วยเหตุนี้เองทำให้แพ็กเกจต่าง ๆที่รวมอยู่ในระบบ stable จึงค่อนข้างเก่าเมื่อเทียบกับระบบแบบอื่น.

Testing

เป็นระบบที่รวมแพ็กเกจที่ผ่านการทดสอบมาระดับหนึ่งจาก unstable และแพ็ก

เกจต่าง ๆเหล่านี้มีข้อบกพร่องหรือบั๊กน้อยพอที่จะเป็นตัวแทนแพ็กเกจที่จะเป็นระบบ stable ต่อไป. ระบบนี้เหมาะสำหรับผู้ที่ต้องการใช้ซอฟต์แวร์รุ่นใหม่ ๆและเสถียรระดับหนึ่ง. อาจจะใช้เป็นระบบเดกสท็อปก้ำกึ่งกับเซิร์ฟเวอร์ซึ่งมีความจำเป็นต้องการซอฟต์แวร์รุ่นใหม่ ๆแต่ไม่ต้องใหม่มากที่สุด.

Unstable

เป็นระบบที่มีแพ็กเกจใหม่ ๆรุ่นล่าสุดซึ่งไม่มีการรับประกันว่าจะใช้ได้ดีตามที่คาดหวัง. บางแพ็กเกจอาจจะยังมีข้อบกพร่องที่ต้องแก้ไข. ระบบนี้มีชื่อว่า unstable ซึ่งแปลความหมายว่า “ไม่เสถียร” แต่ก็ไม่ได้หมายความไม่เสถียรจริง ๆและใช้งานไม่ได้. ระบบนี้เหมาะสำหรับผู้ที่ต้องการใช้ซอฟต์แวร์รุ่นใหม่ล่าสุดและต้องการทดลองความสามารถใหม่ ๆของซอฟต์แวร์นั้น ๆ.

ชื่อรหัสการพัฒนา

ระบบแต่ละแบบจะมี*ชื่อรหัสการพัฒนา (code name)* ซึ่งนำมาจากชื่อตัวละครในภาพยนตร์การ์ตูนเรื่อง Toy Story. ปัจจุบันระบบ stable มีชื่อรหัสการพัฒนาว่า Sarge, testing มีชื่อว่า ??? และ unstable มีชื่อว่า Sid. ชื่อรหัสการพัฒนาของ unstable จะไม่เปลี่ยนแปลงและเรียกว่า Sid เสมอ. ส่วนชื่อรหัสการพัฒนาของระบบอื่น ๆจะเปลี่ยนไปเรื่อย ๆ เมื่อมีการเลื่อนขั้นเช่น Sarge เป็นชื่อรหัสการพัฒนาของ testing มาก่อน. เมื่อระบบมีความเสถียรเป็นที่ยอมรับจึงเลื่อนขั้นมาเป็น stable แต่ยังมีชื่อรหัสเป็น Sarge เหมือนเดิม. ส่วน testing จะมีการตั้งชื่อใหม่ต่อไป.

1.7.4 Gentoo

Gentoo เป็นดิสทริบิวชันที่ค่อนข้างใหม่ประกาศตัวครั้งแรกประมาณเดือนมีนาคมปี ค.ศ.2002. ดิสทริบิวชันนี้มีเอกลักษณ์ที่โดดเด่นแตกต่างจากดิสทริบิวชันอื่นที่ตัวแพ็กเกจเองเป็นรหัสต้นฉบับของซอฟต์แวร์ที่ต้องการติดตั้ง, มีระบบการจัดการแพ็กเกจดีเยี่ยมและมีความยืดหยุ่นสูงในการปรับแต่งระบบ.

โปรแกรมจัดการแพ็กเกจหลักของระบบควบคุมแพ็กเกจ Portage ได้แก่ `emerge` และ `ebuild`.

Gentoo มีระบบการควบคุมแพ็กเกจที่เรียกว่า *portage* ซึ่งได้รับแนวคิดมาจากระบบ port ของยูนิกซ์ BSD. ระบบควบคุมแพ็กเกจนี้สามารถค้นหา, ดาว์นโหลด, แก้ไขความขึ้นกับแพ็กเกจอื่น ๆ (package dependency) และติดตั้งแพ็กเกจที่ต้องการได้โดยอัตโนมัติเช่นเดียวกับระบบ APT ของ Debian. แพ็กเกจของ Gentoo ไม่ใช่ตัวโปรแกรม, ไม่ใช่รหัสต้นฉบับแต่เป็นไฟล์สคริปต์บอกว่ารหัสต้นฉบับอยู่ที่ไหน, มีข้อมูลของวิธีคอมไพล์และติดตั้ง. การติดตั้งแพ็กเกจมีขั้นตอนต่าง ๆได้แก่การดาว์นโหลดรหัสต้นฉบับ, การตรวจสอบความสัมพันธ์กับแพ็กเกจอื่น ๆ, คอมไพล์และติดตั้ง. ในช่วงของการคอมไพล์, ผู้ใช้สามารเลือกปรับแต่งค่าต่าง ๆที่อาจจะมีผลต่อประสิทธิภาพการทำงานเช่นตัวเลือกคอมไพล์ให้เหมาะกับหน่วยประมวลผลข้อมูลที่ใช้, หรือปรับตัวเลือกตอนสร้างของแพ็กเกจนั้น ๆ ให้เหมาะสมตามที่ต้องการได้.

การจัดการแพ็กเกจแบบนี้อาจจะเสียเวลากับการคอมไพล์ถ้าเป็นแพ็กเกจที่ใหญ่แต่มีข้อดีที่สามารถปรับแต่งแพ็กเกจหรือระบบได้จากรหัสต้นฉบับ. การปรับแต่งได้จาก

รหัสต้นฉบับต้นฉบับนี่เองที่ต่างจากดิสทริบิวชันอื่นที่แพ็กเกจเป็นไพล์ไบนารีที่คอมไพล์ไว้เรียบร้อยแล้วซึ่งแพ็กเกจไบนารีเหล่านี้สร้างมาสำหรับคอมพิวเตอร์ทั่วไปไม่ได้เฉพาะเจาะจง.

หลังจากที่ติดตั้ง Gentoo แล้ว, ผู้ใช้สามารถเลือกระบบได้สองประเภทได้แก่

Stable

เป็นระบบที่รวมแพ็กเกจต่าง ๆที่เสถียรเหมาะสำหรับการใช้ตั้งแต่เดกสท็อปส่วนบุคคลถึงเซิฟร์เวอร์.

Unstable

เป็นระบบที่รวแพ็กเกจที่ใหม่ล่าสุดซึ่งอาจจะมีบั๊กแต่ไม่ได้หมายความใช้งานไม่ได้. ระบบนี้เหมาะสำหรับผู้ที่ต้องใช้ซอฟต์แวร์รุ่นใหม่ล่าสุด, ทดลองความสามารถใหม่ ๆ ของซอฟต์แวร์เป็นต้น.

1.7.5 Fedora

Fedora เดิมเป็นโครงการสร้างแพ็กเกจสำหรับใช้ร่วมกับ Red Hat โดยกลุ่มอาสมัครในอินเทอร์เน็ต [16]. โครงการนี้เปิดกว้างให้ใครก็ได้ที่สนใจเข้าร่วมโครงการก็ได้โดยไม่หวังผลกำไรคล้ายกับชุมชนของนักพัฒนา Debian แต่ Fedora เป็นดิสทริบิวชันที่มาฐานมาจาก Red Hat.

ในปีค.ศ.2003, Red Hat ประกาศเลิกสนับสนุน Red Hat ที่แจกจ่ายฟรีโดยเปลี่ยนแนวทางมาสนับสนุน Red Hat ที่เป็นสินค้าจำหน่ายเช่น Red Hat Enterprise โดยที่ผู้ต้องการสนับสนุนจาก Red Hat ต้องเสียค่าใช้จ่าย. เพื่อเป็นการสนับสนุนชุมชนโอเพนซอร์สต่อไป, Red Hat จึงตัดสินใจเลือกโครงการ Fedora ทำเป็นดิสทริบิวชันต่อจาก Red Hat โดยที่ทาง Red Hat เป็นผู้สนับสนุนอย่างเป็นทางการ.

จุดเด่นของ Fedora ได้แก่การเป็นดิสทริบิวชันสร้างจาก Red Hat แต่เปิดกว้างในการพัฒนานาแก่สาธารณะและนำระบบการจัดการแพ็กเกจชั้นสูงของดิสทริบิวชันอื่นมาใช้ด้วย ได้แก่ APT และ `yum`. โดยสภาพรวมทั่วไปแล้วยังคงเหมือนกับ Red Hat.

1.7.6 Knoppix

สำหรับผู้ต้องการลองใช้ลินุกซ์แต่ยังไม่พร้อมที่จะอินสตอลล์ลงในฮาร์ดดิกส์, Knoppix เป็นทางเลือกสำหรับกรณีนี้. Knoppix เป็นแผ่น CD ที่บูตได้, รวบรวมลินุกซ์เคอร์เนลและโปรแกรมเดกสท็อปที่จำเป็นไว้ในแผ่นซีดีแผ่นเดียวโดยมีฐานจาก Debian. แผ่นซีดีนี้สามารถค้นหาฮาร์ดแวร์และอินสตอลล์ไดร์เวอร์ให้โดยอัตโนมัติ. ผู้ใช้เพียงแค่บูตเครื่องคอมพิวเตอร์ด้วยแผ่นซีดี Knoppix ก็จะสามารถใช้โปรแกรมต่าง ๆบนลินุกซ์เช่น KDE, Mozilla, Gimp ฯลฯ. แผ่นซีดีนี้สามารถใช้เป็นแผ่นทดลองหรือจะใช้เป็นแผ่นซีดีกู้ภัยเครื่องคอมพิวเตอร์ก็ได้ในกรณีที่คอมพิวเตอร์ไม่สามารถบูตจากฮาร์ดดิกส์. บางคนใช้ Knoppix เป็นตัวทดสอบก่อนที่จะซื้อเครื่องคอมพิวเตอร์ว่าใช้ลินุกซ์ได้หรือไม่โดยถือแผ่นซีดีไปลองที่ร้าน.

ถ้าลองใช้ Knoppix ไปสักระยะหนึ่งแล้วต้องการติดตั้งลงในฮาร์ดิกส์ก็สามารถทำได้โดยสั่งคำสั่ง `knx-hdinstall` ซึ่งจะแสดงหน้าจอเมนูช่วยแนะนำขั้นตอนต่าง ๆในการติดตั้ง. เนื่องจาก Knoppix พัฒนาจาก Debian, หลังจากติดตั้งเรียบร้อยแล้วก็สามารถใช้คำสั่ง `apt-get` และจัดการระบบได้ทุกอย่างเหมือน Debian.

ช่วงต้นเดือนมีนาคม ค.ศ.2004 คุณ ภัทระ เกียรติเสรี, สมาชิก TLWG ได้ปรับแต่ง Knoppix ให้มีสภาพแวดล้อมเป็นภาษาไทยและใช้ภาษาไทยได้โดยปริยาย [17]. นอกจากนี้ยังมีเพิ่มซอฟต์แวร์ภาษาไทยที่น่าใช้ต่าง ๆด้วยเช่น NECTEC LEXiTEON Dictionary [18], NECTEC Arnthai OCR software [19], Thai LaTeX เป็นต้น.

นอกจากดิสทริบิวชั่นที่กล่าวแล้ว ยังมีดิสทริบิวชั่นอีกหลายค่ายเช่น Suse, Mandriva, Turbolinux ฯลฯ. ผู้อ่านสามารถหาข้อมูลเกี่ยวกับดิสทริบิวชันต่าง ๆได้จากโฮมเพจของ DistroWatch.com [20] ซึ่งเป็นแหล่งรวมข้อมูลของดิสทริบิวชันต่าง ๆที่มีอยู่ทั่วโลกตั้งแต่ดิสทริบิวชันค่ายใหญ่จนถึงค่ายเล็ก.

ชื่อเดิมของ Mandriva คือ Mandrake

1.8 ลินุกซ์ดิสทริบิวชันในประเทศไทย

Linux TLE (ทะเล)

Linux TLE ย่อมาจากคำว่า Linux Thai Language Extension [21] เป็นดิสทริบิวชันที่พัฒนาโดยศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC). มุ่งเน้นการใช้งานภาษาไทยกับลินุกซ์ด้านเดกสท็อป. ประกอบด้วยโปรแกรมใช้งานทั่วไปที่สนับสนุนภาษาไทย ได้แก่ออฟฟิสทะเล, โปรแกรมอ่านอีเมลล์, บราวเซอร์ ฯลฯ.

ลินุกซ์ทะเลเดิมพัฒนามาจาก Mandrake และเปลี่ยนมาพัฒนาจาก Red Hat ในเวลาต่อมา. Linux TLE รุ่นล่าสุดพัฒนามาจาก Fedora.

Linux TLE รุ่นล่าสุดขณะที่เขียนหนังสือเล่มนี้ได้แก่รุ่น 5.5.

Burapha Linux (บูรพาลินุกซ์)

เป็นดิสทริบิวชันไทยที่สร้างโดยคณะวิทยศาตร์ภาคคอมพิวเตอร์ของมหาวิทยาลัยบูรพา. เป็นดิสทริบิวชันที่พัฒนามาจาก Slackware เพื่อให้นักศึกษาเรียนรู้และทำความเข้าใจเกี่ยวกับระบบปฏิบัติการยูนิกซ์ [22].

นอกจากดิสทริบิวชันที่แนะนำไปแล้วยังมีดิสทริบิวชันไทยอื่น ๆด้วยเช่น Grand Linux เป็นดิสทริบิวชันเชิงพาณิชย์และให้บริการกับดิสทริบิวชันในรูปแบบต่าง ๆ.

1.9 Thai Linux Working Group

นอกจากลินุกซ์ดิสทริบิวชันไทยแล้ว, ในประเทศไทยมีกลุ่มชุมชนบนอินเทอร์เน็ตที่ชื่อ *Thai Linux Working Group (TLWG)* [23]. TLWG เป็นกลุ่มของผู้ใช้และพัฒนาซอฟต์แวร์บนลินุกซ์ไม่จำกัดว่าต้องเป็นดิสทริบิวชันใด. TLWG ก่อตั้งขึ้นมาโดยมีจุดประสงค์เพื่อ

บไซด์ [23] โดยตรง

- แลกเปลี่ยนความคิดเห็น และสร้างสรรค์ผลงานที่จะผลักดันให้การใช้งานลินุกซ์ของคนไทยเป็นไปได้อย่างมีประสิทธิภาพ, และถูกต้องตามมาตรฐาน.

- สนับสนุนให้การพัฒนาต่าง ๆในแง่ที่มีผลกับคนไทยโดยรวม, เป็นไปอย่างเปิดเผย.

สมาชิกสามารถสนทนาแสดงความคิดเห็น, ถามตอบ ในเรื่องเกี่ยวกับการใช้งานทั่วไป, การพัฒนาซอฟต์แวร์บนลินุกซ์ ตลอดจนสัพเพเหระได้ผ่านเว็บบอร์ด, mailing list หรือนิวส์กรุป. เว็บไซด์ของ TLWG ที่เรียกย่อ ๆว่า LTN (linux.thai.net)ได้รับการสนับสนุนจากศูนย์เทคโนโลยีอิเล็กทรอนิกส์แห่งชาติ และ Internet Thailand. กิจกรรมและบริการของ TLWG ได้แก่

เว็บบอร์ด

เว็บบอร์ดเป็นที่ถามตอบให้แก่ผู้สนใจได้แก่ เว็บบอร์ดเกี่ยวกับลินุกซ์ทั่วไป, เว็บบอร์ดเกี่ยวกับการพัฒนาซอฟต์แวร์บนลินุกซ์ และเว็บบอร์ดอื่น ๆที่ไม่เข้าข่ายเว็บบอร์ดทั้งสองชนิดข้างต้น.

Mailing list

Mailing list มีเนื้อหาเหมือนกับเว็บบอร์ด, เสนอเนื้อหาในรูปของอีเมลล์เท่านั้น. Mailing list ของ Tlwg เป็นบริการของ yahoogroup.com.

Newsgroup

นิวส์กรุปมีเนื้อหาเหมือนกับเว็บบอร์ด, เสนอเนื้อหาในรูปของนิวส์กรุปโดยมีเซิฟร์เวอร์อยู่ที่ thaigate.nii.ac.jp.

ข่าว

ผู้สนใจสามารถโพสข่าวที่เกี่ยวกับลินุกซ์ขึ้นโฮมเพจหน้าแรกได้. ผู้ดูแลโฮมเพจจะอนุมัติให้ข่าวที่โพสลงในโฮมเพจได้ถ้าเห็นว่าเนื้อหาเหมาะสม.

บทความโดย TLWG หรือสมาชิก

TLWG บริการเนื้อที่สร้างโฮมเพจสำหรับผู้ลงทะเบียนไว้. ผู้ที่สนใจสามารถเขียนบทความที่เกี่ยวกับลินุกซ์เพื่อเป็นประโยชน์สำหรับคนอื่น ๆได้.

CVS

ซอฟต์แวร์ที่ดูแลและพัฒนาโดย TLWG เช่น thailatex, thaifonts-scalable จะเก็บไว้ในเซิฟร์เวอร์โดยใช้ระบบ CVS. ผู้สนใจสามารถใช้คำสั่ง `cvs` เพื่อนำซอฟต์แวร์รุ่นที่ต้องการมาใช้ได้.

CVS ►
Concurrent Versions Systems. เป็นระบบที่ช่วยเก็บและควบคุมเวอร์ชันของรหัสต้นฉบับ. มีประโยชน์อย่างยิ่งโดยเฉพาะการพัฒนาซอฟต์แวร์ร่วมกันหลายคนโดยผ่านทางเน็ตเวิร์ก, ไม่มีข้อจำกัดเรื่องที่อยู่ของนักพัฒนาซอฟต์แวร์.

Ftp

บริการ ftp ให้บริการดาว์นโหลดซอฟต์แวร์หรือเอกสารต่าง ๆที่เกี่ยวกับลินุกซ์.

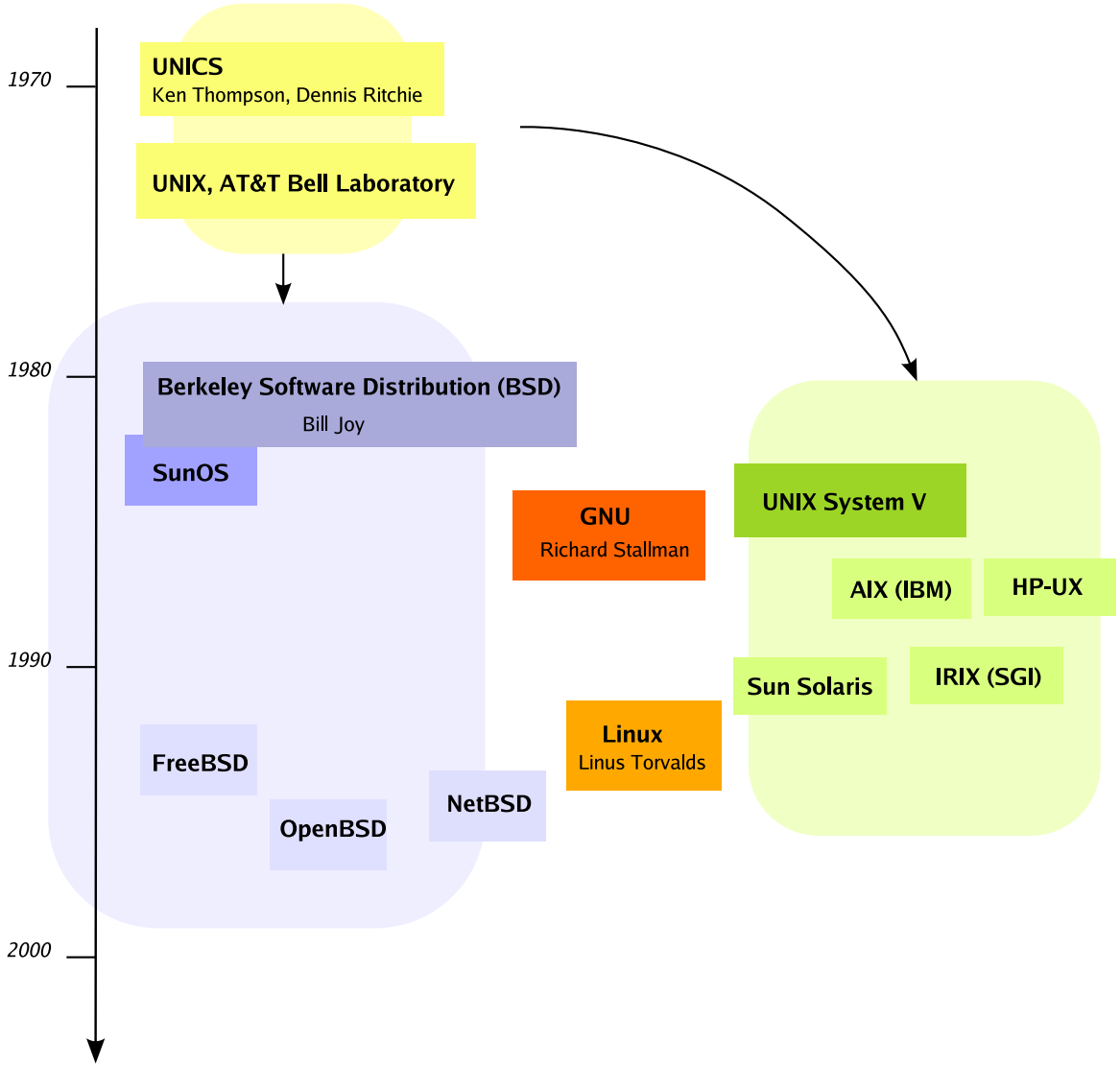
1.10 ลินุกซ์และยูนิกซ์

ถ้าพูดถึงระบบปฏิบัติการลินุกซ์แล้วคงหลีกเลี่ยงที่จะพูดถึงระบบปฏิบัติการยูนิกซ์ด้วยไม่ได้เพราะลินุกซ์เป็นระบบปฏิบัติการตระกูลยูนิกซ์ประเภทหนึ่ง. เพื่อให้ผู้อ่านเข้าใจลินุกซ์มากยิ่งขึ้นจึงแนะนำเรื่องราวเกี่ยวกับระบบปฏิบัติการยูนิกซ์ในช่วงนี้ด้วย.

system call ▶ *ซิสเต็มคอลล์.* ฟังค์ชันภาษาซีสำหรับติดต่อใช้งานเคอร์เนล.

ลินุกซ์สามารถทำทุกอย่างที่ยูนิกซ์ทำได้, มี*คำสั่ง (command)* และ*ซิสเต็มคอลล์ (system call)* เหมือน ๆกัน. สิ่งที่ลินุกซ์แตกต่างจากยูนิกซ์คือ, ลินุกซ์เป็นซอฟต์แวร์เสรีและรหัสต้นฉบับที่เขียนด้วยภาษา C ไม่ได้ลอกเลียนรหัสต้นฉบับของยูนิกซ์ที่มีอยู่แต่เดิม.

ผู้อ่านอาจจะตั้งคำถามว่าทำไมลินุกซ์จึงสร้างเลียนแบบระบบปฏิบัติการยูนิกซ์ทั้ง ๆที่มีระบบปฏิบัติอื่น ๆอีกมากมาย. คำตอบคือเพราะยูนิกซ์เป็นระบบปฏิบัติการที่ออกแบบอย่างดี, มีปรัชญา (UNIX Philosophy) ที่ว่าสร้างระบบปฏิบัติการที่ไม่ใหญ่เกินไป, ทำงาน



รูปที่ 1.1: ระบบปฏิบัติการตระกูลยูนิกซ์

เท่าที่จำเป็น, โปรแกรมต่าง ๆสามารถส่งผลลัพธ์ให้โปรแกรมอื่นต่อได้. ในช่วงต่อไปนี้จะกล่าวถึงประวัติความเป็นมาของระบบปฏิบัติการยูนิกซ์ [24, 25, 26] เพื่อให้ผู้อ่านได้รู้จักยูนิกซ์และเข้าใจพื้นฐานของลินุกซ์ได้ดียิ่งขึ้น.

1.10.1 กำเนิด UNIX

คอมพิวเตอร์ในยุคแรกแตกต่างจากคอมพิวเตอร์ในปัจจุบัน. คอมพิวเตอร์ในยุคแรกเหมือนกับเครื่องคิดเลขขนาดใหญ่ ทำงานได้อย่างเดียวและใช้ได้คนเดียว. ผู้ใช้ซึ่งส่วนใหญ่เป็นนักวิจัยสั่งงานโดยป้อนชุดคำสั่งที่เรียกว่าโปรแกรมผ่านทางเทปหรือ*พันช์การ์ด (pucnh card)*. เมื่อใช้งานเสร็จแล้วผู้ใช้คนต่อไปทำสิ่งเดียวกันคือป้อนข้อมูล, รอคอมพิวเตอร์ทำงาน, รอรับผลที่จะบันทึกกลับลงในเทป. การใช้งานคอมพิวเตอร์ในลักษณะนี้ทำให้เกิดปัญหาหลายอย่างเช่น ในกรณีที่มีผู้ใช้หลายคน คอมพิวเตอร์ไม่สามารถให้บริการได้, การใช้งานแต่ละครั้งมีเสียค่าใช้จ่ายสูงเป็นต้น. จากสาเหตุดังกล่าวจึงเกิดความคิดเกี่ยวกับการสร้างระบบปฏิบัติการขึ้น. ผู้ใช้งานจะสั่งให้คอมพิวเตอร์ทำงานโดยผ่านตัวระบบปฏิบัติการ.

ในปีค.ศ. 1965, Massachusetts Institute of Technology (MIT), Bell Telephone Laboratories (BTL) และ General Electrics Company (GE) ร่วมกันพัฒนาระบบปฏิบัติการแบบ time-sharing ที่เรียกว่า MULTICS (MULtiplexed Information and Computing Service). ในปีค.ศ. 1969 BTL ตัดสินใจถอนตัวจากโครงการดังกล่าว แต่ผู้ร่วมโครงการดังกล่าวจาก BTL ได้แก่ Ken Thompson, Dennis Ritchie, Doug McIlroy, และ J. F. Ossanna ยังรอคอยโอกาสที่จะได้พัฒนาระบบปฏิบัติการในโอกาสหน้า. พวกเขาตั้งใจที่จะพัฒนาระบบปฏิบัติด้วยตัวเอง แต่เนื่องจากคอมพิวเตอร์มีราคาแพงในสมัยนั้น ทำให้ไม่สามารถหาคอมพิวเตอร์ที่จะเอามาทดลองพัฒนาได้. มีความพยายามที่ของบประมาณซื้อคอมพิวเตอร์เพื่อการทดลองแต่ก็ถูกปฏิเสธไป. ถึงแม้ว่าจะไม่สามารถทำการทดลองได้ Thompson, R. H. Canaday, และ Ritchie ได้พัฒนาช่วยกันออกแบบ file system ในทางทฤษฎี.

ในปีค.ศ. 1969, Thompson สร้างเกมส์ “Space Travel”. เกมส์นี้สร้างบนระบบปฏิบัติการ Multics แล้วเขียนใหม่ด้วยภาษา Fortran บนระบบปฏิบัติการ GECOS (ระบบปฏิบัติของ GE ในตอนนั้น). ช่วงนี้เองที่ Thompson หาเครื่องคอมพิวเตอร์ PDP-7 ที่ไม่ค่อยมีใครใช้ได้, จึงเป็นโอกาสให้เขาและ Ritchie เขียนโปรแกรม Space Travel ให้ใช้กับเครื่อง PDP-7 ที่เขาหาได้. สิ่งที่เขาทำก้าวหน้าขึ้นเรื่อย ๆ, เขาเริ่มสร้าง file system ที่เคยคิดไว้. สร้างยูทิลิตี้ระบบดับยูสเซอร์เช่น copy, print, delete, editor, และ shell. โปรแกรมต่าง ๆ พัฒนาด้วย cross-assembler บน GECOS แล้วย้ายตัวโปรแกรมไปสู่ PDP-7 ด้วยเทปกระดาษ. เมื่อสร้าง assembler สำหรับเครื่อง PDP-7 สำเร็จ, การพัฒนาโปรแกรมต่าง ๆก็ไม่ต้องอาศัย GECOS อีกต่อไป. มีแค่ shell, editor, และ assembler ก็สามารถสร้างระบบขึ้นมาใหม่ได้ด้วยตัวเอง(ไม่ต้องพึ่งคอมพิวเตอร์เครื่องอื่น). ต่อมาในปีค.ศ. 1970 Brian Kernighan เรียกระบบปฏิบัติการที่ Thompson และ Ritchie สร้างเลียนแบบ MULTICS ว่า **UNICS** (UNiplexed Information and Computing Service) ซึ่งต่อมากลายเป็นคำว่า **UNIX** นั่นเอง.

GECOS เป็นระบบปฏิบัติการของเครื่องคอมพิวเตอร์ที่ GE ผลิต. หลังจากนั้น HoneyWell เป็นผู้ขายคอมพิวเตอร์นี้ในช่วงเวลาถัดมา

เริ่มแรกยูนิกซ์ใช้ใน*งานประมวลข้อความ (text processing)* ในแผนกสิทธิบัตรของ BTL. โปรแกรมที่ใช้ในตอนนั้นได้แก่ `roff` และ `ed` เป็นต้น. เนื่องจากยูสเซอร์เพิ่มมากขึ้นและ BTL เริ่มเห็นประโยชน์ของระบบปฏิบัติการนี้จึงอนุมัติการซื้อคอมพิวเตอร์รุ่นใหม่คือ PDP-11 ในเวลาต่อมา. เมื่อยูนิกซ์พอร์ตไปใช้กับเครื่อง PDP-11 ได้สำเร็จ ถือว่าเป็นจุดเริ่มของระบบปฏิบัติการยูนิกซ์อย่างเป็นทางการเรียกว่า UNIX First Edition. ต่อมามีการเพิ่มความสามารถการทำงานต่าง ๆเรื่อย ๆจนออกเป็น Second Edition, Third Edition.

1.10.2 จากภาษาแอสเซมบลีสู่ภาษา C

high-level language ▶ *ภาษาชั้นสูง.* หมายถึงภาษาคอมพิวเตอร์ที่มนุษย์อ่านแล้วทำความเข้าใจได้, ไวยกรณ์ไม่ขึ้นกับเครื่องคอมพิวเตอร์ที่ใช้งาน. ตัวอย่างของภาษาชั้นสูงได้แก่ C, C++, Java เป็นต้น.

ยูนิกซ์พอร์ตไปใช้กับเครื่อง PDP-11 ได้ด้วยภาษาแอสเซมบลี. เดิมที Thompson ตั้งใจที่เขียนระบบปฏิบัติการยูนิกซ์ใหม่ด้วย*ภาษาชั้นสูง (high-level language)* แทนภาษาแอสเซมบลี. เขาพยายามเขียนระบบปฏิบัติการยูนิกซ์ด้วยภาษา Fortran แต่ก็ล้มเลิกไปในวันแรกที่ลอง. นอกจากนั้น Thompson ยังสร้างภาษาใหม่ที่เรียกว่าภาษา B (B langauage) เพื่อใช้เขียนระบบปฏิบัติการ แต่มีปัญหาตามมาหลายอย่างเช่น ภาษา B เป็น interpreter ทำให้ทำงานช้าไม่เหมาะกับการเขียนระบบปฏิบัติการเป็นต้น. ต่อมา Ritchie จึงพัฒนาภาษา B ต่อให้เป็นภาษาใหม่ที่เรียกว่าภาษา C.

ในราวปีค.ศ. 1973 เขาทั้งสองเขียนระบบปฏิบัติยูนิกซ์ใหม่หมดด้วยภาษา C แทนภาษาแอสเซมบลี, ทำให้สามารถพอร์ตตัวระบบปฏิบัติการไปใช้กับคอมพิวเตอร์รุ่นใหม่หรือรุ่นอื่นได้ง่ายขึ้น. ปรากฏการนี้ถือว่าเป็นเรื่องใหม่สำหรับวงการคอมพิวเตอร์เพราะระบบปฏิบัติการสมัยนั้นเขียนด้วย*ภาษาชั้นต่ำ (low-level language)* เช่นภาษาแอสเซมบลีเป็นต้น. ในตอนนี้ยูนิกซ์ได้ก้าวเข้าสู่ช่วง Fourth Edition แล้ว. ต่อมายูนิกซ์พอร์ตไปใช้กับคอมพิวเตอร์อื่น ๆนอกจาก PDP-11 เช่น Interdata 7/32, VAX เป็นต้น

1.10.3 Berkeley Software Distribution (BSD)

ในช่วงปีค.ศ. 1976-1977, Thompson ได้รับเชิญจากมหาวิทยาลัย The University of California-Berkeley ให้เป็นศาสตรจารย์พิเศษสอนเกี่ยวกับระบบปฏิบัติการที่เขาสร้าง. และยูนิกซ์ก็เริ่มเป็นที่รู้จักแพร่หลายในวงการศึกษา. ในเวลานั้นนักศึกษามหาวิทยาลัยที่ Berkeley โดยมี Bill Joy เป็นแกนนำ, สร้างยูทิลิตี้โปรแกรม, แก้ไขเคอร์เนล, และรวบรวมแจกจ่ายในชื่อของ Berkeley Software Distribution (BSD) เมื่อปีค.ศ. 1978.

ในปีค.ศ. 1982, Bill Joy, และเพื่อน ๆอีก 3 คนรวมตัวกันสร้างบริษัท Sun Microsystems ซึ่งเป็นบริษัทชั้นนำในวงการยูนิกซ์ปัจจุบัน

ในช่วงนี้ยูนิกซ์พัฒนาไปมาก. Bill Joy สร้างเชลล์ตัวใหม่ที่ชื่อ `csh` มีความสามารถควบคุม*จ็อบ (job)*, มี*ฮิสตอรี่ฟังชั่น (history function)* ซึ่งเชลล์ก่อนหน้านี้ไม่มี. นอกจากนั้นเขายังสร้างบรรณาธิกรณ์ `ex` ซึ่งเปลี่ยนชื่อมาเป็น `vi` ภายหลัง. สิ่งที่สำคัญอีกประการหนึ่งคือในช่วงนั้นกระทรวงกลาโหมของอเมริกาสร้างโปรเจค DARPA (Defense Advanced Research Projects Agency) ซึ่งเป็นแกนนำในการสร้างอินเทอร์เน็ต, ให้เงินทุนสนับสนุนการวิจัยที่ Berkeley อย่างมาก มีผลให้ BSD และยูนิกซ์ในรุ่นต่อมาสนับสนุนการใช้โปรโตคอล *TCP/IP (Transmission Control Protocol / Internet Protocol)* อย่างกว้างขวาง.

`vi` อ่านออกเสียงว่า vee-eye (วีอาย)

1.10.4 UNIX System V

ในขณะที่ Berkeley พัฒนายูนิกซ์เวอร์ชั่นของตัวเอง, ทางด้าน AT&T ก่อตั้ง UNIX Support Group (USG) และสนับสนุนยูนิกซ์ในการค้า. AT&T ออกขายยูนิกซ์ในชื่อ UNIX System III แต่ได้รับการต้อนรับไม่สู้ดีนักเนื่องจากคุณภาพ. ในปีค.ศ. 1983 จึงออกตัวยูนิกซ์ที่ปรับปรุงตัวเคอร์เนลให้มีคุณสมบัติดีขึ้น และเป็นที่รู้จักกันในชื่อของ UNIX System V. หลังจากนั้นมีการปรับปรุงระบบปฏิบัติการให้ดีขึ้นเป็นระยะ ๆ ในชื่อของ UNIX system V Release 2, 3, และ 4.

1.10.5 มาตรฐานยูนิกซ์

ยูนิกซ์แบ่งออกเป็น 2 ค่ายใหญ่ ๆชัดเจนระหว่าง UNIX System V และ BSD. ถึงแม้ว่าจุดเริ่มต้นมาจากที่เดียวกันแต่มีการแก้ไขปรับปรุงรหัสต้นฉบับของเคอร์เนลโดยแต่ละฝ่ายทำให้ยูนิกซ์ทั้งสองไม่มีความเข้ากันได้ (compatibility) ทั้งทางด้านไบนารีและซิสเต็มคอลล์. ซอฟแวร์และฮาร์ดแวร์เวนเดอร์ต่าง ๆเริ่มขายยูนิกซ์ที่ตนเองสร้างแบ่งตามสาย System V และ BSD. ตัวอย่างเช่น SunOS มีพื้นฐานมาจาก BSD เป็นต้น. เดเวล็อปเปอร์ที่พัฒนาโปรแกรมบนยูนิกซ์เกิดความลำบากในการเลือกระบบปฏิบัติที่จะใช้. โปรแกรมที่เขียนบน System V อาจจะต้องนำมาแก้ไขเพื่อให้ใช้ได้บน BSD ทั้ง ๆที่เป็นยูนิกซ์เหมือนกัน.

SunOS เป็นระบบปฏิบัติการเก่าของ Sun Microsystem. ปัจจุบันยูนิกซ์ที่ Sun Mucrosystem ใช้คือ Solaris ซึ่งมีรากฐานจาก System V

ความพยายามที่จะทำให้ยูนิกซ์เป็นมาตรฐานเดียวกันที่เป็นกลางที่สุดได้แก่มาตรฐานของ POSIX (Portable Operating System based on UNIX) ที่กำหนดโดยสถาบัน IEEE (The Institute of Electrical and Electronic Engineers). ตัวอย่างเช่นมาตรฐานนี้กำหนดว่ายูนิกซ์ทุกประเภทต้องประกอบด้วยไลบรารี่มาตรฐานที่กำหนดไว้, ซอฟต์แวร์เวนเดอร์แต่ละรายสามารถเพิ่มไลบรารี่ที่เป็นประโยชน์นอกเหนือจากที่กำหนดได้. นอกจากการกำหนดไลบรารี่แล้ว POSIX ยังกำหนดเชลล์, ยูทิลิตี้, ซิสเต็มคอลล์ที่ต้องมีอยู่ในระบบปฏิบัติการยูนิกซ์อีกด้วย.

1.11 สรุปท้ายบท

- ลินุกซ์ คือระบบปฏิบัติการที่มีคุณสมบัติคล้ายเหมือนกับระบบปฏิบัติการยูนิกซ์ แต่พัฒนาขึ้นมาโดยไม่ได้ลอกเลียนหรือสืบทอดรหัสต้นฉบับของยูนิกซ์. โปรแกรมที่ใช้บน ลินุกซ์ มีความเข้ากันได้กับระบบปฏิบัติการยูนิกซ์อื่นในระดับรหัสต้นฉบับ.

- ลินุกซ์ สามารถแบ่งเป็นสองส่วนใหญ่ ๆคือ *เคอร์เนล*และ*โปรแกรมใช้งาน*. โปรแกรมใช้งานต่าง ๆทำงานได้โดยมีเคอร์เนลเป็นพื้นฐานทำหน้าที่ควบคุมการใช้งานของโปรเซสเซอร์, หน่วยความจำ, ดิกส์ ฯลฯ. เคอร์เนลโดยลำพังทำงานเองไม่ได้.

- แก่นของระบบปฏิบัติการที่เรียกว่าเคอร์เนลเขียนด้วยภาษา C เป็นหลัก. รหัสต้นฉบับของระบบปฏิบัติการเปิดเผยแก่สาธารณะ. ลินุกซ์ ให้อิสระแก่ผู้ใช้ที่จะศึกษา, ดัดแปลง, ก๊อบปี้ ฯลฯ ตราบเท่ารหัสต้นแบบที่เปลี่ยนแปลงเปิดเผยต่อสาธารณะต่อไป.

- เคอร์เนลและโปรแกรมใช้งานต่าง ๆพัฒนาหรือพอร์ตโดยเดเวลลอปเปอร์ทั่วโลกโดยอาศัยอินเทอร์เน็ตเป็นสื่อ. ตัวเคอร์เนลและโปรแกรมต่าง ๆฟรี. ฟรีในที่นี้ไม่ได้ถึงราคาเป็น “0” แต่หมายถึงความมีอิสระเสรีในการใช้.

- ลินุกซ์ สามารถใช้งานได้กับคอมพิวเตอร์ตั้งแต่คอมพิวเตอร์ขนาดใหญ่ถึงคอมพิวเตอร์ระบบเอ็มเบ็ดด์.

- ลินุกซ์ ดิสทริบิวชั่นคือการนำเคอร์เนลและโปรแกรมใช้งานต่าง ๆมารวมกันเป็นแพคเกจ มีอินสตอลเลอร์ช่วยอินสตอลล์เคอร์เนลและโปรแกรมต่าง ๆ ตลอดจนปรับระบบให้เข้ากับคอมพิวเตอร์ที่ใช้อีกด้วย.

- ระบบปฏิบัติการยูนิกซ์เป็นระบบปฏิบัติการที่เสถียร, มีประสิทธิภาพ, และออกแบบอย่างดีแฝงปรัชญาของความเรียบง่ายไว้ภายใน. ยูนิกซ์มีอายุกว่า 30 ปีและยังพัฒนาให้ดีขึ้นเรื่อย ๆในปัจจุบัน.

บทที่ 2
ลินุกซ์ขั้นพื้นฐาน

"LINUX is obsolete"
Andrew Tanenbaum

จากบทที่ 1 เราได้ทราบแล้วว่าลินุกซ์สร้างขึ้นมาโดยมียูนิกซ์เป็นต้นแบบ. คำจำกัดความอย่างง่าย ๆและสั้น ๆของลินุกซ์ในช่วงที่เกิดใหม่ ๆคือ "unix clone". กล่าวคือลินุกซ์ทำอะไรได้ทุกอย่างที่ยูนิกซ์ทำได้. เมื่อวันเวลาผ่านไป, ลินุกซ์พัฒนาไปเรื่อย ๆเพิ่มความสามารถคุณสมบัติพิเศษต่าง ๆจนไม่อาจจะเรียกว่าเป็น unix clone ได้อีกต่อไป. อย่างไรก็ตามคำสั่งหรือโปรแกรมพื้นฐานต่าง ๆที่ใช้ในยูนิกซ์ได้ก็จะมีอยู่ในลินุกซ์ด้วย. บางโปรแกรมอาจจะทำงานเหมือนกันทุกประการ, บางโปรแกรมที่มีชื่อเหมือนกันอาจจะทำงานต่างกัน, หรือบางโปรแกรมที่ยูนิกซ์ไม่มีแต่ลินุกซ์มี.

ในบทนี้จะแนะนำความรู้พื้นฐานเกี่ยวกับลินุกซ์ซึ่งจะครอบคลุมถึงยูนิกซ์บางส่วนด้วย. เนื้อหาส่วนใหญ่จะเกี่ยวกับการใช้งาน*เชลล์ (shell)* และจะแทรกตัวอย่างการใช้โปรแกรมต่าง ๆด้วย. นอกจากนั้นจะแนะนำความรู้เบื้องต้นทั่ว ๆไปที่เกี่ยวกับตัวระบบปฏิบัติการเองด้วย.

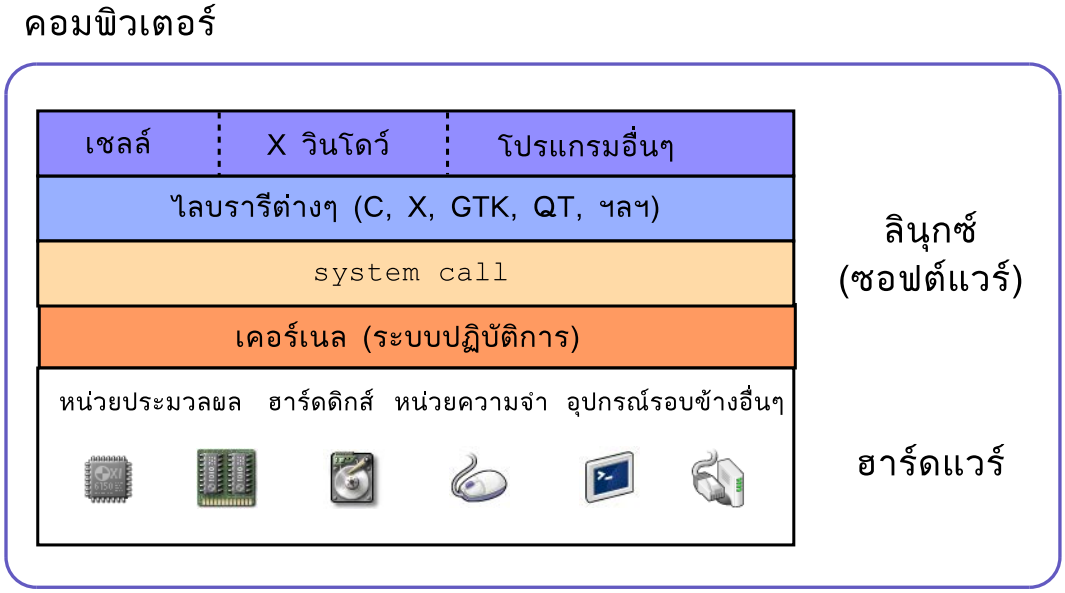
shell ▶
เชลล์. โปรแกรมที่เป็นตัวกลางระหว่างผู้ใช้กับเคอร์เนล. มีหน้าที่แปลคำสั่งจากคีย์บอร์ดส่งต่อให้เคอร์เนลทำงานที่ต้องการ. เชลล์ที่นิยมใช้กับลินุกซ์ได้แก่ `bash`, `csh`, `ksh`, `zsh` เป็นต้น.

ผู้อ่านจะทำความเข้าใจตัวอย่างได้ง่ายขึ้นถ้าติดตั้งลินุกซ์ไว้เรียบร้อยแล้วและทดลองตามไปด้วย. ลินุกซ์ที่ติดตั้งจะเป็นดิสทริบิวชันค่ายใดก็ได้แล้วแต่สะดวก. ผู้อ่านสามารถอ่านรายละเอียดการติดตั้งลินุกซ์ได้จากภาคผนวก.

2.1 ระบบปฏิบัติการ

เนื่องจากลินุกซ์เป็นระบบปฏิบัติการ, จึงมีความสำคัญอย่างยิ่งที่ผู้อ่านควรจะทำความเข้าใจว่าระบบปฏิบัติการคืออะไรและมีการทำงานอย่างไร. เมื่อผู้อ่านมีความรู้เบื้องต้นเกี่ยวกับระบบปฏิบัติการแล้วก็จะทำให้ผู้อ่านเข้าใจการทำงานของลินุกซ์และใช้งานลินุกซ์ได้ดีขึ้นด้วย.

สำหรับความหมายทั่วไป, ลินุกซ์หมายถึงตัวระบบปฏิบัติการซึ่งได้เคอร์เนลและโปรแกรมต่าง ๆประกอบขึ้นเป็นระบบที่ใช้งานกับคอมพิวเตอร์. เพื่อความชัดเจนยิ่งขึ้น, ต่อ



รูปที่ 2.1: ความสัมพันธ์ระหว่างระบบปฏิบัติการ, ฮาร์ดแวร์และซอฟต์แวร์

ไปนี้จะใช้คำว่า “ลินุกซ์” ในความหมายทั่วไปและจะใช้คำว่า “ลินุกซ์เคอร์เนล” เจาะจงถึงตัวเคอร์เนลซึ่งเป็นหัวใจของระบบปฏิบัติการ.

interface ▶
อินเทอร์เฟส. หมายถึงตัวเชื่อมหรือตัวกลางระหว่างของสองสิ่ง. ตัวอย่างเช่น เชลล์เป็นอินเทอร์เฟสแบบคำสั่งบรรทัดระหว่างยูสเซอร์กับเคอร์เนล. X วินโดว์เป็นอินเทอร์เฟสแบบหน้าต่างใช้เมาส์และคีย์บอร์ดในการป้อนข้อมูล

program ▶
โปรแกรม. ชุดคำสั่งภาษาเครื่องกล (machine language) ที่สามารถโหลดเข้าไปในหน่วยความจำและหน่วยประมวลผลแปลความภาษาเครื่องกลเพื่อที่จะทำงานต่อไป.

file ▶
ไฟล์, แฟ้มข้อมูล. ไฟล์คือข้อมูลซึ่งอาจจะเก็บบันทึกในหน่วยความจำถาวรเช่นฮาร์ดดิกส์เป็นต้น. ข้อมูลที่เรียกว่าไฟล์นี้มีลำดับ, ข้อมูลที่บันทึกก่อนจะอยู่ในช่วงต้นของไฟล์. ข้อมูลที่เก็บล่าสุดจะอยู่ในช่วงท้ายของไฟล์.

machine language ▶
ภาษาเครื่องกล. ข้อมูลที่หน่วยประมวลผลเข้าใจและแปลความหมายเพื่อทำงานได้.

process ▶
โปรเซส. สภาพที่โปรแกรมที่กำลังทำงานอยู่. กล่าวคือเนื้อหาของตัวโปรแกรมถูกโหลดจากฮาร์ดดิกส์ไปที่หน่วยความจำและหน่วยประมวลผลข้อมูลแปลคำสั่งเพื่อที่จะกระทำการต่อไป.

รูปที่2.1 แสดงภาพรวมของระบบคอมพิวเตอร์โดยทั่วไป. คอมพิวเตอร์ (ฮาร์ดแวร์) ไม่สามารถทำงานได้ถ้าไม่มีระบบปฏิบัติการ (ซอฟต์แวร์). ผู้ใช้ไม่สามารถใช้คอมพิวเตอร์ได้ถ้าไม่มีโปรแกรมหรือซอฟต์แวร์ต่าง ๆเป็น*ตัวกลาง (interface)* ระหว่างผู้ใช้กับระบบปฏิบัติการ. เคอร์เนลเป็นซอฟต์แวร์พิเศษที่ควบคุมการทำงานของฮาร์ดแวร์ต่าง ๆที่ประกอบกันเป็นคอมพิวเตอร์. ผู้ใช้ไม่สามารถติดต่อใช้งานเคอร์เนลได้โดยตรง, ต้องสั่งงานที่ต้องการทำผ่านทางเชลล์, X วินโดว์ หรือโปรแกรมอื่น ๆ. ในรูปที่ 2.1 มีการแยกเชลล์, X วินโดว์ และโปรแกรมอื่น ๆออกจากกัน. แต่ถ้ามองจากจุดยืนของเคอร์เนลแล้ว, เชลล์หรือ X วินโดว์เป็นเพียงสิ่งที่เรียกว่า*โปรแกรม (program)* เท่านั้น, ไม่มีความแตกต่างกันแต่ประการใด. โปรแกรมเหล่านี้เรียกอีกอย่างได้ว่าเป็น*ซอฟต์แวร์อินเทอร์เฟส (software interface)* ให้กับเคอร์เนล.

โปรแกรมคือ*ไฟล์ (file)* ข้อมูลที่บันทึกคำสั่งเป็น*ภาษาเครื่องกล (machine language)* เก็บไว้ในหน่วยความจำถาวรเช่นฮาร์ดดิกส์. เมื่อสั่งคำสั่งเรียกใช้โปรแกรมแล้ว, เคอร์เนลจะอ่านเนื้อหาของไฟล์ (โปรแกรม) แล้ว*โหลด (load)* บันทึกในหน่วยความจำ. เคอร์เนลจะควบคุมให้หน่วยประมวลผลอ่านข้อมูลที่อยู่บนหน่วยความจำกระทำการต่าง ๆต่อไป. โปรแกรมที่โหลดขึ้นสู่หน่วยความจำและกำลังทำงานอยู่เรียกว่า*โปรเซส (process).*

ลินุกซ์เคอร์เนลสร้างขึ้นมาด้วย*ภาษาซี (C language)* กับ*ภาษาแอสเซมบลี (assembly language)* บางส่วน. การที่โปรแกรมและ*ไลบรารี (library)* ต่าง ๆจะติดต่อกับเคอร์เนลเพื่อสั่งงาน, ต้องติดต่อผ่านทาง*ซิสเตมคอล (system call)* ซึ่งเป็น*ฟังชัน (function)* ในภาษาซี.

2.2 การเข้าสู่ระบบ

ลินุกซ์เป็นระบบปฏิบัติการที่*ผู้ใช้ (user)* หลายคนสามารถใช้งานได้ในเวลาเดียวกัน. ผู้ใช้แต่ละคนจะมี*ชื่อล็อกอิน (login name)* และ*รหัสผ่าน (password)* เฉพาะที่แตกต่างกัน. ผู้ใช้ต้อง*ล็อกอิน (login)* เข้าสู่ระบบเพื่อที่จะใช้คอมพิวเตอร์ผ่านทางเชลล์หรือ X วินโดว์ต่อไป.

การเข้าสู่ระบบของระบบปฏิบัติการ Windows จะเรียกว่า logon แต่สำหรับโลกของยูนิกซ์จะเรียกว่า login

ผู้ใช้ที่มีอำนาจสูงสุดในระบบได้แก่ *root* ทำหน้าที่ดูแลระบบเช่น สร้างผู้ใช้ทั่วไป, ติดตั้งโปรแกรมลงในระบบ, ปรับแต่งระบบ เป็นต้น. ตอนติดตั้งลินุกซ์, โปรแกรมติดตั้งจะสร้างผู้ใช้ที่เป็น root ให้โดยปริยายแล้วให้สร้างรหัสผ่านด้วย. โดยทั่ว ๆไป, โปรแกรมติดตั้งจะเปิดโอกาสให้สร้างผู้ใช้อื่น ๆด้วย. เนื่องจาก root สามารถทำอะไรก็ได้กับระบบเช่นลบไฟล์อะไรก็ได้, จึงทำให้การล็อกอินเป็น root ไม่ปลอดภัยเท่าที่ควรเพราะอาจเผลอเรอลบไฟล์ที่จำเป็นกับระบบปฏิบัติการทิ้งไปโดยไม่ตั้งใจ. ดังนั้นจึงไม่ควรล็อกอินเป็น root ถ้าไม่จำเป็น.

C language ▶
ภาษาซี. ภาษาคอมพิวเตอร์ที่มนุษย์สามารถอ่านทำความเข้าใจได้. เป็นภาษาชั้นสูงที่มีไวยกรณ์แน่นอน. โดยปรกติผู้ใช้จะคอมไพล์รหัสต้นฉบับที่เขียนด้วยภาษาซีเพื่อแปลงเป็นภาษาเครื่องกลที่คอมพิวเตอร์เข้าใจซึ่งเรียกว่าโปรแกรม. ภาษาซีสร้างโดย Dennis Ritchie นักวิจัยของ Bell Laboratories ในราวปี ค.ศ. 1972. ภาษาซีเป็นภาษาใช้งานทั่วไป (general purpose) และใช้ในสร้างระบบปฏิบัติการยูนิกซ์ในเวลาต่อมา. ลินุกซ์เคอร์เนลและโปรแกรมใช้งานส่วนใหญ่ก็ใช้ภาษาซีเขียนเช่นกัน.

assembly language ▶
ภาษาแอสเซมบลี. ภาษาคอมพิวเตอร์ชั้นต่ำ, มีความขึ้นอยู่กับสถาปัตยกรรมที่ใช้มากทำให้ยากต่อการพอร์ตโปรแกรมไปใช้กับสถาปัตยกรรมอื่น ๆ.

2.2.1 ผู้ใช้ในระบบ

ผู้ใช้ในระบบปฏิบัติการลินุกซ์และยูนิกซ์จะมีหมายเลขประจำตัวซึ่งเรียกว่า *user ID* หรือเรียกย่อ ๆว่า *UID* เป็นหมายเลขเฉพาะสำหรับเคอร์เนลไว้แยกแยะผู้ใช้แต่ละคน. ผู้ใช้แต่ละคนจะอยู่ใน*กรุป (group)* ที่กำหนดไว้อย่างน้อยหนึ่งกลุ่ม. เคอร์เนลจะแยกแยะกรุปของผู้ใช้ด้วยหมายเลขเฉพาะเรียกว่า *group ID* หรือเรียกย่อ ๆว่า *GID*. ผู้ใช้จะมีสิทธิ์การใช้คำสั่ง, หรือการใช้ไฟล์ต่าง ๆกันขึ้นอยู่กับกรุปที่ตัวเองอยู่และ UID.

ในระบบปฏิบัติการลินุกซ์จะมีผู้ใช้ที่ระบบเตรียมอยู่ไว้แล้วเช่น bin, daemon ฯลฯ. ผู้เหล่านี้บางรายไม่ได้มีตัวตนจริง, ระบบจะใช้ผู้ใช้เหล่านี้ในการรันโปรแกรมพิเศษต่าง ๆเช่นโปรแกรมเซิฟร์เวอร์เป็นต้น. ในระบบยังมีกลุ่มที่เตรียมไว้อยู่แล้วเช่นเดียวกับผู้ใช้เช่น bin, wheel, sys, mail ฯลฯ. ผู้ใช้ที่อยู่ในกลุ่ม mail ก็จะได้รับอนุญาตให้กระทำการต่าง ๆที่เกี่ยวกับการดูและระบบ mail ได้เป็นต้น.

วิธีการสำรวจ UID และ GID จะใช้คำสั่ง `id` ซึ่งจะแสดง UID และ GID ของผู้ใช้ที่สั่งคำสั่งนั้น.

id อ้างอิงหน้า 413

ตัวอย่างที่ 2.1: สำรวจ UID และ GID.

```
$ id↵
uid=1000(poonlap) gid=100(users) groups=10(wheel),18(audio),100(users)
```

ในระบบปฏิบัติการลินุกซ์จะมีโปรแกรม “su” สำหรับเปลี่ยน UID ของผู้ใช้ให้เป็น UID ของคนอื่น. เช่นใช้เปลี่ยนผู้ใช้ที่ล็อกอินอยู่เป็น root.

su อ้างอิงหน้า 407

เครื่องหมาย ▮ แสดงถึงคีย์บอร์ด*เคอร์เซอร์ (cursor)*

ตัวอย่างที่ 2.2: การใช้ su ด้วยตัวเลือก -

```
$ su -↵
Password:                          ← รหัสผ่านที่พิมพ์จะไม่แสดงบนหน้าจอ
# ▮                                ← ล็อกอินเชลล์ของ root ซึ่งอ่านค่าเริ่มต้นต่างๆแล้ว
```

จากตัวอย่างเป็นการใช้คำสั่ง su กับ*ตัวเลือก (option)* "-" หมายถึงการเปลี่ยนผู้ใช้เป็น root และใช้ล็อกอินเชลล์. การใช้ตัวเลือก - จะเป็นการบอกให้เชลล์อ่านค่าติดตั้งเริ่มต้นของ root. ถ้าไม่ต้องการให้เชลล์อ่านค่าติดตั้งเริ่มต้นของ root ให้สั่งคำสั่ง su โดยไม่มีตัวเลือก.

ตัวอย่างที่ 2.3: การใช้คำสั่ง su ไม่มีตัวเลือก

```
$ su↵
Password:                                   ← รหัสผ่านที่พิมพ์จะไม่แสดงบนหน้าจอ
# ■                                  ← เชลล์เชิงโต้ตอบ, ไม่มีการอ่านค่าเริ่มต้นของ root
```

sg อ้างอิงหน้า 407

คำสั่งที่คล้ายกับคำสั่ง su คือ sg ซึ่งเป็นคำสั่งเปลี่ยนกลุ่มหลักเป็นกลุ่มอื่น.

ตัวอย่างที่ 2.4: การเปลี่ยนกลุ่ม.

```
$ sg wheel↵
$ id↵
uid=1000(poonlap) gid=10(wheel) groups=10(wheel),18(audio),27(video),100(users)
```

จากตัวอย่างจะเห็นว่ากลุ่มปัจจุบันหรือกลุ่มปัจจุบันเปลี่ยนเป็น wheel แทนกลุ่มโดยปริยาย users. นอกจากคำสั่ง sg แล้วยังมีคำสั่ง newgrp ซึ่งทำหน้าที่คล้ายกับ sg.

newgrp อ้างอิงหน้า 407

UID และ GID นี่จะเกี่ยวข้องกับสิทธิ์การใช้ไฟล์, รันโปรแกรม ซึ่งจะแนะนำในบทถัดไป.

2.2.2 ประเภทของการล็อกอินแบ่งตามอินเทอร์เฟส

เท็กซ์โหมด (text mode)

text mode ▶
เท็กซ์โหมด. สภาพของระบบปฏิบัติที่อินเทอร์เฟสเป็นคีย์บอร์ดและหน้าจอเทอร์มินัลเท่านั้น.

ตัวอย่างต่อไปนี้แสดงตัวอย่างการล็อกอินของผู้ใช้ชื่อ somchai แบบ*เท็กซ์โหมด (text mode)* ผ่านทางเทอร์มินอล. การล็อกอินในลักษณะนี้มักจะเป็นการล็อกอินจาก*คอนโซล (console)* หรือใช้คำสั่ง telnetล็อกอินผ่านทางเน็ตเวิร์กจากคอมพิวเตอร์เครื่องอื่น.

ในเท็กซ์โหมด, ถ้าผู้ใช้ธรรมดากดคีย์ Ctrl+Alt+Del จะทำให้ลินุกซ์*รีบูต (reboot)*. ผู้ดูแลระบบสามารถปรับแต่งระบบได้ถ้าไม่ต้องการให้ผู้ใช้รีบูตเครื่องได้ด้วยวิธีนี้

ตัวอย่างที่ 2.5: การล็อกอินแบบเท็กซ์โหมด

```
login: somchai↵
Password:                                   ← รหัสผ่านที่พิมพ์จะไม่แสดงบนหน้าจอ
[somchai@toybox somchai]$ ■
```

เมื่อล็อกอินแล้วผู้ใช้สามารถสั่งคำสั่งต่าง ๆได้โดยการพิมพ์คำสั่งจากคีย์บอร์ด. อินเทอร์เฟสแบบนี้เรียกกันว่า Command Line Interface (CLI) หรือ Text User Interface . เราเรียก "[somchai@toybox somchai]$" ว่า*เชลล์พรอมต์ (shell prompt)* บ่งบอกว่าเชลล์พร้อมที่จะรับคำสั่งจากคีย์บอร์ด.

command line interface ▶
ระบบอินเทอร์เฟสที่ใช้คีย์บอร์ดและหน้าจอแสดงตัวอักษรเป็นหลัก. ถ้าเป็นงานที่ไม่ต้องการกราฟฟิกจะเป็นอินเทอร์เฟสที่สะดวกมากและปฏิบัติการได้เร็วกว่า graphical user interface.

prompt ▶
พรอมต์. เครื่องหมาย, หรืออักษรที่เชลล์แสดงทางหน้าจอเพื่อแสดงว่าพร้อมที่จะรับคำสั่งจากคีย์บอร์ด.

ตัวอย่างที่ 2.5 แสดงเป็นพรอมต์ที่ปรับแต่งให้จากดิสทริบิวชันโดยแสดงชื่อผู้ใช้ (somchai), ชื่อโฮส (toybox) และชื่อไดเรกทอรีปัจจุบัน (somchai). เชลล์ที่ทำงานหน้าที่รอรับคำสั่งหลังจากการล็อกอินเรียกว่า*ล็อกอินเชลล์ (login shell)*. ส่วนเชลล์ที่ไม่ใช่ล็อก

อินเชลล์เราเรียกว่า*เชลล์เชิงโต้ตอบ (interactive shell)*. เชลล์เชิงโต้ตอบเป็นเชลล์ที่ไม่ได้เกิดจากการล็อกอินแต่เป็นการเรียกโปรแกรมเชลล์มาใช้งานเฉย ๆ, เช่นการใช้เชลล์ในเทอร์มินอลเอมิวเลเตอร์. เชลล์จะทำงานอยู่ในไดเรกทอรีที่เรียกว่า*โฮมไดเรกทอรี (home directory)* ซึ่งเป็นไดเรกทอรีเริ่มต้นหลังจากการล็อกอิน. ไดเรกทอรีนี้เป็นพื้นที่เฉพาะของผู้ใช้ที่ล็อกอินสำหรับเก็บข้อมูลต่าง ๆ. ผู้ใช้มีสิทธิ์ที่จะสร้างไฟล์หรือสร้างไดเรกทอรีเพิ่มเติมในโฮมไดเรกทอรีของตัวเอง.

home directory ▶
โฮมไดเรกทอรี. ไดเรกทอรีที่ผู้ใช้เป็นเจ้าของ. เป็นไดเรกทอรีเริ่มต้นเวลาล็อกอินเข้าสู่ระบบหรือเรียกใช้เทอร์มินัลเอมิวเลเตอร์.

ล็อกอินเชลล์และเชลล์โต้ตอบต่างกันที่เวลาเริ่มต้นการทำงาน, ล็อกอินเชลล์จะอ่านและเซ็ตค่าเริ่มต้นการทำงานเช่นค่าตัวแปรสภาพแวดล้อมจากไฟล์ `/etc/profile` ถ้ามีไฟล์นี้อยู่ในระบบ. หลังจากนั้นจะหาไฟล์ `.bash_profile`, `.bash_login` และ `.login` ที่อยู่ในโฮมไดเรกทอรีตามลำดับ. ถ้าเจอไฟล์ไหนก่อนก็จะอ่านค่าเริ่มต้นจากไฟล์นั้น. ส่วนเชลล์เชิงโต้ตอบจะอ่านค่าเริ่มต้นจากไฟล์ `.bashrc` ที่อยู่ในโฮมไดเรกทอรีเท่านั้น.

ในโลกของ Windows จะเรียกไดเรกทอรีว่าโฟลเดอร์ (folder)

เพื่อความสะดวกในการอ่าน, ต่อจากนี้เป็นต้นไปจะใช้เครื่องหมาย “$” แทนเชลล์พรอมต์ของผู้ใช้ธรรมดา.

ตัวอย่างที่ 2.6: เชลล์พรอมต์ของผู้ใช้ทั่วไป

```
$ ■
```

และใช้เครื่องหมาย “#” แทนเชลล์พรอมต์ของ root.

ตัวอย่างที่ 2.7: เชลล์พรอมต์ของ root

```
# ■
```

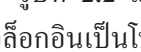
กราฟฟิกโหมด (graphic mode)

รูปที่ 2.2 แสดงตัวอย่างหน้าจอล็อกอินแบบ*กราฟฟิกโหมด (graphic mode)*. หน้าจอล็อกอินเป็นโปรแกรมที่เรียกว่า X Display Manager มีหน้าที่ถามชื่อผู้ใช้และรหัสผ่าน. โปรแกรมหน้าจอต้อนรับเข้าสู่ระบบนี้มีหลายประเภทเช่น `xdm`, `gdm`, `kdm` เป็นต้น. การล็อกอินแบบนี้ต้องการคอมพิวเตอร์ที่สามารถแสดงผลแบบกราฟฟิกทางหน้าจอได้.

graphic mode ▶
กราฟฟิกโหมด. สภาพที่ระบบปฏิบัติการใช้อินเทอร์เฟสแบบกราฟฟิกติดต่อกับผู้ใช้. โดยทั่วไปคอมพิวเตอร์แบบกราฟฟิกโหมดสามารถผลเป็นแบบหน้าต่าง, สร้างปุ่ม, เมนู ฯลฯ. ผู้ใช้โต้ตอบกับคอมพิวเตอร์ด้วยคีย์บอร์ดและเมาส์.

X Display Manager ▶
หน้าจอแบบกราฟฟิกใช้ในการล็อกอิน. ก่อนที่จะมี GNOME และ KDE, ระบบ X วินโดว์ใช้ X Display Manager ที่เรียกว่า `xdm` ซึ่งปัจจุบันไม่นิยมใช้กันแล้ว. โปรแกรมที่มาแทน `xdm` ได้แก่ `gdm`, `kdm` ฯลฯ.

กราฟฟิกโหมดต่างจากเท็กซ์โหมดตรงที่มี *X เซิร์ฟเวอร์ (X server)* ซึ่งเป็นโปรแกรมพื้นฐานสำหรับระบบวินโดว์คอยรับคำสั่งสร้างส่วนประกอบของกราฟฟิก. ดิสทริบิวชันทั่วไปมักจะมีระบบ Desktop Environment เช่น GNOME หรือ KDE เป็นระบบเดกส์ท็อปมาตรฐานให้ผู้ใช้ใช้งาน. ระบบ desktop environment นี้เองที่เป็นตัวจัดการการทุกอย่างที่เกี่ยวกับเดกส์ท็อปเช่นหน้าจอล็อกอิน, เมนู, โปรแกรมสำหรับใช้งานที่เป็นแบบ graphical user interface (GUI) ให้. ในกราฟฟิกโหมดผู้ใช้จะใช้งานได้สะดวกกว่าเท็กซ์โหมด, เพราะสามารถรันโปรแกรมหลายโปรแกรมโดยที่แต่ละโปรแกรมมีวินโดว์เฉพาะของตัวเอง. เนื่องจากระบบ GUI มีเมนูต่าง ๆเตรียมไว้ให้, ผู้ใช้ที่ไม่คุ้นเคยกับการใช้เชลล์ก็สามารถใช้โปรแกรมต่าง ๆได้สะดวกขึ้น.



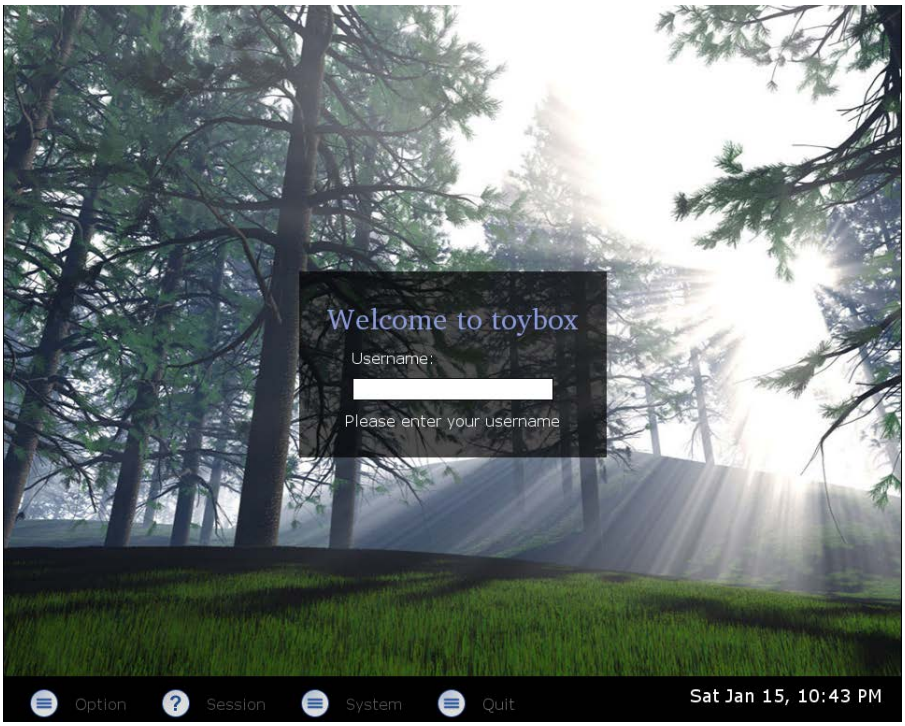
ในกราฟฟิกโหมด, ถ้าผู้ใช้กดคีย์ Ctrl+Alt+BackSpace จะเป็นการยกเลิกการทำงาน X เซิร์ฟเวอร์ที่ใช้อยู่. ผู้ดูแลระบบสามารถปรับแต่งระบบให้ยกเลิกการใช้คีย์เหล่านี้ได้.

Desktop Environment ▶
ระบบ, สภาพแวดล้อมที่เสนอกราฟฟิกอินเทอร์เฟสสำหรับผู้ใช้โดยที่อินเทอร์เฟสเหล่านั้นมีความเข้ากันได้ดีเช่นรูปร่างหน้าตาเหมือนหรือคล้ายกัน, ทำงานร่วมกันได้ ฯลฯ. ตัวอย่าง Desktop Environment ที่นิยมได้แก่ GNOME, KDE เป็นต้น.

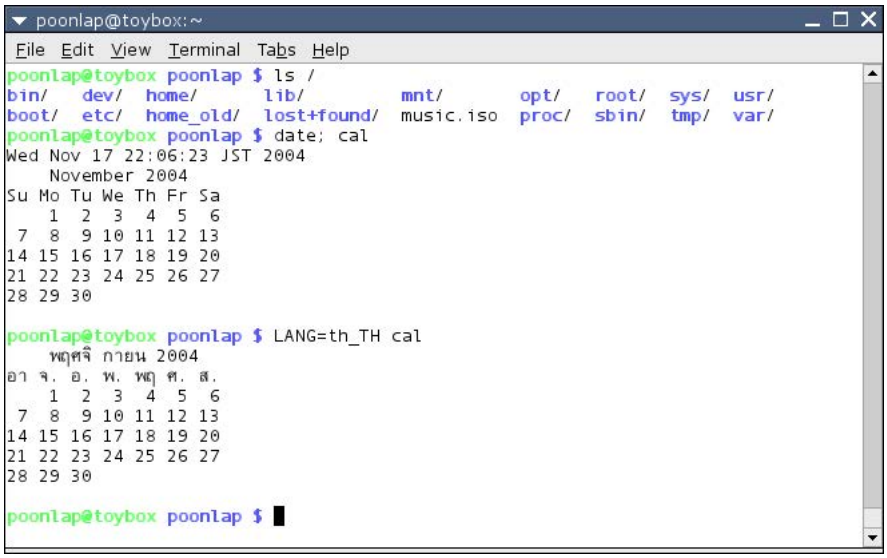
graphical user interface (GUI) ▶
อินเทอร์เฟสแบบกราฟฟิกที่แสดงหน้าต่าง, ปุ่มกด, เมนู ทางหน้าจอ. ผู้ใช้จะใช้เมาส์หรือคีย์บอร์ดในการโต้ตอบกันระบบ.

ในระบบ X วินโดว์, ผู้ใช้จะใช้เชลล์ผ่านทาง*เทอร์มินอลเอมิวเลเตอร์ (terminal emulator)* ซึ่งเป็นโปรแกรม GUI ที่จำลองจอภาพเพื่อแสดงผลบนระบบ X วินโดว์. โปรแกรมเ

terminal emulator ▶
เทอร์มินอลเอมิวเลเตอร์. โปรแกรมที่จำลองหน้าจอแสดงผลตัวอักษรเพื่อใช้กับเชลล์. เทอร์มินอลเอมิวเลเตอร์จะเป็นโปรแกรมที่ใช้ใน X วินโดว์เช่น `xterm`, `gnome-terminal`,



รูปที่ 2.2: หน้าจอล็อกอินแบบกราฟฟิก



รูปที่ 2.3: เทอร์มินอลเอมูเลเตอร์ (`gnome-terminal`) ที่รันบน X วินโดว์

ทอร์มินอลเอมิวเลเตอร์มีหลายปรเภทเช่น `xterm`, `gnome-terminal`, `konsole` ขึ้นอยู่กับความชอบพอของผู้ใช้. โปรแกรมเหล่านี้จะเตรียมเชลล์เชิงโต้ตอบซึ่งจะรอแปรคำสั่งที่ป้อนเข้ามาทางคีย์บอร์ดต่อไป.

อินเทอร์เฟสแบบ CLI และ GUI มีจุดเด่นและจุดด้อยแตกต่างกันไป. ผู้ใช้ไม่ต้อง

ใช้เชลล์เลยก็ได้ถ้าล็อกอินเข้าทางกราฟฟิกโหมด. GUI สะดวกในการใช้เพราะเราเห็นทุกอย่างเป็นสิ่งที่จับต้องได้เช่นหน้าต่าง ๆ, ปุ่ม, เมนู. ในทางกลับกันข้อดีนี้ก็อาจจะกลายเป็นข้อด้อยได้เช่นกัน. เช่นเมนูของการกระทำที่โปรแกรมเตรียมไว้ไม่เพียงพอ, คอมพิวเตอร์ที่ใช้ไม่มีประสิทธิภาพเพียงพอที่จะรันโปรแกรมให้เร็ว, ไม่สามารถทำงานที่ต้องการโดยอัตโนมัติได้เพราะผู้ใช้ต้องโต้ตอบกับโปรแกรมนั้น ๆเสมอ. ข้อโดยที่ยกตัวอย่างไปนี้เป็นสิ่งที่เชลล์ถนัด. กล่าวคือเชลล์มีไม่ยึดติดกับงานอย่างใดอย่างหนึ่ง, เราสามารถใช้คำสั่งบรรทัดหลาย ๆคำสั่งรวมกันทำงานเฉพาะที่เราต้องการได้. ที่สำคัญคือเชลล์ทำงานได้ทั้งแบบเชิงโต้ตอบและแบบอัติโนมัติ. ถ้าเรารู้สิ่งขั้นตอนของงานที่เราต้องการทำก็สามารถเขียน*เชลล์สคริปต์ (shell script)* ให้ทำงานโดยอัตโนมัติได้. ส่วนข้อด้อยของเชลล์คือไม่สะดวกในการใช้สำหรับผู้ที่ไม่คุ้นเคย. และงานในหลายอย่างต้องการการสิ่งที่นอกเหนือจาก*ข้อมูลเท็กซ์ (text data)* เช่นการตัดต่อวีดีโอ, การดูข้อมูลทางเว็บ เป็นต้น.

shell script ▶
เชลล์สคริปต์. ไฟล์ที่รวมคำสั่งที่ใช้ในเชลล์เข้าด้วยกัน. เชลล์จะตีความคำสั่งที่อยู่ในไฟล์นั้นบรรทัดต่อบรรทัด. สะดวกสำหรับการสั่งคำที่เป็นขั้นตอน. เนื่องจากเชลล์มีไวยกรณ์, สามารถสร้างตัวแปร, สามารถควบคุมข้อแม้ต่าง ๆได้จึงถือเป็นภาษาคอมพิวเตอร์แบบ interpreter ด้วย. เชลล์สคริปต์มีบทบาทสำคัญกับลินุกซ์มากเช่น เชลล์สคริปต์ใช้ในการเริ่มต้นโปรแกรมเซิร์ฟเวอร์ต่าง ๆ.

2.2.3 การล็อกอินแบ่งตามการใช้เน็ตเวิร์ก

ล็อกอินผ่านคอนโซล

คอนโซล (console) หมายถึงชุดคีย์บอร์ดและจอแสดงผลที่ต่อโดยตรงกับคอมพิวเตอร์. การล็อกอินผ่านทางคอนโซลเป็นการล็อกอินแบบไม่ใช้เน็ตเวิร์ก, จะเป็นการล็อกอินแบบเท็กซ์โหมดหรือกราฟฟิกโหมดก็ได้แล้วแต่ความสามารถของคอมพิวเตอร์ในการแสดงผล.

ล็อกอินผ่านทางเน็ตเวิร์ก

การล็อกอินประเภทผ่านทางเน็ตเวิร์กทำได้โดยใช้โปรแกรมเช่น `telnet`, `ssh`, X ฯลฯ ติดต่อกับเครื่องคอมพิวเตอร์ที่ต้องการล็อกอิน. การล็อกอินผ่านทางเน็ตเวิร์กสามารถทำได้ทั้งแบบเท็กซ์โหมดและกราฟฟิกโหมดโดยใช้*เน็ตเวิร์กโปรโตคอล (network protocol)* ที่แตกต่างกัน. ตัวอย่างเช่นโปรแกรม `telnet` จะใช้โปรโตคอล telnet ในการสื่อสารระหว่างคอมพิวเตอร์, การล็อกอินแบบกราฟฟิกจะใช้โปรโตคอล xdmcp (X Disaplay Manager Control Protocol) เป็นต้น.

ที่ผมใช้คำว่า "ไม่สะดวกในการใช้สำหรับผู้ที่ไม่คุ้นเคย" แทนคำว่า "ไม่สะดวกในการใช้" เพราะเป็นความจริงที่ว่าความยากง่ายของการใช้อยู่ที่ความเคยชิน

text data ▶
ข้อมูลเท็กซ์. ข้อมูลที่หรือไฟล์ที่มนุษย์สามารถอ่านแล้วเข้าใจได้. โดยปรกติข้อมูลเท็กซ์จะหมายถึงไฟล์ที่เขียนด้วยภาษาอังกฤษที่อ่านได้, ไม่มีอักขระควบคุม (control character) ปะปน.

2.3 การออกจากระบบ

2.3.1 เท็กซ์โหมด

เมื่อต้องการออกจากระบบให้ใช้คำสั่ง "`logout`".

การออกจากระบบของระบบปฏิบัติการ Windows เรียกว่า logoff แต่ในโลกของยูนิกซ์จะเรียกว่า logout

`logout` อ้างอิงหน้า 411

ตัวอย่างที่ 2.8: การล็อกเอาท์แบบเท็กซ์โหมด

```
$ logout↵
↓ ระบบจะเคลียร์หน้าจอและรอการล็อกอินต่อไป
login: ▮
```

คำสั่ง `logout` นี้จะทำได้เมื่อล็อกอินเท่านั้น. เพราะว่าเชลล์ของเทอร์มินอลเอมูเลเตอร์ไม่ใช่ล็อกอินเชลล์จึงใช้คำสั่ง `logout` ไม่ได้, ให้ใช้คำสั่ง "`exit`" แทน. สิ่ง

`exit` อ้างอิงหน้า 421

ที่แตกต่างกันระหว่าง `logout` กับ `exit` คือ `logout` จะอ่านคำสั่งหรือค่าติดตั้งในไฟล์ `.bash_logout` ที่อยู่ในโฮมไดเรกทอรีก่อนจะจบการทำงานของเชลล์. ส่วนคำสั่ง `exit` จะจบการทำงานของเชลล์เฉย ๆ. โดยทั่วไปแล้วการกดคีย์ Ctrl+d ในเชลล์พรอมต์ก็เป็นการจบการทำงานของเชลล์ได้เช่นกัน.

Ctrl+d เป็นอักขระควบคุมเทอร์มินอลหมายถึงไม่มีข้อมูลนำเข้าอีกต่อไป (end of file). ซึ่งก็หมายถึงการจบการทำงานของเชลล์.

2.3.2 กราฟฟิกโหมด

ในระบบ Desktop Environment จะมีเมนูหลักสำหรับให้ผู้ใช้เลือกเพื่อล็อกเอาท์ออกจากระบบ.

2.4 เทอร์มินอลเสมือน

โดยปรกติคอมพิวเตอร์หนึ่งเครื่องจะมีคอนโซลหนึ่งชุด. เมื่อผู้ใช้คนหนึ่งล็อกอินผ่านทางคอนโซล, ผู้ใช้คนอื่นจะไม่สามารถใช้คอมพิวเตอร์ได้จนกว่าผู้ใช้ที่ใช้คอนโซลล็อกเอาท์. ถ้าต้องการล็อกอินเข้าสู่ระบบเดียวกันพร้อม ๆกัน, ผู้ใช้คนอื่นต้องล็อกอินผ่านทางเน็ตเวิร์ก. ลินุกซ์คอนโซลสามารถแก้ปัญหานี้ได้ด้วยสิ่งที่เรียกว่า*เทอร์มินอลเสมือน (virtual terminal).* เทอร์มินอลเสมือนจะจำลองคอนโซลให้มีหลายชุดในเวลาเดียวกัน. อย่างไรก็ตามเนื่องจากคอมพิวเตอร์หนึ่งเครื่องจะมีคีย์บอร์ด, เมาส์และจอแสดงผลเพียงหนึ่งชุดเท่านั้น, เวลาเปลี่ยนคอนโซลโดยใช้เทอร์มินอลเสมือนจะใช้การกดคีย์ Ctrl และ Alt ร่วมกับ*ฟังชันคีย์ (function key).*

virtual terminal ▶ *เทอร์มินอลเสมือน.* เทอร์มินอลในเท็กซ์โหมดของลินุกซ์ที่สามารถเปลี่ยนหน้าจอเป็นหลายหน้าจอได้โดยมีหน้าจอที่เป็นรูปธรรมเพียงหน้าจอเดียว. วิธีเปลี่ยนหน้าจอทำได้โดยกด Ctrl และ Alt ร่วมกับฟังก์ชันคีย์.

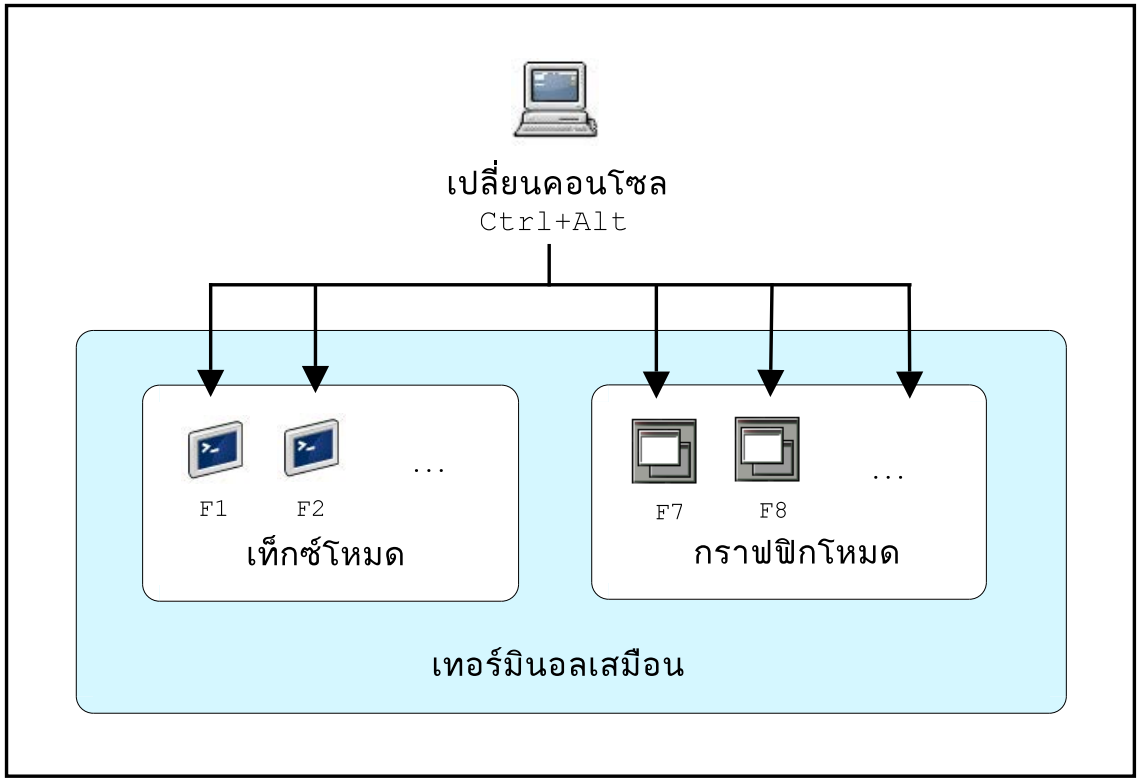
2.4.1 การเปลี่ยนโหมด

สมมติว่าผู้อ่านกำลังล็อกอินด้วยกราฟฟิกโหมดอยู่และต้องการเปลี่ยนเป็นเท็กซ์โหมด, ผู้อ่านสามารถทำได้โดยกดคีย์ Ctrl+Alt+F1 เพื่อเปลี่ยนเป็นเทอร์มินัลที่ 1 (ดูรูปที่ 2.4 ประกอบ). กดคีย์ Ctrl+Alt+F2 เพื่อเปลี่ยนเป็นเทอร์มินัลที่ 2 ซึ่งเป็นเท็กซ์โหมดเช่นกัน. ถ้าจะเปลี่ยนกลับเป็นกราฟฟิกโหมดเหมือนเดิมให้กดคีย์ Ctrl+Alt+F7.

Ctrl+Alt+F1 หมายความว่าให้กดคีย์ Ctrl และ Alt ค้างไว้แล้วกดคีย์ F1 ตาม. ในเท็กซ์โหมดไม่ต้องกดคีย์ Ctrl ก็ให้ผลเหมือนกัน.

โดยปรกติแล้ว X เซิร์ฟเวอร์จะรันอยู่ตัวเดียวเพราะฉะนั้นเทอร์มินอลที่เป็นกราฟฟิกโหมดจะเป็นเทอร์มินอลที่ 7 (F7) เท่านั้น. ถ้ามี X เซิร์ฟเวอร์รันอยู่มากกว่าหนึ่งตัว, X เซิร์ฟเวอร์ตัวที่สองจะรันอยู่ที่คอนโซลที่ 8 (F8) เป็นต้น.

ตัวอย่างการใช้เทอร์มินอลเสมือนเช่น นาย ก. ใช้คอมพิวเตอร์รันโปรแกรมคำนวณซึ่งกินเวลาเป็นชั่วโมงอยู่และล็อกอินค้างไว้. นาย ข. มีธุระด่วนต้องอ่านเมลล์และไม่มีคอมพิวเตอร์ตัวอื่นให้ล็อกอินผ่านทางเน็ตเวิร์ก. ถ้าเป็นคอมพิวเตอร์ที่ไม่ใช่ลินุกซ์ นาย ก. ต้องล็อกเอาท์เพื่อให้นาย ข. ใช้คอมพิวเตอร์. ในกรณีนี้นาย ก. ไม่จำเป็นต้องล็อกเอาท์ออกจากระบบ, เพียงแต่เปลี่ยนเป็นเทอร์มินัลเสมือนเป็นตัวอื่นให้นาย ข. ได้ใช้ชั่วคราวโดยที่ไม่มีผลกระทบกับโปรแกรมที่ตัวเองรันอยู่. เมื่อนาย ข. ใช้งานเสร็จก็เปลี่ยนเทอร์มินัลเสมือนกลับเป็นตัวเดิมให้นาย ก. ใช้อีกที. ตัวอย่างการใช้เทอร์มินอลเสมือนอื่น ๆเช่น การเปลี่ยนจากกราฟฟิกโหมดเป็นเท็กซ์โหมดเวลากราฟฟิกโหมดมีปัญหาไม่สามารถรับข้อมูลจากคีย์บอร์ดหรือเมาส์ได้.



รูปที่ 2.4: เทอร์มินอลเสมือน

2.5 เชลล์เบื้องต้น

ในปัจจุบันถึงแม้ว่าอินเทอร์เฟสแบบกราฟฟิกจะเป็นที่นิยมกันอย่างกว้างขวางแต่ไม่ได้หมายความว่าอินเทอร์เฟสแบบบรรทัดคำสั่งเป็นสิ่งล้าสมัย. เชลล์เป็นโปรแกรมแอินเทอร์เฟสที่โต้ตอบกับผู้ใช้คอมพิวเตอร์โดยอาศัยแป้นพิมพ์และการแสดงผลเป็นตัวอักษรเป็นหลัก. เนื่องจากความต้องการพื้นฐานของเชลล์ได้แก่แป้นพิมพ์และจอภาพที่สามารถแสดงตัวอักษร, ทำให้คอมพิวเตอร์ที่ใช้ไม่จำเป็นต้องมีทรัพยากรมากก็ใช้ได้. การแสดงผลเป็นตัวอักษร, ข้อความซึ่งเป็นภาษาที่มนุษย์เข้าใจได้ทันที. สำหรับผู้ที่คุ้นเคยกับแป้นพิมพ์สามารถสั่งคำสั่งได้รวดเร็วเมื่อเปรียบเทียบกับการใช้เมาส์. และที่สำคัญที่สุดคือผู้ใช้สามารถเขียนคำสั่งเป็นขั้นตอนแล้วดำเนินการทีเดียวได้แบบอัตโนมัติ. ด้วยเหตุผลเหล่านี้เองที่ผู้ใช้ใหม่ควรจะศึกษาการใช้เชลล์อย่างจริงจังเพื่อที่จะเป็นพื้นฐานในการใช้ลินุกซ์ต่อไป.

ในระบบปฏิบัติการยูนิกซ์ยุคแรกใช้เชลล์ที่เรียกกันว่า Bourne shell (`sh`) ซึ่งตั้งชื่อตามผู้สร้างคือนาย Stephen Bourne. หลังจากที่เกิด BSD ยูนิกซ์, Bill Joy ได้สร้างเชลล์ที่มีไวร์กรณ์คล้ายกับภาษาซีและมีความสามารถพิเศษมากขึ้นกว่าเดิมเรียกว่า C shell (`csh`). นอกจากนี้ยังมีเชลล์หลายตัวเกิดขึ้นเช่น Korn shell (`ksh`), Bourne-again shell (`bash`), Zsh (`zsh`) ฯลฯ. ในระบบปฏิบัติการลินุกซ์เลือกใช้ Bourne-again shell, หรือที่เรียกสั้น ๆว่า `bash` เป็นเชลล์ปริยาย. Bash เป็นโปรแกรมหนึ่งในโปรเจค GNU. Bash

หลายคนเข้าใจผิดว่าเชลล์คือหรือเหมือน DOS (Disk Operating System). ในความเป็นจริงแล้ว DOS คือระบบปฏิบัติการแต่เชลล์คือโปรแกรม. แต่ทั้งเชลล์และ DOS ใช้อินเทอร์เฟส CLI จึงทำให้คนทั่วไปคิดว่าเชลล์และ DOS เหมือนกัน.

การใช้เมาส์ถือว่าช้ากว่าการใช้แป้นพิมพ์เพราะผู้ใช้ต้องกระทำการหลายขั้นตอนได้แก่ เลื่อนเมาส์, เล็งจุดของเมาส์ให้ตรงกับตำแหน่งที่ต้องการ, กดเมาส์, เลือกเมนู ฯลฯ. ถ้าผู้ใช้วางตำแหน่งนิ้วได้ถูกต้องและจำตำแหน่งของคีย์ต่าง ๆได้จะเร็วมาก. ถ้าเทียบเวลาการเรียนรู้แล้ว, เมาส์ใช้เวลาเรียนรู้น้อยกว่าการเรียนใช้แป้นพิมพ์.

ในระบบลินุกซ์ `sh` จะเป็น symbolic link ของ `bash`.

"z" ใน zsh มีความหมายแฝงหมายถึงเชลล์ตัวสุดท้าย. zsh เป็นเชลล์ที่รวบรวมสิ่งดี ๆของเชลล์ต่าง ๆรวมกันและมีจุดมุ่งหมายเป็นเชลล์ที่ดีที่สุดในบรรดาเชลล์ทั้งหลาย.

สร้างขึ้นโดย Brain Fox และ Chet Ramey ให้มีความเข้ากันได้กับ Bourne เชลล์มีจุดเด่นในการแก้ไขบรรทัดคำสั่ง, การโต้ตอบกับผู้ใช้, การเติมเต็มคำสั่งหรือไฟล์ ฯลฯ.

สำหรับระบบปฏิบัติการตระกูลยูนิกซ์, ผู้ใช้นิยมที่จะสั่งคำสั่งให้คอมพิวเตอร์หรือเคอร์เนลทำงานโดยผ่านเชลล์. สาเหตุที่เรียกว่าเชลล์เพราะเชลล์ทำหน้าที่เป็นเปลือกหุ้มเป็นตัวกลางระหว่างเคอร์เนลและผู้ใช้. ในทางทฤษฎี, เชลล์เป็น*โปรแกรมแปรคำสั่ง (command interpreter)* อย่างหนึ่งที่มีคุณสมบัติ*เชิงโต้ตอบ (interactive)* กับผู้ใช้, มีความสามารถควบคุมการดำเนินงาน, รับตัวแปรซึ่งอาจจะเป็นค่าที่พิมพ์จากคีย์บอร์ดหรือชื่อไฟล์ส่งต่อให้โปรแกรม, เรียกกระทำการโปรแกรมต่าง ๆ, รับและส่งข้อมูลให้กับโปรแกรมที่เรียกใช้เป็นต้น.

command ▶
คำสั่ง. คำหรือตัวอักษรที่พิมพ์ทางแป้นพิมพ์ส่งต่อให้เชลล์แปลความหมายและกระทำการต่อไป. ความหมายทั่วไปยังหมายถึงโปรแกรมหรือชื่อโปรแกรมที่เรียกใช้ผ่านเชลล์.

คำสั่ง (command) ที่เชลล์แปรความในที่นี้อาจจะหมายถึงคำสั่งที่เชลล์สามารถกระทำการเองได้ซึ่งเรียกว่า*คำสั่งภายใน (shell built-in command)* หรือคำสั่งที่เชลล์ไม่สามารถกระทำการได้เองเรียกว่า*คำสั่งภายนอก (external shell command)* ซึ่งก็คือ*โปรแกรม (program)* นั่นเอง.

ตัวอย่างคำสั่งภายในเช่น `cd`, `pwd`, `fg` เป็นต้น. ตัวอย่างคำสั่งภายนอกหรือโปรแกรมได้แก่คำสั่ง `ls`, `cp`, `rm` ฯลฯ.

2.5.1 อักขระ

อักขระ (character) ที่พิมพ์ในเชลล์พรอมต์โดยปรกติได้แก่อักษรภาษาอังกฤษที่เรียกว่าอักขระ ASCII (American Standard Code for Information Interchange) ซึ่งแสดงในตารางที่ ก.1 (หน้า 377). อักขระ ASCII ยังแบ่งได้เป็น*อักขระควบคุม (control character)* สำหรับควบคุมการประมวลผลข้อมูลหรือควบคุมการส่งข้อมูล, และ*อักขระกราฟฟิก (graphic character)* ซึ่งเป็นอักขระที่พิมพ์แล้วมองเห็นได้ซึ่งได้แก่อักษรภาษาอังกฤษรวมถึงเครื่องหมายต่าง ๆที่ใช้ทั่วไป.

อักขระหมายถึงตัวอักษรรวมถึงเครื่องหมายต่าง ๆ.

เนื่องจากอักขระควบคุมเหล่านี้ไม่มีรูปร่างหน้าตา, บางครั้งจึงเรียกว่า non-printable character.

โดยทั่วไป, ผู้ใช้สามารถพิมพ์อักขระกราฟฟิกได้ด้วยคีย์บอร์ด. แต่อักขระควบคุมที่ใช้บ่อยบางตัวเท่านั้นที่สามารถพิมพ์จากคีย์บอร์ดได้โดยตรงเช่น Backspace Tab Enter เป็นต้น. ในกรณีที่คีย์บอร์ดที่ใช้ไม่มีคีย์พิเศษดังกล่าว, เราสามารถกดปุ่ม Ctrl ค้างไว้แล้วกดคีย์ตัวอักษรภาษาอังกฤษตามเพื่อสร้างอักขระควบคุมได้. ตัวอย่างเช่น Ctrl+h ใช้แทน Backspace, Ctrl+i ใช้แทน Tab และ Ctrl+m ใช้แทน Enter เป็นต้น.

อักขระควบคุมบางตัวมีความหมายพิเศษต่อเทอร์มินอลหรือเชลล์. ตัวอย่างเช่น อักขระ EOT (End Of Transmission) หมายถึงสิ้นสุดข้อมูลนำเข้า. ผู้ใช้สามารถส่งอักขระควบคุมนี้โดยกดคีย์ Ctrl+d ในเชลล์พรอมต์เพื่อจบการทำงานของเชลล์. การกดคีย์ Ctrl+l จะหมายถึง FF (Form Feed) ซึ่งมีความหมายถึงการขึ้นหน้าใหม่. ในกรณีนี้สิ่งที่แสดงบนหน้าจอไปแล้วจะถูกลบทิ้ง (เคลียร์) แล้วจะเหลือเชลล์พรอมต์ไว้ที่บรรทัดเริ่มต้น. ตารางที่ 2.1 แสดงอักขระควบคุมที่ใช้บ่อย.

บ้างก็เรียก EOT ว่า EOF (End Of File).

ตารางที่ 2.1: อักขระควบคุมที่มีความหมายพิเศษต่อเทอร์มินอลหรือเชลล์

อักขระควบคุม	คีย์	คำอธิบาย
ETX	Ctrl+c	สัญญาณ SIGINT (interrupt) ยกเลิกการทำงานของคำสั่ง
EOT	Ctrl+d	สิ้นสุดข้อมูลนำเข้า, จบการทำงานในกรณีที่ใช้อักขระควบคุมนี้เป็นตัวแรกในเชลล์พรอมต์
		ลบตัวอักษรที่อยู่ตรงเคอร์เซอร์ในกรณีที่พิมพ์คำสั่งอยู่, ใช้แทน Del ได้
DC1	Ctrl+q	แสดงผลทางหน้าจออีกครั้งหลังจากที่ถูกหยุดด้วยอักขระควบคุม DC3
DC3	Ctrl+s	หยุดการแสดงผลทางหน้าจอชั่วคราว (ขึ้นอยู่กับเทอร์มินอลที่ใช้)
SUB	Ctrl+z	สัญญาณ SIGSTOP (stop) หยุดการทำงานของคำสั่ง(ชั่วคราว)

2.5.2 คำสั่ง

คำสั่งคือสิ่งที่ต้องการให้คอมพิวเตอร์ทำงาน. ถ้าคอมพิวเตอร์เข้าใจภาษามนุษย์, เราอาจจะสั่งคำสั่งได้ดังต่อไปนี้.

ในสภาพแวดล้อมที่กำหนด ให้ทำสิ่งที่สั่ง(ทำอะไร) แบบนี้(ทำอย่างไร) โดยให้ใช้ข้อมูลต่อไปนี้

การสั่งคำสั่งในเชลล์พรอมต์ก็มีลักษณะคล้ายกับการสั่งคำสั่งด้วยวาจาโดยมีรูปแบบการสั่งคำสั่งทั่วไปเป็น

```
$ ตัวแปรสภาพแวดล้อม=ค่า␣ตัวแปรสภาพแวดล้อม=ค่า␣...␣คำสั่ง␣อาร์กิวเมนต์␣อาร์กิวเมนต์␣... ↵
```

เครื่องหมาย ↵ ใช้แสดงถึงการกดคีย์ Enter และเครื่องหมาย ␣ แสดงถึงช่องไฟ.

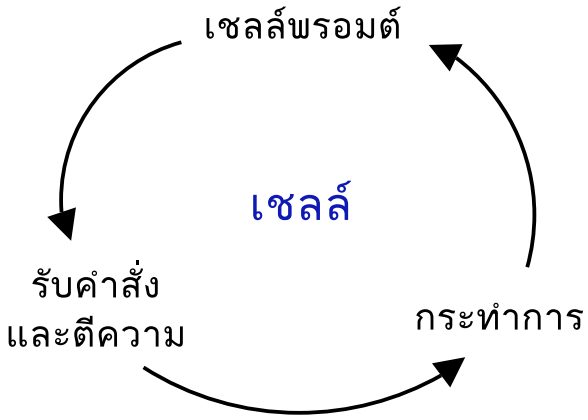
ส่วนที่จำเป็นในการสั่งคำสั่งได้แก่ชื่อคำสั่ง. ส่วนที่เป็น "*ตัวแปรสภาพแวดล้อม=ค่า*" และ "*อาร์กิวเมนต์*" จะมีหรือไม่มีก็ได้.

คำสั่งโดยปรกติจะเป็น "คำ" ภาษาอังกฤษ. คำในที่นี้หมายถึงการนำอักษรภาษาอังกฤษมาเรียงกันซึ่งอาจจะไม่มีความหมายก็ได้. คำในที่นี้ยังหมายถึงกลุ่มอักษรที่แบ่งได้ด้วยช่องไฟหรือแคร่เช่น `abc def` จะถือว่ามีสองคำคือ `abc` กับ `def`. เชลล์ยังแยกแยะความแตกต่างระหว่างตัวอักษรภาษาอังกฤษตัวเล็ก (small letter) และตัวใหญ่ (capital letter) ด้วย.

ชื่ออีกแบบของตัวอักษรภาษาอังกฤษตัวเล็กคือ lower-case letter. และชื่อของตัวอักษรภาษาอังกฤษตัวใหญ่คือ upper-case letter.

เมื่อผู้ใช้กดคีย์ Enter แล้ว, เชลล์จะตรวจสอบว่าคำที่ใส่เข้าไปนั้นเป็นอะไร, ถ้าเชลล์สามารถตีความได้ก็จะทำงานตามที่สั่งต่อไป. เมื่องานที่สั่งไปจบเรียบร้อยแล้วเชลล์ก็จะแสดงเชลล์พรอมต์เพื่อที่จะรับคำสั่งถัดไปเรื่อย ๆเป็นวัฏจักรตามรูปที่ 2.5.

สิ่งที่อยู่หน้าคำสั่ง (มีหรือไม่ก็ได้) ได้แก่*ตัวแปรสภาพแวดล้อม (environment variable)* และค่าของตัวแปรนั้น ๆ. ให้สังเกตว่าตัวแปรและค่าของตัวแปรขั้นด้วยเครื่องหมาย



รูปที่ 2.5: วงจรการรันคำสั่งของเชลล์

เท่ากับโดยที่ข้างหน้าและข้างหลังเครื่องหมายเท่ากับไม่มีช่องไฟ. โดยทั่วไปเวลาสั่งคำสั่งมักจะไม่ระบุตัวแปรสภาพแวดล้อมและค่าแบบนี้, ยกเว้นในกรณีที่ต้องการระบุตัวแปรและค่าของตัวแปรให้คำสั่งที่ต้องการสั่งรับรู้เฉพาะครั้ง (ตัวอย่างที่ 2.16).

สิ่งที่พิมพ์ถัดจากคำสั่งเรียกว่า*อาร์กิวเมนต์ (argument)* อาจจะเป็น*ตัวเลือก (option)*, ตัวแปรหรือค่าที่ต้องการส่งให้คำสั่ง, ชื่อไฟล์เป็นต้น. คำสั่งและอาร์กิวเมนต์จะแบ่งด้วย*ช่องไฟ (space)* หรือ*จุดตั้งระยะ (tab)*. หลังจากที่พิมพ์คำสั่งที่ต้องการแล้วให้กดคีย์ Enter กระทำการ.

ตัวอย่างต่อไปนี้แสดงการสั่งคำสั่งต่าง ๆในเชลล์.

ตัวอย่างที่ 2.9: การสั่งคำสั่งผ่านทางเชลล์

```
$ pwd↵    ← การสั่งคำสั่งเดี่ยวๆ
/home/somchai
$ ls↵                                                    ← การสั่งคำสั่งเดี่ยวๆ
Desktop  file1
$ /bin/ls↵                      ← สั่งคำสั่งโดยระบุที่อยู่ของโปรแกรมทั้งหมด (full path)
Desktop  file1
$ ../../bin/ls↵                 ← สั่งคำสั่งโดยระบุที่อยู่ของโปรแกรมเทียบกับตำแหน่งปัจจุบัน
$ ls -l↵                                  ← การสั่งคำสั่งกับอาร์กิวเมนต์ (ตัวเลือก)
total 4
drwxr-xr-x    2 somchai  somchai      4096 Mar 17 22:36 Desktop
-rw-rw-r--    1 somchai  somchai         0 Apr  4 23:06 file1
$ LANG=th_TH ls -l↵                         ← การระบุตัวแปรสภาพแวดล้อมเฉพาะครั้ง
total 4
drwxr-xr-x    2 somchai  somchai      4096 มี.ค. 17 22:36 Desktop
-rw-rw-r--    1 somchai  somchai         0 เม.ย.  4 23:06 file1
$ echo $PATH↵
/usr/bin:/bin:/usr/X11R6/bin
$
```

การสั่งคำสั่งทำได้โดยพิมพ์ชื่อคำสั่งเฉย ๆเช่น “ls” ก็ได้หรือระบุไดเรกทอรี (ที่อยู่) ของคำสั่งที่ต้องการสั่งด้วยเช่น “/bin/ls” หรือ “../../bin/ls”. ในกรณีที่พิมพ์แค่ชื่อคำสั่งอย่างเดียว, เชลล์จะค้นหาตัวโปรแกรม (คำสั่ง) จาก*ตัวแปรสภาพแวดล้อม*

(environment variable) ที่ชื่อ PATH. ค่าของตัวแปรสภาพแวดล้อม PATH เป็นชื่อไดเรกทอรีต่าง ๆที่มีโปรแกรมอยู่ขั้นด้วยตัวอักษร ":". เมื่อสั่งคำสั่งโดยไม่ระบุไดเรกทอรีเช่น ls แล้ว, เชลล์จะตรวจสอบว่า ls เป็นคำสั่งภายในหรือไม่. ถ้าไม่ได้เป็นคำสั่งภายในก็จะหาตัวโปรแกรม ls จากไดเรกทอรี /usr/bin, /bin และ /usr/X11R6/bin ตามลำดับ.

การสั่งคำสั่งโดยระบุไดเรกทอรีสามารถแบ่งออกเป็น 2 แบบคือการระบุไดเรกทอรีของไฟล์หรือโปรแกรมที่ต้องการใช้แบบ *full path* และแบบ *relative path*. ตัวอย่างของการระบุโปรแกรมแบบ full path ได้แก่ /bin/ls เป็นการระบุไดเรกทอรีทั้งหมดตั้งแต่*รูทไดเรกทอรี (root directory)* (/), bin จนถึงตัวโปรแกรมซึ่งก็คือไฟล์ไบนารีตามลำดับ. ส่วนการระบุที่อยู่ของโปรแกรมแบบ relative path จะอ้างอิงจากไดเรกทอรีที่ทำงานอยู่โดยใช้จุด "." แทนไดเรกทอรีที่ทำงานอยู่และใช้ ".." แทนไดเรกทอรีที่อยู่เหนือไดเรกทอรีปัจจุบันขึ้นไปหนึ่งขั้น. ดังนั้นจากตัวอย่างที่ 2.9, ../../bin/ls จึงเป็นการระบุโปรแกรม /bin/ls เพราะไดเรกทอรีที่ทำงานอยู่ตอนนั้นคือ /home/somchai.

สมมติว่าเรามีโปรแกรมอยู่ในไดเรกทอรีที่ทำงานอยู่ชื่อ a.out และต้องการจะรันโปรแกรมนี้. เราต้องระบุด้วยว่า a.out อยู่ที่ไหนโดยใช้ relative path ตามตัวอย่างต่อไปนี้

a.out เป็นชื่อไฟล์ผลลัพธ์ที่ได้จากการคอมไพล์โปรแกรม. ว่ากันว่าชื่อนี้ตั้งโดย Dennis Ritchie ผู้เขียนภาษา C หมายถึง "the output of the assembler" จึงให้ชื่อไฟล์ที่สร้างจากคอมไพล์รหัสต้นฉบับว่า a.out.

ตัวอย่างที่ 2.10: การรันโปรแกรมที่อยู่ในไดเรกทอรีที่ทำงานอยู่

```
$ ls -l a.out↵
-rwxr-xr-x    1 somchai  users          28 Apr 28 20:48 a.out
$ a.out↵
-bash: a.out: command not found
$ ./a.out↵
Hello world.
```

ถ้าไม่ต้องการพิมพ์ "./" บ่อย ๆก็ให้ใส่จุด (.) ซึ่งหมายถึงไดเรกทอรีที่ทำงานอยู่ไว้ในตัวแปรสภาพแวดล้อม PATH ด้วย.

2.5.3 ตรวจสอบประเภทของคำสั่ง

ตามที่ได้แนะนำไปแล้วว่าคำสั่งที่ใช้กันนั้นแบ่งออกเป็น 2 ประเภทใหญ่ ๆได้แก่คำสั่งประกอบภายในและโปรแกรมคำสั่ง. ในเชลล์ bash จะมีคำสั่งประกอบภายใน type ใช้สำหรับตรวจสอบดูว่าคำสั่งที่พิมพ์ไปนั้นเป็นคำสั่งประเภทใด.

type อ้างอิงหน้า 417

ตัวอย่างที่ 2.11: ตรวจสอบประเภทคำสั่ง.

```
$ type cd↵
cd is a shell builtin
$ type ls↵
ls is aliased to 'ls --color=auto -F'
$ type /bin/ls↵
/bin/ls is /bin/ls
```

จากตัวอย่างทำให้เรารู้ว่าคำสั่ง cd เป็นคำสั่งประกอบภายในเชลล์, คือเชลล์สามารถทำงานได้ทันทีไม่ต้องอ่านไฟล์ไบนารีจากฮาร์ดดิกส์แล้วกระทำการ. นอกจากนั้นทำให้เรารู้ว่าถ้าสั่งคำสั่ง ls เฉย ๆเชลล์จะหมายถึง "ls --color=auto -F", แต่ถ้าสั่งคำสั่งเต็ม ๆ /bin/ls จะเป็นการโหลดโปรแกรมคำสั่ง /bin/ls มากระทำโดยตรงโดยไม่มีการให้ตัวเลือกใด ๆ.

ดูเรื่องเกี่ยวกับ alias ได้ที่หน้า 51.

2.5.4 ตัวแปรสภาพแวดล้อม

ตัวแปรสภาพแวดล้อม (environment varible) เป็น*ตัวแปรเชลล์ (shell variable)* ที่ใช้เพื่อเก็บค่าหรือข้อมูลบางอย่างเพื่อใช้ในเชลล์เองและมีผลต่อโปรแกรมที่เรียกใช้จากเชลล์นั้นด้วย. ตัวอย่างเช่นตัวแปรสภาพแวดล้อม PATH เป็นตัวแปรที่เก็บค่าของไดเรกทอรีที่มีโปรแกรมต่าง ๆและเชลล์จะค้นหาคำสั่งจากไดเรกทอรีเหล่านี้เมื่อผู้ใช้พิมพ์ชื่อคำสั่งในเชลล์พรอมต์.

การกำหนดตัวแปรสภาพแวดล้อมและค่าของตัวแปรทำได้โดยใช้เครื่องหมาย "=" และใช้คำสั่ง export ประกาศให้ตัวแปรนั้นเป็นตัวแปรสภาพแวดล้อม. เมื่อต้องการแสดงค่าของตัวแปรนั้น ๆจะใช้เครื่องหมายดอลลาร์ ($) นำหน้าชื่อตัวแปรเป็นการอ้างอิงค่า.

export อ้างอิงหน้า 421

เพื่อความชัดเจนในการแสดงค่าของตัวแปร, บางครั้งอาจจะใช้เครื่องหมายวงเล็บปีกกา ({}) กำกับด้วย เช่น ${PATH}

ตัวอย่างที่ 2.12: การตั้งค่าตัวแปรสภาพแวดล้อม

```
$ PATH=/bin:/usr/bin:/usr/local/bin↵
$ export PATH↵
$ echo $PATH↵
/bin:/usr/bin:/usr/local/bin
```

บรรทัด PATH=/bin:/usr/bin:/usr/local/bin เป็นการกำหนดและตั้งค่าตัวแปร. ส่วนคำสั่ง export จะทำให้ตัวแปรที่กำหนดนั้นเป็นตัวแปรสภาพแวดล้อม. ถ้าผู้ใช้ไม่สั่งคำสั่ง export ตัวแปรที่ตั้งไปจะถือเป็นตัวแปรธรรมดาในเชลล์. ความแตกต่างระหว่างตัวแปรสภาพแวดล้อมกับตัวแปรเชลล์คือโปรแกรมหรือคำสั่งที่กระทำการอยู่ในเชลล์นั้น ๆ จะรับรู้ (สืบทอด) ชื่อตัวแปรสภาพแวดล้อมและค่าของมันด้วย. ในทางกลับกัน, โปรแกรมหรือคำสั่งที่กระทำอยู่ในเชลล์นั้นจะไม่รู้จักรับรู้ตัวแปรเชลล์อื่น ๆที่ไม่ใช้ตัวแปรสภาพล้อม (สืบทอดไปสู่โปรแกรมที่สั่งในเชลล์นั้นไม่ได้). ผู้ใช้สามารถตั้งค่าตัวแปรสภาพแวดล้อมได้ภายในหนึ่งบรรทัดดังนี้.

ตัวอย่างที่ 2.13: การตั้งค่าตัวแปรสภาพแวดล้อมภายในบรรทัดเดียว

```
$ export PATH=/bin:/usr/bin:/usr/local/bin↵
```

ถ้าสั่งคำสั่ง export อย่างเดียวจะเป็นการแสดงตัวแปรสภาพแวดล้อมที่มีอยู่ในเชลล์นั้น.

โดยปรกติ, ตัวระบบจะตั้งค่า PATH โดยปริยายไว้ให้อยู่แล้ว. ให้ระวังเวลาตั้งค่า PATH ใหม่ด้วยตัวเองเพราะอาจจะทำให้ลบค่า PATH ที่มีอยู่แล้วทำให้เชลล์หาโปรแกรมบางอย่างไม่เจอ. ในกรณีที่ต้องการเพิ่มไดเรกทอรีของคำสั่งลงในตัวแปรสภาพแวดล้อม PATH ควรจะทำตามตัวอย่างต่อไปนี้เพื่อรักษรค่าเดิมของตัวแปร. กล่าวคือพิมพ์ไดเรกทอรีที่ต้องการเพิ่มเท่านั้น, ไม่ต้องได้เรกทอรีเองทั้งหมด.

ตัวอย่างที่ 2.14: การเพิ่มค่าให้ตัวแปรสภาพแวดล้อม PATH

```
$ echo $PATH↵
/usr/bin:/bin:/usr/X11R6/bin
$ export PATH=$PATH:/home/somchai/bin↵
$ echo $PATH↵
/usr/bin:/bin:/usr/X11R6/bin:/home/somchai/bin
```

เมื่อผู้ใช้รันโปรแกรมใด ๆก็ตามในเชลล์, โปรแกรมที่รันนั้นจะสืบทอดและรับรู้ตัวแปรสภาพสภาพแวดล้อมจากเชลล์ที่รันคำสั่งนั้นด้วย. ทำให้การรันคำสั่งอย่างเดียวกันในเชลล์ที่มีตัวแปรสภาพแวดล้อมต่างกันมีผลต่างกัน. ตัวอย่างเช่นตัวแปรสภาพแวดล้อม "LANG" จะเป็นตัวบอกสภาพแวดล้อมของภาษาที่ใช้อยู่. มีผลทำให้การแสดงต่อโปรแกรมที่รู้จักตัวแปรสภาพแวดล้อมนี้. ตัวอย่างเช่น.

ตัวอย่างใช้เทอร์มินอลที่แสดงผลภาษาไทยได้

date อ้างอิงหน้า 411

ตัวอย่างที่ 2.15: ผลกระทบของค่าตัวแปรสภาพแวดล้อมต่อคำสั่งที่ใช้

```
$ export LANG=en_US↵
$ date↵                                          ← สภาพแวดล้อมภาษาอังกฤษ
Wed Oct  8 01:15:31 JST 2003
$ export LANG=th_TH.TIS-620↵
$ date↵                                          ← สภาพแวดล้อมภาษาไทย
พ. 8 ต.ค. 2546 01:16:23 JST
```

Bash เชลล์มีคุณสมบัติพิเศษที่สามารถเจาะจงสภาพแวดล้อมเฉพาะคำสั่งที่ต้องการได้. ตัวอย่างต่อไปนี้เป็นการแสดงวันเดือนปีเป็นภาษาไทยในขณะที่สภาพแวดล้อมของเชลล์ที่สั่งคำสั่งนั้นเป็นภาษาอังกฤษ.

ตัวอย่างที่ 2.16: ผลกระทบของค่าตัวแปรสภาพแวดล้อมต่อคำสั่งที่ใช้เฉพาะครั้ง

```
$ export LANG=C↵
$ date↵
Sun Oct 12 13:57:23 JST 2003
$ LANG=th_TH.TIS-620 date↵                       ← ระบุสภาพแวดล้อมภาษาไทยเฉพาะครั้ง
อา. 12 ต.ค. 2546 13:58:00 JST
$ date↵
Sun Oct 12 13:59:05 JST 2003
```

จากตัวอย่างดังกล่าวจะเห็นได้ว่าการพิมพ์ LANG=th_TH.TIS-620 หน้าคำสั่งที่สั่งจะให้ผลเฉพาะคำสั่งนั้น ๆ. วิธีการดังกล่าวมีประโยชน์เมื่อต้องการระบุสภาพแวดล้อมชั่วคราวให้คำสั่งที่ต้องการ.

ตารางที่ 2.2: ตัวแปรสภาพแวดล้อมทั่วไป

ตัวแปรสภาพแวดล้อม	คำอธิบาย
LANG	สภาพแวดล้อมของภาษาที่ใช้
DISPLAY	display ที่ใช้แสดงผลของ X window

ต่อหน้าถัดไป

ต่อจากหน้าที่แล้ว

ตัวแปรสภาพแวดล้อม	คำอธิบาย
HOME	โฮมไดเรกทอรี
HOSTNAME	ชื่อโฮส์ต
PATH	รายการไดเรกทอรีที่เชลล์ใช้หาคำสั่ง
TERM	ประเภทของเทอร์มินอลที่ใช้งานอยู่

printenv อ้างอิงหน้า 414

ถ้าต้องการดูตัวแปรสภาพแวดล้อมทั้งหมดทำได้โดยใช้คำสั่ง `export -p` หรือคำสั่ง `printenv`. การตรวจสอบดูตัวแปรสภาพแวดล้อมทั้งหมดมีประโยชน์เป็นข้อมูลประกอบเพื่อ debug โปรแกรมหรือระบบ. เพราะโปรแกรมบางตัวอาจจะได้รับผลกระทบจากการตั้งค่าตัวแปรสภาพแวดล้อมบางตัวโดยที่ไม่ได้ตั้งใจ.

2.5.5 คำสั่งไหน

เวลาเราสั่งคำสั่งด้วยการเขียนชื่อคำสั่งในพรอมต์. เชลล์จะแปลความหมายว่าคำสั่งนั้นเป็นคำสั่งแบบไหน. ถ้าเป็นโปรแกรม, เชลล์จะหาว่าโปรแกรมนั้นอยู่ที่ไหนจากตัวแปรสภาพแวดล้อม PATH. ผู้ใช้ไม่มีความจำเป็นต้องรู้ว่าคำสั่งที่สั่งนั้นอยู่ที่ไหน. บางครั้งในระบบอาจจะมีโปรแกรมตัวเดียวกันแต่อยู่คนละไดเรกทอรี. เวลาเรียกสั่งคำสั่งที่เป็นโปรแกรมนั้น, เชลล์จะเลือกรันโปรแกรมที่เจอตัวแรกในไดเรกทอรีที่กำหนดในสภาพแวดล้อม PATH. ถ้าต้องการตรวจสอบว่าคำสั่งที่ใช้นั้นจริง ๆแล้วเป็นโปรแกรมอยู่ที่ไหนให้ใช้คำสั่ง `which`.

which อ้างอิงหน้า 418

ตัวอย่างที่ 2.17: แสดง full path ของคำสั่ง.

```
$ which -a ls↵
/bin/ls
/usr/bin/ls
```

ถ้าสั่งคำสั่ง `which` โดยไม่มีตัวเลือก, จะแสดง full path ของคำสั่งที่เจอตัวแรกจากตัวแปรสภาพแวดล้อม PATH. ตัวเลือก `-a` จะแสดง full path ของคำสั่งทุกตัวที่เจอในตัวแปรสภาพแวดล้อม PATH. จากตัวอย่างจะเห็นว่าคำสั่ง `ls` อยู่ในไดเรกทอรี `/bin`. ถ้าสั่งคำสั่ง `ls` ก็จะหมายถึง `/bin/ls`.

-a เป็นตัวเลือกสั้นของ --all.

whereis อ้างอิงหน้า 418

คำสั่งที่คล้ายกับคำสั่ง `which` คือ `whereis`. คำสั่ง `whereis` จะแสดงไฟล์ที่เกี่ยวข้องกับคำสั่งที่เป็นอาร์กิวเมนต์ได้แก่ไฟล์ไบนารี (โปรแกรม), ไฟล์คู่มือใช้งาน, และไฟล์รหัสต้นฉบับ.

ตัวอย่างที่ 2.18: แสดงไฟล์ที่เกี่ยวข้องกับคำสั่ง.

```
$ whereis ls↵
ls: /bin/ls /usr/bin/ls /usr/man/man1/ls.1.gz /usr/man/man1p/ls.1p.gz /usr/share
/man/man1/ls.1.gz /usr/share/man/man1p/ls.1p.gz
```

2.5.6 ตัวเลือก

ตัวเลือกของคำสั่งในระบบปฏิบัติการตระกูลยูนิกซ์ไม่มีกฎตายตัว. แต่ตัวเลือกของคำสั่งเหล่านี้มักจะมีรูปแบบที่คล้าย ๆกันโดยจะใช้เครื่องหมาย "-" (hyphen) เป็นการบ่งบอกถึงการใช้ตัวเลือก. โดยจะพิมพ์ชื่อตัวเลือกข้างหลังเครื่องหมาย -. ชื่อตัวเลือกได้แก่ อักษรหรือคำ. บางกรณีมีการส่งค่าหรือชื่อไฟล์ตามตัวเลือกที่ระบุด้วย. การระบุชื่อตัวเลือกให้คำสั่งมีรูปแบบต่าง ๆดังนี้.

ตัวเลือกเดี่ยว

ตัวเลือกเดี่ยวได้แก่การใช้อักษรตัวเดียวตามหลังเครื่องหมาย "-". ตัวอย่างเช่น

ตัวอย่างที่ 2.19: การใช้คำสั่งกับตัวเลือก

```
$ ls -a↵
.   .bash_logout   .bashrc  .emacs  .kde     .zshrc  file01
..  .bash_profile  .canna   .gtkrc  .xemacs  dir01
```

ผู้ใช้อาจจะใช้ตัวเลือกมากกว่าหนึ่งอย่างได้เช่น

ตัวอย่างที่ 2.20: การใช้คำสั่งกับตัวเลือกหลายตัว

```
$ ls -a -l↵
total 52
drwx------    5 somchai  somchai      4096 Sep 22 22:21 .
drwxr-xr-x    4 root     root         4096 Sep 22 22:15 ..
-rw-r--r--    1 somchai  somchai        24 Sep 22 22:15 .bash_logout
-rw-r--r--    1 somchai  somchai       191 Sep 22 22:15 .bash_profile
...
```

ผู้ใช้อาจจะสลับลำดับของตัวเลือกก็ได้ซึ่งขึ้นอยู่กับโปรแกรมที่ใช้. สำหรับโปรแกรม `ls`, "`ls -a -l`" จะมีผลลัพธ์เหมือนกับ "`ls -l -a`".

ตัวเลือกผสม

ตัวเลือกผสมเป็นการรวมเอาตัวเลือกเดี่ยวหลายตัวสั่งเป็นตัวเลือกทีเดียว. เช่นการรวมตัวเลือก `-a` กับ `-l` เป็น "`-al`" หรือ "`-la`".

ตัวอย่างที่ 2.21: การใช้คำสั่งกับตัวเลือกหลายตัวในทีเดียว

```
$ ls -al↵
total 52
drwx------    5 somchai  somchai      4096 Sep 22 22:21 .
drwxr-xr-x    4 root     root         4096 Sep 22 22:15 ..
-rw-r--r--    1 somchai  somchai        24 Sep 22 22:15 .bash_logout
-rw-r--r--    1 somchai  somchai       191 Sep 22 22:15 .bash_profile
...
```

ตัวเลือกแบบยาว

ตัวเลือกแบบยาวมักจะเป็นคำที่มีความหมายตามหลังเครื่องหมาย “--” (hyphen สองตัว). เช่นตัวเลือก “--all” เป็นตัวเลือกแบบยาวของตัวเลือก “-a”.

ตัวอย่างที่ 2.22: ตัวเลือกแบบยาว

```
$ ls --all↵
.   .bash_logout   .bashrc  .emacs  .kde     .zshrc  file01
..  .bash_profile  .canna   .gtkrc  .xemacs  dir01
```

ตัวเลือกแบบยาวเป็นตัวเลือกที่สื่อความหมายทำให้บางครั้งอาจจะจำได้ง่ายกว่าตัวเลือกเดี่ยว.

สำหรับคำสั่งบางคำสั่ง, ชื่อตัวเลือกแบบยาวอาจจะตามหลังเครื่องหมาย “-” (hyphen ตัวเดียว) ก็ได้. เช่นตัวเลือกที่ใช้กับคำสั่ง find.

find อ้างอิงหน้า 396

ตัวอย่างที่ 2.23: ตัวเลือกแบบสื่อความหมาย

```
$ find /usr -name thai↵
find: /usr/share/ssl/CA: Permission denied
/usr/share/texmf/fonts/tfm/public/thai
/usr/share/texmf/fonts/type1/public/thai
/usr/share/texmf/fonts/vf/public/thai
/usr/share/texmf/fonts/afm/public/thai
```

ในกรณีนี้ “-name” เป็นตัวเลือกตัวเดียวไม่ใช่ -n, -a, -m และ -e แยกกัน.

ชื่อตัวเลือกที่ไม่ต้องมีเครื่องหมาย - นำหน้า

ps อ้างอิงหน้า 403

การระบุตัวเลือกแบบนี้ไม่ต้องพิมพ์เครื่องหมาย - นำหน้าชื่อตัวเลือก. ตัวอย่างเช่น

ตัวอย่างที่ 2.24: ตัวเลือกที่ไม่มีเครื่องหมาย - นำ

```
$ ps aux↵
```

เป็นการสั่งคำสั่ง ps ประกอบกับตัวเลือกสามตัวคือ a, u และ x พร้อม ๆกัน.

การส่งค่าประกอบกับตัวเลือก

การใช้ตัวเลือกบางอย่างต้องส่งค่าประกอบตามด้วยจึงจะมีความหมาย. เช่นตัวอย่างที่ 2.23, ตัวเลือก -name ต้องการค่าประกอบซึ่งในตัวอย่างได้แก่คำที่ปรากฏในชื่อไฟล์ที่ต้องการค้นหา. ค่าประกอบที่ส่งพร้อมกับตัวเลือกจะพิมพ์ต่อจากตัวเลือกโดยมักใช้ช่องไฟแยกระหว่างชื่อตัวเลือกและค่าประกอบ.

dd อ้างอิงหน้า 385

ค่าประกอบสำหรับตัวเลือกบางกรณีอาจจะขั้นด้วยเครื่องหมาย “=” เช่นคำสั่ง “dd”.

ตัวอย่างที่ 2.25: การส่งค่าให้ตัวเลือกโดยใช้เครื่องหมาย =

```
$ dd if=/boot/vmlinuz-2.4.20-8 of=/dev/fd0↵
2191+1 records in
2191+1 records out
```

ตัวอย่างดังกล่าวเป็นการก็อปปี้เคอร์เนลลงแผ่นฟล็อปปี้เพื่อใช้เป็นบูตดิสก์โดยที่ `/boot/vmlinuz-2.4.20-8` เป็นค่าประกอบของตัวเลือก `if` (input file) และ `/dev/fd0` เป็นค่าประกอบของตัวเลือก `of` (output file).

ตัวเลือกที่ให้คำสั่งแสดงสถานะการทำงาน

คำสั่งบางอย่างจะไม่มีการแสดงผลทางเทอร์มินอลทำให้ผู้ใช้ไม่รู้ว่าคำสั่งที่สั่งไปนั้นทำงานอยู่หรือไม่โดยเฉพาะคำสั่งที่กินเวลานาน. ในกรณีนี้โปรแกรมอาจจะมีตัวเลือก `-v` หรือ `--verbose` เพื่อแสดงสถานะ, แสดงข้อมูลที่เกี่ยวกับสิ่งที่ทำอยู่ทำให้ผู้ใจมั่นใจว่าโปรแกรมกำลังทำงานจริง ๆ. ตัวเลือกที่แสดงสิ่งที่คำสั่งกระทำการอยู่นี้ไม่จำเป็นต้องเป็น `-v` หรือ `--verbose` ก็ได้, ขึ้นอยู่กับคำสั่งที่ใช้. ในบางกรณีอาจจะไม่มีตัวเลือกที่แสดงสิ่งที่กระทำการหรือผลลัพธ์เลยก็ได้. ตัวอย่างต่อไปนี้แสดงการใช้คำสั่ง `cp` ร่วมกับตัวเลือก `-v` เพื่อแสดงไฟล์ที่กำลังก็อปปี้และไดเรกทอรีเป้าหม้าย.

สำหรับบางคำสั่ง `-v` อาจจะหมายถึง version คือให้แสดงชื่อรุ่นของคำสั่งทางหน้าจอ.

cp อ้างอิงหน้า 395

ตัวอย่างที่ 2.26: ตัวเลือกที่ใช้แสดงสถานะการทำงาน (`-v` หรือ `--verbose`)

```
$ cp -v * /tmp↵
‘file1’ -> ‘/tmp/file1’
‘file2’ -> ‘/tmp/file2’
...
```

ตัวเลือกขอความช่วยเหลือ

โปรแกรมที่ดีมักจะมีตัวเลือกสรุปการใช้อย่างคร่าว ๆไว้ให้. ซึ่งมักจะเป็น `-h` หรือ `--help`. ตัวอย่างเช่น

wc อ้างอิงหน้า 393

ตัวอย่างที่ 2.27: ตัวเลือกแสดงความช่วยเหลือ (`--help`)

```
$ wc --help↵
Usage: wc [OPTION]... [FILE]...
Print newline, word, and byte counts for each FILE, and a total line if
more than one FILE is specified.  With no FILE, or when FILE is -,
read standard input.
  -c, --bytes            print the byte counts
  -m, --chars            print the character counts
  -l, --lines            print the newline counts
  -L, --max-line-length  print the length of the longest line
  -w, --words            print the word counts
      --help     display this help and exit
      --version  output version information and exit

Report bugs to <bug-textutils@gnu.org>.
```

ถ้าผู้ใช้ไม่แน่ใจว่าคำสั่งที่กำลังจะสั่งใช้อย่างไรก็อาจจะลองสั่งคำสั่งนั้นประกอบกับตัวเลือก `-h` หรือ `--help` ดูก็ได้.

ตารางที่ 2.3: สรุปตัวเลือกทั่วไป

ตัวเลือก	ความหมาย	คำอธิบาย	คำสั่ง
-		ข้อมูลนำเข้ามาตรฐาน	`tar`, `cat`, `wc`
`-f`	force	บังคับ	`rm`, `cp`, `mv`
`-f`	file	ชื่อไฟล์	`make`, `tar`
`-i`	interactive	เชิงโต้ตอบ, ถามย้ำ	`rm`, `cp`
`-l`	long	แบบยาว	`ls`, `ps`
`-n`	number	จำนวน, บรรทัด	`head`, `tail`
`-o`	output	ข้อมูลออก, ไฟล์	`sort`, `gcc`
`-r`	recursive	ทำซ้ำไปเรื่อย ๆ	`cp`, `rm`, `grep`
`-R`	recursive	ทำซ้ำไปเรื่อย ๆ	`chmod`, `ls`
`-r`	reverse	กลับกัน	`sort`, `ls`
`-v`	verbose	แสดงสถานะการทำงาน	`gzip`, `cp`

2.5.7 ข้อมูล (data)

การทำความเข้าใจการประมวลผลของคำสั่งต่อ*ข้อมูลเข้า (input data)* และ*ข้อมูลออก (output data)* มีความสำคัญอย่างยิ่งสำหรับการเรียนรู้เชลล์. เพื่อความเข้าใจการทำงานของคำสั่ง, ผู้ใช้ควรจะทำความเข้าใจว่าข้อมูลมาไหน, เมื่อประมวลผลแล้วออกมาทางใด, การกระทำโดยปริยายของคำสั่งจะทำอะไร ฯลฯ.

ข้อมูลเข้า

สำหรับตัวโปรแกรมแล้ว, ข้อมูลเข้าคือข้อมูลจากภายนอกโปรแกรมที่นำเข้ามาในตัวโปรแกรมให้รับรู้. วิธีการนำเข้าของข้อมูลโดยใช้เชลล์เป็นอินเทอร์เฟสนั้นแยกออกเป็นประเภทใหญ่ ๆได้ 3 ประเภทได้แก่

- ข้อมูลตัวเลือก
  โดยปรกติถ้าไม่มีการระบุเลือก, คำสั่งทุกคำสั่งจะมีการกระทำที่กำหนดไว้โดยปริยายอยู่แล้ว. การระบุตัวเลือกเป็นอาร์กิวเมนต์ของคำสั่งถือเป็นข้อมูลที่ป้อนให้คำสั่งอย่างหนึ่งเพื่อให้คำสั่งนั้นกระทำการบางอย่างที่ไม่ใช่การกระทำโดยปริยาย. ตัวอย่างเช่นถ้าใช้ตัวเลือก `--help` กับคำสั่ง `wc` (ตัวอย่างที่ 2.27) จะทำให้คำสั่งแสดงวิธีใช้แทนที่จะนับบรรทัด, คำ, อักขระตามที่ควรจะเป็น.

- ข้อมูลที่อยู่ในไฟล์
  เป็นการเตรียมข้อมูลไว้ก่อนในรูปของไฟล์แล้วส่งชื่อไฟล์ให้โปรแกรมในรูปของอาร์กิวเมนต์. เมื่อโปรแกรมรับรู้ชื่อไฟล์แล้วก็จะเปิดอ่านข้อมูลนำมาประมวลผล

ต่อไป. ตัวอย่างเช่นถ้าให้ชื่อไฟล์เป็นอาร์กิวเมนต์ของโปรแกรม wc, ตัวโปรแกรม wc ก็จะทำการเปิดไฟล์นั้นและนับจำนวนบรรทัด, คำ, และอักขระแล้วส่งเป็นข้อมูลที่ประมวลผลเรียบร้อยแล้วออกทางหน้าจอ.

ตัวอย่างที่ 2.28: การใช้ wc นับบรรทัด, คำ, และอักขระในไฟล์.

```
$ wc /etc/passwd↵
     48      66    2125 /etc/passwd
```

- ข้อมูลนำเข้ามาตรฐาน
  ข้อมูลนำเข้ามาตรฐาน (standard input) หรือเรียกย่อๆว่า stdin เป็นช่องทางที่โปรแกรมรับข้อมูลที่ป้อนให้จากเชลล์. โดยปรกติแล้ว stdin จะถูกเชื่อมเข้ากับคีย์บอร์ด. กล่าวคือข้อมูลที่พิมพ์ด้วยคีย์บอร์ดก็คือ stdin นั่นเอง. คำสั่งยูนิกซ์ (ลินุกซ์ก็เช่นกัน) ส่วนใหญ่มักจะรับข้อมูลจาก stdin ถ้าไม่มีการระบุชื่อไฟล์ (ข้อมูล) ที่ต้องการประมวลผล.

  เรามาดูตัวอย่างคำสั่ง wc อีกทีแต่ครั้งนี้จะสั่งคำสั่ง wc โดยไม่มีอาร์กิวเมนต์. ในกรณีนี้ wc จะรับข้อมูลจาก stdin ซึ่งก็คือข้อมูลที่เราต้องพิมพ์จากคีย์บอร์ดนั่นเอง.

  ตัวอย่างที่ 2.29: การป้อนข้อมูลทาง stdin ให้โปรแกรม

```
$ wc↵                                   ← รับข้อมูลจาก stdin ผ่านทางคีย์บอร์ด
If you can't be a pine on the top of the hill,↵
Be a scrub in the valley -- but be↵
The best little scrub by the side of the rill;↵
Be a bush, if you can't be a tree.↵
Ctrl+d                                  ← ส่งอักขระควบคุม EOT เพื่อบอกจบข้อมูล
       4      40     164
```

  วิธีการบอกให้โปรแกรมรู้ว่าข้อมูลที่ป้อนให้หมดแล้วทำได้โดยการส่งอักขระควบคุม EOT (End Of Transmission) ซึ่งก็คือ Ctrl+d. หลังจากนั้น, wc ก็จะประมวลผลที่เราป้อนเข้าไปทาง stdin.

จะเห็นได้ว่าป้อนข้อมูลให้คำสั่งทำได้หลายวิธีที่แนะนำไปแล้ว. โดยทั่วไปคำสั่งมาตรฐานในลินุกซ์จะมีรูปแบบเป็น

```
$ คำสั่ง ตัวเลือก ชื่อไฟล์
```

ตัวเลือกอาจจะอยู่ก่อนหรือหลังชื่อไฟล์ก็ได้. ชื่อไฟล์อาจจะมีมากกว่าหนึ่งไฟล์หรือไม่ระบุชื่อไฟล์เลยก็ได้. ถ้าไม่ระบุชื่อไฟล์โปรแกรมนั้นอาจจะรับข้อมูลจาก stdin. ในบางโปรแกรมเช่น tar ถ้าระบุชื่อไฟล์เป็น "-" จะถือว่าเป็นการรับข้อมูลจาก stdin ซึ่งไม่ใช่ไฟล์จริงๆ.

ข้อมูลออก

เมื่อคำสั่งประมวลผลข้อมูลเรียบร้อยแล้ว, คำสั่งนั้นๆจะแสดงผลลัพธ์โดยการส่งข้อมูลออกมาในรูปแบบต่างๆ. ข้อมูลที่ออกมาจากการประมวลผลของคำสั่งนี้แบ่งออกได้เป็น

- ข้อมูลออกมาตรฐาน
  ข้อมูลออกมาตรฐาน (standard output) หรือเรียกย่อๆว่า stdout เป็นช่องทางที่โปรแกรมหรือคำสั่งส่งข้อมูลออกให้เชลล์. โดยปรกติแล้ว stdout จะแสดงผลทางหน้าจอเทอร์มินอลที่สั่งคำสั่ง. แต่ข้อมูลที่แสดงทางเทอร์มินอลไม่จำเป็นต้องเป็น stdout เสมอ, อาจจะเป็น stderr ก็ได้. คำสั่งมาตรฐานในลินุกซ์ถ้าไม่มีการระบุชื่อไฟล์เก็บบันทึกข้อมูลออก, คำสั่งนั้นมักจะส่งข้อมูลออกทาง stdout โดยปริยาย.

- ข้อผิดพลาดมาตรฐาน
  ข้อผิดพลาดมาตรฐาน (standard error) หรือเรียกสั้นๆว่า stderr เป็นช่องทางที่คำสั่งหรือโปรแกรมส่งข้อผิดพลาดให้เชลล์รับรู้. ข้อผิดพลาด stderr นี้จะออกมาทางหน้าจอที่แสดงคำสั่งในกรณีที่คำสั่งที่สั่งทำงานไม่เสร็จบริบูรณ์, เกิดข้อผิดพลาดระหว่างประมวลผลโดยไม่ได้คาดหมาย, เกิดข้อผิดพลาดจากการใช้งานผิด ฯลฯ. ตัวอย่างเช่นคำสั่ง `diff` เป็นคำสั่งที่แสดงความแตกต่างของไฟล์สองไฟล์แต่ถ้าเราไม่ใส่ชื่อไฟล์เป็นอาร์กิวเมนต์หรือใส่ชื่อไฟล์ไม่ครบสองไฟล์, ถือว่าเป็นข้อผิดพลาด. ทำให้คำสั่งนั้นทำงานไม่ถูกต้องหรือทำงานไม่ได้.

diff อ้างอิงหน้า 386

ตัวอย่างที่ 2.30: ข้อมูล stderr เมื่อคำสั่งที่สั่งเกิดข้อผิดพลาด

```
$ diff↵
diff: missing operand after ‘diff’
diff: Try ‘diff --help’ for more information.
```

- ข้อมูลออกบันทึกในไฟล์
  โปรแกรมบางโปรแกรมนอกจากจะแสดงผลทางเทอร์มินอลแล้วยังสามารถเก็บผลลัพธ์ออกบันทึกลงไฟล์ได้ด้วย. ในกรณีของคำสั่ง `sort` กับตัวเลือก `-o` *`filename`* ตัวโปรแกรมจะประมวลผลข้อมูลเข้าแล้วเก็บบรรทัดลงไฟล์ให้แทนที่จะแสดงออกทางหน้าจอ.

sort อ้างอิงหน้า 390

ตัวอย่างที่ 2.31: คำสั่ง `sort` กับตัวเลือกเก็บผลลัพธ์บันทึกลงไฟล์

```
$ sort -o result.txt↵
If you can’t be a pine on the top of the hill,↵
Be a scrub in the valley -- but be↵
The best little scrub by the side of the rill;↵
Be a bush, if you can’t be a tree.↵
Ctrl+d
$ cat result.txt↵
Be a bush, if you can’t be a tree.
Be a scrub in the valley -- but be
If you can’t be a pine on the top of the hill,
```

```
The best little scrub by the side of the rill;
```

สำหรับบางโปรแกรมอาจจะบันทึกข้อมูลออกลงไฟล์ที่กำหนดโดยโปรแกรมเองโดยปริยายก็ได้. ในกรณีนี้ต้องดูคือมือประกอบว่าไฟล์ที่บันทึกให้นั้นชื่ออะไรอยู่ที่ไหน. หรือถ้าเป็นคำสั่งที่ใจดีก็จะถามผู้ใช้ให้ตั้งชื่อไฟล์ที่จะบันทึก.

stdout และ stderr ส่งข้อมูลออกทางหน้าจอเทอร์มินอลเหมือนกัน. จึงยากที่จะรู้ว่าข้อมูลไหนเป็น stdout อันไหนเป็น stderr แต่เชลล์จะรู้เพราะข้อมูลเหล่านี้ออกมาคนละช่องทางกัน. ช่องทางที่ว่านี้ในระบบปฏิบัติการจะมีหมายเลขกำกับไว้ให้เรียกว่า *file descriptor*. หมายเลข file descriptor ของ stdin, stdout และ stderr ถูกกำหนดไว้ตายตัวแล้วได้แก่ 0, 1 และ 2 ตามลำดับ.

file descriptor ▶ ตัวเลขจำนวนเต็มที่ระบบปฏิบัติการใช้อ้างอิงไฟล์ที่เปิดไว้. โดยปรกติจะมีการกำหนดค่า file descriptor ให้กับ stdin, stdout และ stderr โดยปริยายเป็น 0, 1 และ 2 ตามลำดับ.

วิธีการดูว่าคำสั่งที่สั่งไปนั้นเกิดข้อผิดพลาดหรือไม่ทำได้โดยการตรวจสอบตัวแปรเชลล์ "$?" หลังจากที่สั่งคำสั่งเรียบร้อยแล้ว. ตัวแปรเชลล์นี้เป็นตัวบอก*สถานะการจบการทำงาน (exit status)* ของคำสั่ง.

ไฟล์ `/etc/shadow` เป็นไฟล์ที่เก็บรหัสผ่านของผู้ใช้งานในระบบ. ไฟล์นี้ root จะอ่านได้เพียงผู้เดียว.

ตัวอย่างที่ 2.32: การตรวจสอบสถานะการจบการกระทำ

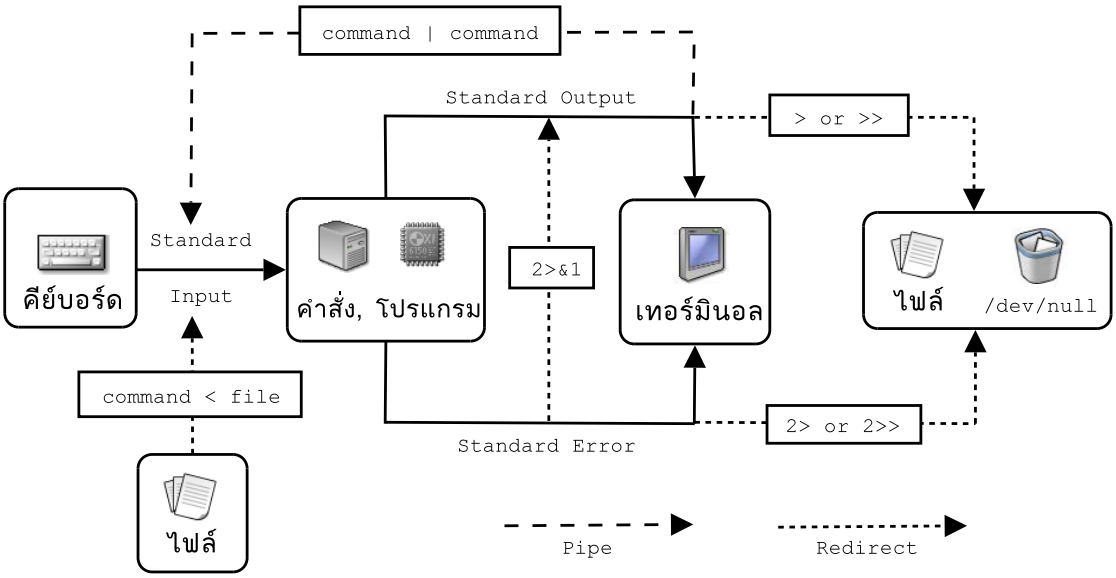
```
$ ls -l /etc/shadow↵
-rw-------    1 root     root          638 Apr 16 19:54 /etc/shadow
$ echo $?↵
0
$ cat /etc/shadow↵
cat: /etc/shadow: Permission denied
$ echo $?↵
1
```

เป็นธรรมเนียมของยูนิกซ์ที่ว่าถ้าคำสั่งทำงานจบบริบูรณ์ exit status จะมีค่าเป็น 0 (โปรแกรมตั้งค่าให้เป็น 0). ถ้าเกิดข้อผิดพลาดขึ้นโปรแกรมนั้นจะส่ง exit status ที่ไม่ใช่ 0 อาจจะเป็นเลขจำนวนลบหรือจำนวนก็ได้แล้วแต่ผู้ออกแบบโปรแกรมนั้น ๆ. เนื่องจากชนิดของข้อผิดพลาดมีหลายประเภทเช่น ใช้งานผิดจึงเกิดข้อผิดพลาด, ใช้งานถูกต้องแต่ไม่มีสิทธิในการกระทำ ฯลฯ ดังนั้นโปรแกรมอาจจะส่ง exit status เป็นค่าที่ไม่เหมือนกันเพื่อแยกแยะประเภทของข้อผิดพลาดให้รู้ด้วย. ส่วนค่าของตัวเลขนั้นมีความหมายอย่างไรต้องไปอ่านคู่มือประกอบการใช้งานของโปรแกรมหรือคำสั่งนั้น ๆ.

2.5.8 รีไดเรกและไปป์

ถึงจุดนี้อาจจะเกิดคำถามขึ้นว่าทำไมต้องมี stdin, stdout และ stdout ด้วย? คำตอบคือข้อมูลเหล่านี้มี "ช่องทาง" และเราสามารถเปลี่ยนช่องทางของข้อมูลไปมาได้ตามต้องการ. หมายความว่าเราอาจจะนำผลลัพธ์ของคำสั่งหนึ่งไปเป็นข้อมูลเข้าของอีกคำสั่ง. หรือเอาข้อผิดพลาดเก็บลงไฟล์เพื่อเป็นบันทึกการทำงาน. วิธีการเหล่านี้เองทำให้เชลล์มีประสิทธิภาพการใช้งานได้ยืดหยุ่น. ถ้างานที่ต้องการทำไม่สามารถทำได้ด้วยหนึ่งคำสั่งก็นำคำสั่งหลายคำสั่งมาใช้ด้วยกันได้. นี่เป็น*ปรัชญายูนิกซ์ (unix philosophy)* อย่างหนึ่ง

ที่ว่าคำสั่งหรือโปรแกรมเป็น*เครื่องมือ (tool)* เวลาต้องการทำงานอย่างใดอย่างหนึ่งให้ใช้เครื่องมือที่เหมาะกับงาน. เครื่องมือนั้นควรเป็นเครื่องมือที่เล็ก (โปรแกรมขนาดเล็ก) ที่ทำงานพอตัวไม่ใช่เป็นเครื่องมือที่ทำอะไรได้ทุกอย่าง. ถ้าต้องการทำงานใหญ่ให้ใช้เครื่องมือร่วมกันทำงาน.



รูปที่ 2.6: ความสัมพันธ์ระหว่างข้อมูล, รีไดเรกชัน และไปป์.

ไปป์

เราเรียกวิธีการนำผลลัพธ์จาก stdout ของโปรแกรมหนึ่งไปใส่ข้อมูลเข้าทาง stdin ของอีกโปรแกรมหนึ่งว่า*ไปป์ (pipe)*. สมมติว่าเราใช้คำสั่ง `ls` ดูรายการไฟล์ในไดเรกทอรีจะเห็นว่าคำสั่ง `ls` แสดงผลออกมาทางหน้าจอให้ดูง่ายเช่นแสดงชื่อไฟล์เป็นหลายคอลัมภ์.

ถ้าจะแปลตามศัพท์ก็คือท่อ.

ตัวอย่างที่ 2.33: ข้อมูลออกทางเทอร์มินอล (คำสั่ง `ls`)

```
$ ls↵
file1 file2 file3
```

ตัวอย่างนี้ไม่ค่อยสร้างสรรค์เท่าไหร่นัก. แต่ต้องการแสดงให้ว่าข้อมูลที่ออกทาง stdout แตกต่างจากข้อมูลออกทางเทอร์มินอล.

คราวนี้เรานำข้อมูลออกของคำสั่ง `ls` มาใส่เป็นข้อมูลเข้า stdin ของคำสั่ง `cat` จะได้ผลลัพธ์ออกมาเป็นชื่อไฟล์หนึ่งไฟล์ต่อหนึ่งบรรทัด.

ตัวอย่างที่ 2.34: การใช้ไปป์เปลี่ยนข้อมูลจาก stdout ของโปรแกรมหนึ่งไปเป็น stdin ของโปรแกรมหนึ่ง

```
$ ls | cat↵
file1
file2
file3
```

นี่เป็นตัวอย่างให้เห็นว่าโปรแกรมบางอย่างเช่น `ls` ตรวจสอบว่าข้อมูลที่ส่งออกมานั้นไปที่เทอร์มินอลหรือไม่. ถ้าที่ที่แสดงผลเป็นเทอร์มินอลก็จะปรับแต่งให้เหมาะสมให้ดูสะดวก.

โปรแกรมที่ใช้กับไปป์บางทีเรียกว่า*ตัวกรอง (filer)* เพราะผู้ใช้จะเลือกคำสั่งรับข้อมูลและสกัดข้อมูลที่ต้องการต่อไป. โปรแกรมที่กรองข้อมูลที่ต้องการ (หรือไม่ต้องการ) มักจะประมวลผลบรรทัดต่อบรรทัด. และในลินุกซ์จะถือว่าอักขระที่เป็นตัวแบ่งบรรทัด (EOL, end-of-line) ได้แก่อักขระ line feed (LF). โปรแกรมกรองข้อมูลเป็นบรรทัดต่อบรรทัดที่ใช้บ่อยได้แก่ `grep`, `sed`, `head`, `tail`, `sort`, `uniq` ฯลฯ. ตัวอย่างเช่นการตรวจดูจำนวนการใช้ฮาร์ดดิกส์ใต้ไดเรกทอรี `/tmp` ว่าแต่ละไดเรกทอรีใช้พื้นที่ไปเท่าไรด้วยคำสั่ง `du` จะแสดงผลดังนี้.

line feed มีชื่อเรียกอีกอย่างว่า new line ถ้าเขียนอักขระนี้ในภาษา C จะใช้ `'\n'` แทนตัวอักขระนี้.

ในระบบปฏิบัติการวินโดวส์จะใช้อักขระ carriage-return และ line-feed (CR/LF) สองตัวติดกันเป็นตัวแบ่งบรรทัด. ถ้านำไฟล์ของวินโดวส์มาใช้ต้องเปลี่ยนอักขระสองตัวนี้ให้เป็น LF.

du อ้างอิงหน้า 395

ตัวอย่างที่ 2.35: การตรวจการใช้พื้นที่ในแต่ละไดเรกทอรีที่กำหนด

```
$ du -D /tmp↵
4       /tmp/.ICE-unix
4       /tmp/.X11-unix
4       /tmp/.font-unix
4       /tmp/ssh-gvHu4731
...
4       /tmp/ksocket-poonlap
3796    /tmp
```

ถ้าเราต้องการเรียงลำดับดูว่าไดเรกทอรีไหนถูกใช้พื้นที่มากต้องดูเองด้วยตาเพราะคำสั่ง `du` ทำหน้าที่รายงานผลการใช้พื้นที่ฮาร์ดดิสก์เท่านั้น. ในกรณีนี้เราสามารถใช้ไปป์ช่วยส่งผลลัพธ์ของคำสั่ง `du` ที่เป็น stdout ให้ไปป์เปลี่ยนไปเป็น stdin ของคำสั่ง `sort` ซึ่งไว้ใช้เรียงลำดับข้อมูลแต่ละบรรทัดได้.

ตัวอย่างที่ 2.36: การใช้ไปป์กรองข้อมูลที่ต้องการ

```
$ du -D /tmp | sort -nr↵
3796    /tmp
624     /tmp/kde-poonlap
492     /tmp/vmware-config0
456     /tmp/vmware-config0/vmnet-only
8       /tmp/orbit-poonlap
8       /tmp/mcop-poonlap
...
```

ถ้าข้อมูลออกทีแสดงบนหน้าจอมีมากเกินหนึ่งหน้าก็อาจจะใช้ไปป์ส่งข้อมูลต่อให้เพจเจอร์ `less` ก็ได้. หรือถ้าต้องการดูผลตามจำนวนที่ต้องการก็ใช้คำสั่ง `head` เลือกจำนวนบรรทัดได้.

head อ้างอิงหน้า 388

ตัวอย่างที่ 2.37: การใช้ `head` แสดงบรรทัดต้น ๆของไฟล์

```
$ du -D /tmp | sort -nr | head -n 3↵
3796    /tmp
24     /tmp/kde-poonlap
492     /tmp/vmware-config0
$
```

cut อ้างอิงหน้า 385

ถ้าเราไม่สนใจตัวเลขแต่ต้องการดูแค่ชื่อไดเรกทอรีก็สามารถใช้คำสั่ง cut ตัวเอาส่วนที่ต้องการมาดูต่อได้ดังนี้.

ตัวอย่างที่ 2.38: การใช้ cut ตัดคอลัมภ์ที่ต้องการ

```
$ du -D /tmp | sort -nr | head -n 3 | cut -f 2↵
/tmp
/tmp/kde-poonlap
/tmp/vmware-config0
$
```

รีไดเรก

ไปป์เป็นการเปลี่ยนช่องทางของข้อมูลจาก stdout ของโปรแกรมหนึ่งไปเป็น stdin ของอีกโปรแกรมหนึ่ง. ส่วนการเปลี่ยน stdout ไปบันทึกในไฟล์ (ใช้เครื่องหมาย > หรือ >>), หรือนำข้อมูลจากไฟล์มาใส่ใน stdin (ใช้เครื่องหมาย <) นั้นเรียกกรรมวิธีนี้ว่า*รีไดเรก (redirect).*

ถ้าจะแปรตามศัพท์ก็คือเปลี่ยนทิศทาง.

ตัวอย่างที่ 2.39: รีไดเรก

```
$ cat > poem.txt↵                          ← นำผลลัพธ์ของ cat (stdout) บันทึกลงไฟล์
If you can't be a bush, be a bit of the grass,↵
And some highway happier make;↵
Ctrl+d
$ cat poem.txt↵                                ← cat เป็นตัวเปิดและอ่านข้อมูลจากไฟล์
If you can't be a bush, be a bit of the grass,
And some highway happier make;
$ cat >> poem.txt↵                         ← บันทึกข้อมูลจากอมูลจาก stdout ต่อท้ายไฟล์
If you can't be a muskie, then just be a bass --↵
But the liveliest bass in the lake!↵
Ctrl+d
$ cat < poem.txt↵                       ← เชลล์เป็นตัวอ่านข้อมูลจากไฟล์แล้วป้อนใส่ stdin
If you can't be a bush, be a bit of the grass,
And some highway happier make;
If you can't be a muskie, then just be a bass --
But the liveliest bass in the lake!
```

จากตัวอย่าง cat รับข้อมูลจาก stdin (คีย์บอร์ด) และเขียนส่งข้อมูลออกทาง stdout (cat รับข้อมูลมาอย่างไรก็ส่งออกอย่างนั้น). เมื่อเชลล์เจอเครื่องหมาย “>” จึงแปรความและเก็บบันทึกข้อมูลออกลงในไฟล์ชื่อ poem.txt. คำสั่ง “cat >> poem.txt” ก็คล้ายกันแต่ใช้เครื่องหมาย “>>” เป็นการเพิ่มข้อมูลต่อในไฟล์โดยไม่เขียนทับเป็นไฟล์ใหม่ (ข้อมูลเดิมยังอยู่).

ให้สังเกตุว่า “cat file” กับ “cat < file” มีความหมายต่างกันแต่ให้ผลที่เหมือนกัน. สำหรับคำสั่ง cat file นั้น, ตัวโปรแกรม cat จะเป็นตัวที่เปิดไฟล์แล้วอ่านข้อมูลจากไฟล์นั้น. ส่วน cat < file นั้น, เชลล์จะเปิดไฟล์ให้แล้วเอาข้อมูลที่อ่านจากไฟล์ป้อนให้โปรแกรมทาง stdin.

การใช้เครื่องหมาย “>” รีไดเรกบางควรจะระมัดระวังไฟล์ที่ระบุ. ถ้าหากระบุไฟล์ที่มีอยู่แล้วจะเป็นการเขียนไฟล์ทับไฟล์เก่า, อาจจะทำให้ข้อมูลเสียหายโดยไม่ตั้งใจได้. ใน

กรณีนี้เราสามารถใช้คำสั่งภายในเชลล์ “set” ปรับแต่งเชลล์ไม่ให้เขียนไฟล์ทับไฟล์เวลาใช้รีไดเรกในกรณีที่มีไฟล์ที่ระบุอยู่แล้ว.

set อ้างอิงหน้า 415

ตัวอย่างที่ 2.40: ใช้ set ปรับแต่งเชลล์ห้ามเขียนทับไฟล์เวลารีไดเรก

```
$ ls -l poem.txt↵
-rw-r--r--    1 somchai  users         163 Apr 23 02:22 poem.txt
$ set -o noclobber↵                                    ← หรือ set -C ก็ได้
$ > poem.txt↵                              ← รีไดเรก stdout (ไม่มีอะไร) ลงในไฟล์
-bash: poem.txt: cannot overwrite existing file
$ >| poem.txt↵                              ← บังคับการีไดเรกถึงแม้จะไฟล์นั้นจะมีตัวตน
$ ls -l poem.txt
-rw-r--r--    1 somchai  users           0 Apr 23 02:25 poem.txt
```

เครื่องหมาย “>|” เป็นการบังคับรีไดเรกลงไฟล์ในกรณีที่เชลล์ตั้งค่าไม่ให้รีไดเรกถ้ามีไฟล์นั้นอยู่แล้วก็ตาม.

เราลองมาพิจารณาคำสั่งต่อไปนี้

ตัวอย่างที่ 2.41: การรีไดเรกชันข้อมูลเข้าและออกไฟล์เดียวกัน

```
$ set +o noclobber↵                                          ← หรือ set +C
$ cat > poem3.txt↵
We can't all be captains, we've got to be a crew,↵
There's something for all of us here.↵
There's a big work to do and there's a lesser to do.↵
And the task we must do is the near.↵
Ctrl+d
$ cat -n poem3.txt↵                              ← เติมหมายเลขบรรทัดให้แต่ละบรรทัด
     1  We can't all be captains, we've got to be a crew,
     2  There's something for all of us here.
     3  There's a big work to do and there's a lesser to do
     4  And the task we must do is the near.
$ cat -n < poem3.txt > poem3.txt↵              ← ใช้ชื่อไฟล์ข้อมูลเข้าและออกเดียวกัน
cat: poem3.txt: input file is output file
$ cat poem3.txt↵                                        ← กลายเป็นไฟล์ว่างเปล่า
$ ls -l poem3.txt↵
-rw-r--r--    1 somchai  users           0 Apr 24 20:27 poem3.txt
```

จากตัวอย่างที่ 2.41 จะเห็นว่าการรีไดเรกไฟล์ข้อมูลเข้าและออกที่มีชื่อเดียวกันจะทำให้เกิดผลลัพธ์ที่ไม่ถูกต้องตามต้องการ (ดูผิวเผินแล้วน่าจะถูกต้อง) ทำให้ไฟล์ต้นฉบับเป็นไฟล์ว่างเปล่า. ถ้าต้องการทำเช่นนี้จริง ๆก็ควรรีไดเรกข้อมูลออกไปไฟล์ชั่วคราวก่อนแล้วเปลี่ยนชื่อไฟล์ชั่วคราวนั้นไปเป็นไฟล์ที่ต้องการอีกที. หรือปรับแต่งเชลล์ให้ปัจจัย noclobber มีผล, เพื่อป้องการการรีไดเรกไฟล์ที่มีอยู่แล้ว. สำหรับโปรแกรมบางอย่างเช่น `sort` มีการเตรียมตัวเลือก `-o` *`file`* ให้และสามารถระบุชื่อไฟล์ที่ต้องการเก็บข้อมูลออกเป็นไฟล์เดียวกันได้.

การรีไดเรกไม่จำกัดเฉพาะแค่การเปลี่ยนทิศทางระหว่างไฟล์กับ stdin หรือ stout เท่านั้นแต่สามารถเปลี่ยนทิศทางไปมาได้ระหว่าง stdin, stdout, stderr, และไฟล์.

ในกรณีที่โปรแกรมหรือคำสั่งแสดงข้อมูลออกทาง stderr และ stdout พร้อม ๆกันทางหน้าจอ, จะทำแยกแยะผลลัพธ์ที่ต้องการได้ไม่สะดวกนัก. แต่ข้อผิดพลาดที่แสดงออกทาง

ตารางที่ 2.4: รูปแบบการรีไดเรกต่าง ๆ

รูปแบบการรีไดเรก	คำอธิบาย
[n]<*file*	ให้ file descriptor, n มาจากการเปิดรับข้อมูลเข้าจากไฟล์ *file*. ถ้าไม่ระบุ n ถือว่า n คือ 0 (stdin).
[n]>*file*	นำข้อมูลออกจาก file descriptor, n บันทึกลงในไฟล์ *file*. ถ้าไม่ระบุ n ถือว่า n คือ 1 (stdout)
[n]>>*file*	นำข้อมูลออกจาก file descriptor, n บันทึกลงต่อท้ายไฟล์ *file*. ถ้าไม่ระบุ n ถือว่า n คือ 1 (stdout)
&>*file* หรือ >&*file*	บันทึกข้อมูลจาก stdout และ stderr บันทึกลงในไฟล์ *file*. ให้ผลเหมือนกับ ">*file* 2>&1".
[n]<&[N]	ให้ข้อมูลเข้าของ file descriptor, n มาจากข้อมูลเข้าของ file descriptor, N (file descriptor copy).
[n]>&[N]	ให้ข้อมูลออกของ file descriptor, n ไปที่ข้อมูลออกของ file descriptor, N.
<<*word*	อ่านข้อมูลต่อไปเรื่อย ๆ จนกว่าจะถึงบรรทัดที่มีคำว่า *word* ปรากฏอยู่ที่ต้นบรรทัด.
<<<*word*	ให้ *word* เป็นข้อมูลนำเข้า stdin.

หน้าจอก็มีประโยชน์เพราะผู้ใช้จะได้รู้ว่าเกิดข้อผิดพลาดอะไรบ้าง. ถ้าต้องการแยกข้อมูล stdout ออกจาก stderr และบันทึกลงไฟล์, ทำได้ดังนี้.

ไฟล์ /etc/shadow เป็นไฟล์ที่เก็บข้อมูลรหัสผ่านของผู้ใช้ระบบและ root เท่านั้นที่อ่านเนื้อหาภายในได้.

ตัวอย่างที่ 2.42: การแยกข้อมูลออก stdout และ stderr บันทึกลงคนละไฟล์

```
$ cat /etc/shadow /etc/passwd↵
cat: /etc/shadow: Permission denied              ← บรรทัดนี้คือข้อผิดพลาด (stderr)
root:x:0:0:root:/root:/bin/bash          ← ตั้งแต่บรรทัดนี้คือผลลัพธ์ปรกติ (stdout)
bin:x:1:1:bin:/bin:/bin/false
daemon:x:2:2:daemon:/sbin:/bin/false
...
$ cat /etc/shadow /etc/passwd > result.txt 2> error.txt↵
```

ในกรณีที่สนใจข้อผิดพลาดที่คำสั่งแสดง, เราอาจจะรีไดเรกข้อผิดพลาดไปที่ไฟล์พิเศษที่ชื่อว่า /dev/null. ไฟล์นี้คล้ายถังขยะ, เราสามารถรีไดเรกข้อมูลที่ไม่ต้องการหรือต้องการทิ้งลงไฟล์นี้ได้. ตัวอย่างการรีไดเรกข้อมูลจาก stderr ไปที่ /dev/null ทำได้ดังนี้.

ตัวอย่างที่ 2.43: การรีไดเรกข้อผิดพลาดทิ้ง

```
$ cat /etc/shadow /etc/passwd 2> /dev/null↵
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false
daemon:x:2:2:daemon:/sbin:/bin/false
...
```

คราวนี้เรามาดูการรีไดเรกอีกแบบที่เรียกว่า *here document*.

ตัวอย่างที่ 2.44: การใช้ here document

```
$ cat <<EOF > file.txt↵                          ← หรือ cat > file.txt <<EOF
> You can put a variable here, $HOME.↵                ← ใช้ตัวแปรเชลล์ได้
> Or use command substitute ‘date‘↵                  ← ใช้การแทนคำสั่งได้
> The word EOF must be at the beginning of the line.↵
> EOF↵                                               ← บอกจบข้อมูลนำเข้า
$ cat file.txt↵
You can put a variable here, /home/somchai.
Or use command substitute, Sat Apr 24 23:38:05 JST 2004
The word EOF must be at the beginning of the line.
```

เมื่อเทียบกับตัวอย่างที่ 2.39 การใช้ here document ดูแล้วคล้ายกับการป้อนข้อมูลเข้าทาง stdin ตามปรกติ. แต่จะมีความแตกต่างกันดังนี้

- ในตัวอย่างที่ 2.39, คำสั่ง `cat` จะเป็นตัวรับข้อมูลเข้าจาก stdin แต่สำหรับการใช้ here document แล้ว, เชลล์จะรับข้อมูลเข้าจาก stdin ของเชลล์แล้วส่งต่อให้ทาง stdin ของ `cat`.

- เนื่องจากเชลล์เป็นตัวจัดการ here document และรับรู้ว่าเราป้อนข้อมูลทางเทอร์มินอล, เชลล์จึงสร้าง*พรอมต์ลำดับที่สอง (secondary prompt)* ซึ่งในที่นี้ได้แก่เครื่องหมายมากกว่า (>) เพื่อให้รู้ว่าเชลล์กำลังรอรับข้อมูลอยู่. ในกรณีนี้จะเข้าใจง่ายกว่าตัวอย่างที่ 2.39 เพราะ `cat` ไม่ได้แสดงพรอมต์อะไรเลยอาจทำให้ผู้ใช้ที่เพิ่งเริ่มใช้ไม่เข้าใจ.

  ผู้ใช้สามารถปรับแต่งการพรอมต์ลำดับที่สองด้วยตัวแปรสภาพแวดล้อม PS2 ให้เป็นเครื่องหมายอย่างอื่นได้.

- here document จะใช้คำที่กำหนดไว้ตอนแรกเป็นตัวบอกว่าหมดข้อมูลนำเข้า. ในตัวอย่างใช้คำว่า `EOF` ซึ่งจริงๆแล้วจะเป็นคำอะไรก็ได้. เมื่อพิมพ์คำที่กำหนดไว้ต้นบรรทัดแล้วกด Enter เชลล์ก็จะรับรู้ว่าไม่มีข้อมูลเข้าอีกต่อไป. ตรงนี้จะต่างกับการรับข้อมูลเข้าของโปรแกรมทาง stdin ที่ว่าเราต้องกดคีย์ Ctrl+d เพื่อบอกการจบของข้อมูลเข้า.

- เราสามารถใช้ตัวแปรเชลล์และการแทนที่คำสั่ง (หน้า 51) ได้ด้วย.

การรีไดเรกที่คล้ายกับ here document อีกอย่างได้แก่ “`<<<`*word*”. เชลล์จะถือว่าคำ (*word*) ที่พิมพ์ไปเป็นข้อมูลเข้าทาง stdin. ไม่รับข้อมูลหลายบรรทัดเหมือน here document.

`tee`

การใช้รีไดเรกเป็นการนำข้อมูลจาก stdio บันทึกลงไฟล์จึงทำให้ไม่เห็นผลลัพธ์ทางหน้าจอ. ในกรณีที่ต้องการให้แสดงผลลัพธ์ทางหน้าจอด้วยแล้วบันทึกผลลัพธ์นั้นลงในไฟล์พร้อมๆให้ส่งข้อมูลให้ไปป์ไปหาคำสั่ง `tee`. คำสั่ง `tee` จะรับข้อมูลจาก stdin แล้วส่งออกทาง stdout พร้อมกับบันทึกลงไฟล์.

tee อ้างอิงหน้า 416

ตัวอย่างที่ 2.45: การใช้ tee แสดงผลทางหน้าจอและบันทึกลงไฟล์พร้อม ๆกัน.

```
$ echo Hello World! | tee result.txt↵
Hello World!
$ cat result.txt↵
Hello World!
```

2.5.9 การจัดกลุ่มคำสั่ง

ผู้ใช้สามารถสั่งคำสั่งมากกว่า 2 คำสั่งในบรรทัดเดียวได้โดยใช้เครื่องหมาย semi-colon (;) คั่นคำสั่ง. ตัวอย่างต่อไปนี้เป็นการใช้คำสั่ง cal และ date ในบรรทัดเดียวกัน.

cal อ้างอิงหน้า 409

ตัวอย่างที่ 2.46: การสั่งคำสั่งมากกว่าสองคำสั่งในบรรทัดเดียว

```
$ cal; date↵
     เมษายน 2003
อา  จ. อ. พ. พฤ ศ. ส.
        1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

ส. เม.ย. 5 01:55:25 JST 2003
```

เราลองมาพิจารณาคำสั่งต่อไปนี้

```
$ cal; date > result.txt↵
     เมษายน 2003
อา  จ. อ. พ. พฤ ศ. ส.
        1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
$ cat result.txt↵
ส. เม.ย. 5 01:56:34 JST 2003
```

จะเห็นว่าผลลัพธ์ที่เก็บลงในไฟล์จะเป็นผลลัพธ์ของคำสั่ง date เท่านั้นเพราะอักขระที่แยกคำสั่งในที่นี้คือ semi-colon (;). ดังนั้น “date > result.txt” ถือเป็นคำสั่งหนึ่งคำสั่ง. ในกรณีที่ต้องเก็บผลลัพธ์ของคำสั่ง “cal; date” พร้อม ๆกันให้ใช้เครื่องหมายวงเล็บเพื่อจับกลุ่มของคำสั่งให้เป็นหนึ่งเดียวกัน.

ตัวอย่างที่ 2.47: การรวมผลลัพธ์ของคำสั่งตั้งแต่ 2 คำสั่งด้วยกัน

```
$ (cal; date) > result.txt↵
$ cat result.txt↵
     เมษายน 2003
อา  จ. อ. พ. พฤ ศ. ส.
        1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
```

```
27 28 29 30

ล. เม.ย. 5 01:56:50 JST 2003
```

2.5.10 การแทนที่ (substitue) คำสั่ง

การแทนที่คำสั่งได้แก่การนำผลลัพธ์ของคำสั่งมาใช้ในบรรทัดคำสั่ง. ตัวอย่างต่อไปนี้เป็นการสร้างไดเรกทอรีให้มีชื่อในรูปแบบ “ปี-เดือน-วัน” โดยใช้ผลลัพธ์จากคำสั่ง `date`.

mkdir อ้างอิงหน้า 397
date อ้างอิงหน้า 411

ตัวอย่างที่ 2.48: การนำผลลัพธ์คำสั่งมาใส่ในบรรทัดคำสั่ง

```
$ mkdir ‘date +%F‘↵
$ ls -l↵
total 4
drwxrwxr-x    2 somchai  somchai      4096 Oct 12 17:00 2003-10-12/
```

การใช้ผลลัพธ์จากคำสั่งที่ต้องการมาแทนที่ในบรรทัดคำสั่งมีรูปแบบเป็น ‘*คำสั่ง*‘ คือใช้เครื่องหมาย backquote คร่อมคำสั่งที่ต้องการแสดงผลลัพธ์. หรือใช้การแทนที่คำสั่งในรูปแบบ `$(`*คำสั่ง*`)` ก็ได้.

เครื่องหมาย backquote (‘) จะคล้ายกับเครื่องหมาย single quote (’) ให้ระวังกดผิดคีย์.

ตัวอย่างที่ 2.49: การนำผลลัพธ์คำสั่งมาใส่ในบรรทัดคำสั่ง (แบบที่ 2)

```
$ mkdir $(date +%F)↵
$ ls -l↵
total 4
drwxrwxr-x    2 somchai  somchai      4096 Oct 12 17:02 2003-10-12/
```

การแทนที่คำสั่งโดยใช้เครื่องหมาย backquote เป็นวิธีที่ใช้ได้ด้วยกับเชลล์ดั้งเดิมเช่น bourn shell, ส่วนการใช้แทนที่ในลักษณะ `$(`คำสั่ง`)` ใช้ได้ใน bash เชลล์.

2.5.11 นามแฝง (alias)

alias เป็นความสามารถของเชลล์อย่างหนึ่งที่ช่วยตั้งชื่อคำสั่งใหม่เพื่อช่วยให้ทำงานได้คล่องขึ้น. การสร้าง alias ทำได้โดยใช้คำสั่งภายในเชลล์ `alias` เช่นการสร้างคำสั่ง `rm` ให้เป็น `alias` ของตัวเองแต่ใส่ตัวเลือก `-i` ด้วย.

alias อ้างอิงหน้า 409

ตัวอย่างที่ 2.50: สร้าง alias

```
$ alias rm='rm -i'↵
```

หลังจากที่สร้าง alias เรียบร้อยแล้วสั่งคำสั่ง `rm` ก็เหมือนกับเราสั่งคำสั่ง “`rm -i`”. การสร้าง `alias` นี้จะทำให้เวลาลบไฟล์ด้วยคำสั่ง `rm` จะถามย้ำทุกครั้งก่อนจะลบไฟล์ทิ้ง. ผู้อ่านอาจจะตั้ง alias ของคำสั่ง `mv` และ `cp` กับตัวเลือก `-i` เพื่อให้ถามย้ำเวลามีไฟล์ชื่อเดียวกันอยู่.

เราสามารถสำรวจดูได้ว่าตอนนี้มี alias อะไรอยู่บ้างโดยสั่งคำสั่ง alias เดี่ยวๆ หรือถ้าต้องการดูนิยามของ alias ก็ใช้ชื่อ alias เป็นอาร์กิวเมนต์.

ตัวอย่างที่ 2.51: สำรวจดู alias ที่นิยามไว้

```
$ alias↵
alias cp='cp -i'
alias latex='latex --translate-file=cp8bit.tcx'
alias ls='ls -F'
alias mozillaj='LANG=ja_JP XMODIFIERS=@xim=kinput2 mozilla'
alias mv='mv -i'
alias rm='rm -i'
```

โดยปรกติ alias จะเซ็ตในไฟล์เริ่มต้นของเชลล์ได้แก่ /etc/bashrc หรือ $HOME/.bashrc.

การยกเลิก alias ทำได้โดยคำสั่ง unalias ตามด้วย alias ที่ต้องการยกเลิก. หรือถ้าต้องการยกเลิก alias ชั่วคราว, ให้พิมพ์เครื่องหมาย "\" นำหน้า alias นั้น.

unalias อ้างอิงหน้า 424

touch อ้างอิงหน้า 400

ตัวอย่างที่ 2.52: การยกเลิก alias

```
$ alias ls rm↵                    ← ตรวจดูนิยามของ alias ที่เป็นอาร์กิวเมนต์ของคำสั่ง
alias ls='ls -F'
alias rm='rm -i'
$ unalias ls↵                                          ← ยกเลิก alias ของ ls
$ touch file↵                                           ← สร้างไฟล์ใหม่ชื่อ file
$ rm file↵                                              ← เหมือนสั่งคำสั่ง rm -i
rm: remove regular empty file 'file'? n↵
$ \rm file↵                                            ← ยกเลิก alias ชั่วคราว
$ cursorprompt
```

การตั้งค่า alias ทำได้โดยเขียนไว้ในไฟล์ /etc/profile ถ้าต้องการตั้งค่าสำหรับทุกคนในระบบ. หรือถ้าต้องการตั้งค่าเฉพาะบุคคลก็เขียน alias ไว้ที่ไฟล์ $HOME/.bashrc ก็ได้.

ไฟล์ /etc/profile เป็นไฟล์ตั้งค่าเริ่มต้นสำหรับเชลล์ล็อกอินของทุกคนในระบบ.

2.5.12 การเติมเต็ม

การเติมเต็มคำสั่ง (auto complementation) เป็นคุณสมบัติของ bash เชลล์ใช้ในการอำนวยความสะดวกเวลาพิมพ์คำสั่งหรือพิมพ์ชื่อไฟล์. ผู้ใช้สามารถพิมพ์คำสั่งบางส่วนแล้วกดคีย์ Tab ให้เชลล์เติมเต็มคำสั่งที่เป็นได้. ตัวอย่างเช่นถ้าต้องการสั่งคำสั่งเรียกโปรแกรม oofice ผ่านทางเชลล์, ผู้ใช้ไม่จำเป็นต้องพิมพ์ตัวอักษรทั้งหมด. ผู้ใช้อาจจะพิมพ์แค่คำว่า "oo" แล้วกดคีย์ Tab Tab ตาม (2 ครั้ง), เชลล์จะค้นหาชื่อโปรแกรมแล้วแสดงชื่อคำสั่ง (โปรแกรม) ที่เป็นไปได้ให้เลือก.

csh มีความสามารถเติมเต็มคำสั่งหรือไฟล์เช่นกันแต่วิธีการใช้จะแตกต่างจาก bash.

ตัวอย่างที่ 2.53: การเติมเต็มคำสั่งโดยใช้ Tab ช่วย

```
$ oo Tab Tab
oocalc     ooffice    oomath     oosetup    oowriter
oodraw     ooimpress  oopadmin   ootags
$ oo
```

ในตัวอย่างใช้ ▮ แสดงตำแหน่งของเคอร์เซอร์.

ถ้าผู้ใช้พิมพ์ตัวอักษร “f” แล้วกด Tab อีกที, เชลล์จะเติมตัวอักษรที่เหลือซึ่งได้แก่ “fice” ให้โดยอัตโนมัติ.

ตัวอย่างที่ 2.54: การเติมเต็มคำสั่งโดยใช้ Tab ช่วย (ต่อ)

```
$ oof[Tab] ⇒ $ office
```

เมื่อเชลล์เติมเต็มคำสั่งให้แล้ว, เชลล์จะเติมช่องไฟหลังคำสั่งให้. ถ้าเชลล์ไม่สามารถเติมเต็มคำสั่งได้เนื่องจากไม่มีตัวเลือกที่เป็นไปได้หรือยังมีส่วนเติมเต็มมากกว่าหนึ่งอย่าง, เชลล์จะไม่เติมช่องไฟให้.

ในกรณีที่มีตัวเลือกมากเกินที่จะแสดงบนหน้าจอได้, เชลล์จะถามย้ำว่าต้องการแสดงส่วนเติมเต็มทั้งหมดหรือไม่. เชลล์จะแสดงชื่อโปรแกรมที่เป็นไปได้ทั้งหมด. หลังจากนั้นผู้ใช้พิมพ์คีย์บอร์ดเพิ่มเติมเพื่อรันโปรแกรมที่ต้องการ.

ตัวอย่างที่ 2.55: การเติมเต็มคำสั่งในกรณีที่คำสั่งมีให้เลือกมาก

```
$ x[Tab][Tab]
Display all 154 possibilities? (y or n) ← กด [y] เพื่อแสดงตัวเลือก
x11perf                     xml2man
x11perfcomp                 xml2pot
...
xcpustate                   xpawhelloworld
--More-- ← กด [space] เพื่อแสดงหน้าต่อไป
xcursor-config              xpdf
xcursorgen                  xphelloworld
...
xft-config                  xset
--More-- ← กด [q] เพื่อยกเลิกการแสดงผล
$ x
$ xd[Tab][Tab]
xdelta          xditview       xdpyinfo       xdvi.bin
xdelta-config   xdm            xdvi           xdviprint
$ xdpy[Tab]
$ xdpyinfo↵
```

นอกจากการเติมเต็มคำสั่งแล้ว, เชลล์ยังสามารถเติมเต็มชื่อไฟล์ที่จะเป็นอาร์กิวเมนต์ของคำสั่งได้ด้วย.

ตัวอย่างที่ 2.56: การเติมเต็มชื่อไฟล์

```
$ cat /proc/m[Tab][Tab]
mdstat   meminfo  misc     modules  mounts   mtrr
$ cat /proc/me[Tab] ⇒ $ cat /proc/meminfo↵
        total:    used:    free:  shared: buffers:  cached:
Mem:  253112320 248029184  5083136        0  4538368 122847232
Swap: 534118400 330682368 203436032
MemTotal:       247180 kB
MemFree:          4964 kB
MemShared:           0 kB
Buffers:          4432 kB
Cached:         109504 kB
SwapCached:      10464 kB
```

...

การเติมเต็มชื่อไฟล์มีประโยชน์อย่างยิ่งเมื่อไฟล์ที่มีชื่อยาวเป็นอาร์กิวเมนต์ของคำสั่งที่ต้องการเรียก. กล่าวคือผู้ใช้ไม่จำเป็นต้องจำชื่อเต็ม ๆทั้งหมดของไฟล์หรือไดเรกทอรี่ที่ไฟล์อยู่, เพียงแต่พิมพ์ชื่อไฟล์บางส่วนแล้วใช้การเติมเต็มช่วยในการหาไฟล์ที่ต้องการ. การเติมเต็มของคำสั่งก็เช่นกัน, ถ้าผู้ใช้จำคำสั่งเต็ม ๆไม่ได้แต่จำส่วนต้น ๆของคำสั่งได้ก็สามารถใช้การเติมเต็มช่วยหาคำสั่งที่ต้องการได้. บางครั้งการใช้การเติมเต็มของคำสั่งช่วยให้รู้จักคำสั่งใหม่ ๆที่ไม่เคยรู้จักด้วย.

2.5.13 ไวล์ดคาร์ด

ในกรณีที่ต้องการระบุชื่อไฟล์หลาย ๆไฟล์พร้อม ๆกันส่งเป็นอาร์กิวเมนต์ให้โปรแกรมใดโปรแกรมหนึ่งจะใช้*ไวล์ดคาร์ด (wildcard)*. เช่นถ้าต้องการลบไฟล์ทุกไฟล์ที่อยู่ในไดเรกทอรีที่ทำงานอยู่, สามารถทำได้โดยคำสั่ง `rm` ร่วมกับไวล์ดคาร์ด.

rm อ้างอิงหน้า 399

*ตัวอย่างที่ 2.57: การใช้ไวล์ดคาร์ด **

```
$ ls↵
file1  file2  file3
$ rm *↵
$ ls↵
$
```

เชลล์จะแปลอักขระ “*” เป็น “file1 file2 file3” แล้วส่งต่อเป็นอาร์กิวเมนต์ให้ “rm” ต่อไป. กล่าวคือคำสั่ง “rm *” จะให้ผลเหมือนกับ “rm file1 file2 file3”. มีข้อควรระวังอยู่คือไวล์ดคาร์ด “*” จะใช้ได้กับไฟล์ธรรมดาที่ไม่ขึ้นต้นด้วย “.” เท่านั้น.

ตารางที่ 2.5: ไวล์ดคาร์ดที่ใช้บ่อย

ไวล์ดคาร์ด	ความหมาย
*	ตัวอักษรใด ๆไม่ว่าจะมีหรือไม่มีก็ได้
?	ตัวอักษรใด ๆหนึ่งตัว
[ตัวอักษร]	ตัวอักษรที่อยู่ใน [] ตัวใดตัวหนึ่ง
[!ตัวอักษร]	ตัวอักษรอะไรก็ได้ยกเว้นตัวอักษรที่อยู่ใน []
{ตัวอักษร,ตัวอักษร,...}	ตัวอักษรอะไรก็ได้ที่อยู่ใน {}

อักขระ “?” ต่างจากอักขระ “*” ที่แทนตัวอักษรใดหนึ่งตัวเท่านั้น.

rm อ้างอิงหน้า 399

ตัวอย่างที่ 2.58: การใช้ไวล์ดคาร์ด ?

```
$ ls↵
a  aa  ab  b  ba  bb
$ rm -v a?↵
removed ‘aa’
```

```
removed ‘ab’
$ rm -v b*↵
removed ‘b’
removed ‘ba’
removed ‘bb’
```

จากตัวอย่างข้างบนจะเห็นว่าเชลล์จะตีความ “a?” เป็นชื่อไฟล์ที่ประกอบด้วยอักษรสองตัวได้แก่ aa, ab. สำหรับ “b*”, เชลล์จะตีความเป็นชื่อไฟล์ที่ขึ้นต้นด้วยอักษร b และมีจำนวนอักษรตั้งแต่หนึ่งตัวอักษรขึ้นไปได้แก่ b, ba และ bb.

นอกจากการใช้ตัวไวล์ดคาร์ด “*” และ “?” แล้ว, เรายังสามารระบุชื่อไฟล์ที่ซับซ้อนกว่านี้ได้โดยใช้ “[]”. ตัวอย่างเช่น “[abc]*” หมายถึงชื่อไฟล์ที่ขึ้นต้นด้วย a หรือ b หรือ c แล้วตามด้วยอะไรก็ได้หรือไม่มีอะไรตามหลังก็ได้. การใช้งานจริง ๆเช่นสมมติว่ามีไฟล์หลายอย่างอยู่ในไดเรกทอรีดังนี้.

ls อ้างอิงหน้า 397

```
$ ls -F↵
COPYING     berrno.h     cygwin_inttypes.h  ffplay.o       output_example*
CREDITS     cmdutils.c   doc/               ffplay_g*      output_example.c
CVS/        cmdutils.h   ffmpeg*            ffserver*      output_example.o
Changelog   cmdutils.o   ffmpeg.c           ffserver.c     q
INSTALL     common.h     ffmpeg.o           ffserver.h     tests/
Makefile    config.h     ffmpeg_g*          ffserver.o     vhook/
README      config.mak   ffplay*            libavcodec/    xvmc_render.h
VERSION     configure*   ffplay.c           libavformat/
```

ถ้าต้องการแสดงไฟล์ที่ขึ้นต้นด้วย “ff” และลงท้ายด้วย “.c” หรือ “.h” เท่านั้นสามารถใช้ไวล์ดคาร์ดตามตัวอย่างต่อไปนี้.

ในตัวอย่างเป็นรหัสต้นฉบับของโปรแกรม ffmpeg ซึ่งมีโปรแกรมต่าง ๆสำหรับ encode วีดีโอ, แพร่ภาพกระจายเสียงผ่านทางเน็ตเวิร์ก.

ตัวอย่างที่ 2.59: การใช้ไวล์ดคาร์ด []

```
$ ls ff*.[ch]↵
ffmpeg.c  ffplay.c  ffserver.c  ffserver.h
```

สมมติว่าต้องการเพิ่มข้อแม้อีกนิดเช่นชื่อไฟล์ต้องมีคำว่า “mpeg” หรือ “server” อยู่ด้วย. ไวล์ดคาร์ดก็จะเป็นดังนี้.

ตัวอย่างที่ 2.60: การใช้ไวล์ดคาร์ด {}

```
$ ls ff*{mpeg,server}*.[ch]↵
ffmpeg.c  ffserver.c  ffserver.h
```

ในทางกลับกัน, ถ้าต้องการแสดงชื่อไฟล์ที่ไม่ได้ลงท้ายด้วย “.c” หรือ “.h” ก็ทำได้โดยเติมเครื่องหมาย “!” ลงไป.

ไฟล์ “.o” เป็น object file ที่เกิดจากการคอมไพล์รหัสต้นฉบับ.

ตัวอย่างที่ 2.61: การใช้ไวล์ดคาร์ด !

```
$ ls ff*{mpeg,server}*.[!ch]↵
ffmpeg.o  ffserver.o
```

ตารางที่ 2.6: ตัวอย่างการใช้ไวล์ดคาร์ดและความหมาย

ตัวอย่างไวล์ดคาร์ด	ความหมาย
a*	ไฟล์ที่ขึ้นต้นด้วยอักษร “a” เช่น a, ab, abc เป็นต้น
a?	ไฟล์ที่ประกอบด้วยอักษรสองตัวและขึ้นต้นด้วย “a” เช่น ab, az เป็นต้น
[abc]	ตัวอักษรตัวเดียว, a หรือ b หรือ c
[a-b]	ตัวอักษรตัวเดียว, a หรือ b หรือ c มีความหมายเหมือนกับตัวอย่างที่แล้ว
[a-zA-Z]	ตัวอักษรตัวเดียวที่เป็นตัวอักษรภาษาอังกฤษเช่น a, b, C, D เป็นต้น
[a-z][a-z][0-9]*	ไฟล์ที่ตัวอักษรสองตัวแรกเป็นตัวอักษรภาษาอังกฤษตัวเล็กแล้วตามด้วยเลขอารบิก ต่อจากนั้นจะเป็นอะไรก็ได้ เช่น ad9, kP0abc, aa8003.jpg เป็นต้น
*{jpg,gif}	อะไรก็ได้ที่ลงท้ายด้วย jpg หรือ gif เช่น pic01.jpg, pic.gif, abcgif, xyzjpg เป็นต้น

2.5.14 อักขระที่มีความหมายพิเศษ

นอกจากไวล์ดคาร์ดแล้วเชลล์ยังแยกแยะอักขระที่มีความหมายพิเศษตามที่แสดงในตารางที่ 2.7. อักขระเหล่านี้ไม่เหมาะที่จะนำมาใช้เป็นชื่อไฟล์เพราะเชลล์จะตีความหมายเป็นอย่างอื่น, ไม่สะดวกต่อการใช้งาน, สร้างความสับสน.

การใช้อักขระที่มีความหมายพิเศษตามตัวอักษรในบรรทัดคำสั่งสามารถทำได้โดยใช้เครื่องหมายคำพูด (quote) คร่อมหรือใช้ escape sequence นำหน้าอักขระที่ต้องการ.

Quote

สมมติว่าเรามีไฟล์ชื่อ “important note.txt” และต้องการจะลบไฟล์นี้ทิ้ง. ถ้าเราสั่งคำสั่ง “rm important note.txt” เชลล์จะตีความว่าเราต้องการลบไฟล์ 2 ฉบับที่ชื่อว่า “important” และ “note.txt” เพราะว่าเชลล์ตีความว่าช่องไฟคืออักขระแบ่งอาร์กิวเมนต์. ในกรณีนี้ผู้ใช้สามารถใช้เครื่องความ double quote (") หรือ single quote (’) คร่อมคำหรือชื่อไฟล์ที่ต้องการตามตัวอย่างที่แสดงดังต่อไปนี้.

ตัวอย่างที่ 2.62: การใช้เครื่องหมายคำพูดในบรรทัดคำสั่ง

```
$ rm important note.txt↵
rm: cannot lstat ‘important’: No such file or directory
rm: cannot lstat ‘note.txt’: No such file or directory
$ rm "important note.txt"↵
หรือ
$ rm ’important note.txt’↵
```

ตารางที่ 2.7: อักขระที่มีความหมายพิเศษในเชลล์

ตัวอักษร	ชื่อ	คำอธิบาย
"	double quote	ใช้เขียนคร่อมประโยคหรือคำที่ไม่ต้องการให้เชลล์ตีความหมายอักขระพิเศษบางชนิดที่อยู่ข้างใน.
#	number sign	เชลล์จะไม่สนใจสิ่งที่อยู่เครื่องหมายนี้และถือเป็นคอมเมนต์.
$	ดอลลาร์	เขียนหน้าชื่อตัวแปรเพื่อแสดงค่าตัวแปร
&	ampersand	เขียนท้ายคำสั่งเพื่อสั่งคำสั่งแบบ background
’	single quote	เช่นเดียวกับเครื่องหมาย " แต่ไม่ตีความเครื่องหมายพิเศษใด ๆเลยที่อยู่ข้างใน
(และ)	วงเล็บ	ใช้จับกลุ่มคำสั่งเหมือนเป็นคำสั่งเดียว.
*	ดอกจัน	ไวล์ดคาร์ดแทนอักษรใด ๆ
;	semicolon	ตัวแบ่งคำสั่ง
<	น้อยกว่า	รีไดเรกชัน, นำข้อมูลเข้าจากไฟล์ส่งผ่านทาง standard input
>	มากกว่า	รีไดเรกชัน, บันทึกผลจาก standard output ลงไฟล์
?	เครื่องหมายคำถาม	แทนอักษรใด ๆตัวเดียว
[และ]	bracket	ใช้เป็นไวล์ดคาร์ด. ใช้แทนคำสั่งภายใน `if`.
\	backslash	เขียนหน้าเครื่องหมายพิเศษใช้แสดงตัวอักษรนั้น ๆ (escape sequence)
‘	back quote (grave)	นำผลลัพธ์ของคำสั่งแทนที่ในตำแหน่งที่ต้องการ.
{ และ }	วงเล็บปีกกา	ใช้เป็นไวล์ดคาร์ด.
|	bar	ไปป์, ส่งข้อมูลต่อให้โปรแกรมอื่น
~	tilde	แทนโฮมไดเรกทอรี

ความแตกต่างระหว่าง single quote และ double quote ได้แก่ใน double quote เชลล์จะตีความอักขระพิเศษได้แก่ $, ‘, \ ด้วย. ถ้าใช้ single quote เชลล์จะถือว่าอักขระที่คร่อมด้วย single quote เป็นอักขระตามที่เขียน, ไม่มีความพิเศษสำหรับเชลล์.

Escape sequence

นอกจากการใช้ quote เพื่อให้เชลล์ตีความอักขระที่พิมพ์ตามที่พิมพ์แล้ว, เราสามารถใช้เครื่องหมาย backslash (\) นำหน้าอักขระพิเศษเพื่อไม่ให้เชลล์ตีความชั่วคราวได้ด้วย. เราเรียกเครื่อง backslash ที่ทำหน้าที่ยกเลิกความหมายพิเศษของอักขระพิเศษชั่วคราว

แบบนี้ว่า *escape sequence*. เราได้ใช้ escape sequence ไปแล้วในตัวอย่างที่ 2.52 (หน้า 52) เพื่อยกเลิก alias ชั่วคราว.

การใช้ escape sequence โดยให้ผลเหมือนตัวอย่างที่ 2.62 ได้แก่

ตัวอย่างที่ 2.63: การใช้ escape sequence

```
$ rm important\ note.txt↵
```

คำสั่งโดยทั่วไปจะมีความยาวไม่มากเป็นคำเดี่ยว ๆ, แต่ในบางครั้งถ้ามีการใช้ตัวเลือกและอาร์กิวเมนต์มาก ๆจะทำให้คำสั่งโดยรวมยาวอ่านลำบาก. ในกรณีที่ต้องการขึ้นบรรทัดใหม่เพื่อความสวยงามหรือสะดวกในการอ่าน, สามารถทำได้โดยใช้ escape sequence แล้วขึ้นบรรทัดใหม่ตามตัวอย่าง,

gcc อ้างอิงหน้า 408

*ตัวอย่างที่ 2.64: การแบ่งบรรทัดคำสั่งใหม่ด้วย *

```
$ gcc  -O -I./gc/include  -I/usr/include/openssl \↵
> -I/usr/include -I./libwc -I. -o myprogram \↵
> main.c file.c buffer.c display.c search.c \↵
> -L./libwc -lwc -L. -L/usr/lib -lssl -lcrypto -lpthread -lm↵
$
```

หลังจากที่พิมพ์ backslash (\) แล้วขึ้นบรรทัดใหม่, เชลล์จะแสดงพรอมต์ลำดับที่สอง “>” ให้เพื่อรับข้อมูลเป็นบรรทัด ๆต่อไป.

เครื่องหมายตรรกะ

เชลล์เป็นโปรแกรมแปลภาษาและนอกจากการสั่งคำสั่งธรรมดา, เราสามารถใช้ตรรกะในบรรทัดคำสั่งได้ด้วย. เครื่องหมาย “&&” แทน “และ (AND)” และเครื่องหมาย “||” แทน “หรือ (OR)”. เรามาดูตัวอย่างการสั่งคำสั่ง “ถ้ามีไฟล์ file.txt แล้วให้แสดงเนื้อหา”. เป็นการทดสอบดูว่าไฟล์ *file* มีตัวตนหรือไม่. คำสั่ง `test` จะไม่แสดงผลทางหน้าจอ, จะให้ผลลัพธ์เป็นค่าสถานการจบการทำงาน ($?).

test อ้างอิงหน้า 416

ตัวอย่างที่ 2.65: การใช้ตรรกะ AND ในบรรทัดคำสั่ง.

```
$ test -f file.txt && cat file.txt↵
```

ถ้าไฟล์ `file.txt` มีอยู่จริง, เชลล์ก็จะดำเนินการสั่งคำสั่ง `cat file.txt` ต่อไป เพราะคำสั่ง “`test -f file.txt`” มีค่าเป็นจริง (true). ค่าในที่นี้คือสถานะการจบการทำงานเป็น 0, ไม่มีข้อผิดพลาด. ในกรณีที่ไม่มีไฟล์ `file.txt`, เชลล์ก็จะไม่สั่งคำสั่ง `file.txt` ต่อเพราะคำสั่งแรกเป็นเท็จ (false) ไปแล้ว. วิธีการอย่างนี้เป็นการหลีกเลี่ยง stderr ที่แสดงบนหน้าจอจากคำสั่ง `cat` ถ้าไม่มีไฟล์ `file.txt`.

การใช้เครื่องหมาย “||” ก็คล้ายกับ “&&” แต่เป็นตรรกะแบบ OR. กล่าวคือถ้าคำสั่งแรกประสบผลสำเร็จก็จะไม่สั่งคำสั่งที่สอง, เพราะคำสั่งโดยรวมมีค่าเป็นจริงแล้ว. เราลองมาดูตัวอย่าง “ถ้าไม่มีไฟล์ foo ให้สร้างไฟล์ขึ้นมา” เขียนได้ดังตัวอย่างต่อไปนี้.

ตัวอย่างที่ 2.66: การใช้ตรรกะ OR ในบรรทัดคำสั่ง.

```
$ test -f foo || touch foo↵
```

2.5.15 การตรวจแก้บรรทัดคำสั่ง

การตรวจแก้บรรทัดคำสั่ง (command line editing) อย่างง่าย ๆทำได้โดยการเลื่อนเคอร์เซอร์ด้วยคีย์ลูกศร (←) (→) และการใช้ Backspace หรือ Delete ลบสิ่งที่ไม่ต้องการ.

เราลองมาดูตัวอย่างที่มักเกิดขึ้นจริงเช่นผู้ใช้พิมพ์คำสั่งยาว ๆและต้องการแก้ตัวอักษรที่อยู่ต้นบรรทัด. ในกรณีนี้ผู้ใช้อาจจะใช้คีย์ (←) เลื่อนไปต้นบรรทัดแล้วใช้ Backspace ลบตัวอักษรที่ต้องการ. แต่สำหรับผู้ใช้ที่คุ้นเคยกับเชลล์เป็นอย่างดีจะใช้วิธีลัดโดยการกดคีย์ Ctrl+a เลื่อนเคอร์เซอร์ไปที่ต้นบรรทัดแล้วใช้ Ctrl+f เลื่อนเคอร์เซอร์ทีละตัวอักษรไปตำแหน่งที่ต้องการลบ. แล้วใช้ Ctrl+d แทนคีย์ Delete ลบตัวอักษรที่ต้องการ.

บางครั้งการใช้คีย์ลูกษรเพื่อเลื่อนเคอร์เซอร์ไม่เร็วพอที่จะแก้ไขคำสั่งที่ยาว ๆเช่นกระโดดไปต้นบรรทัด, ลบอักษรหรือคำบางคำทีอยู่กลางบรรทัด, กระโดดกลับไปท้ายบรรทัด, หรือแม้กระทั่งย้อนกลับไปเรียกคำสั่งที่เคยสั่งไปแล้ว. การกระทำต่าง ๆเหล่านี้เรียกว่า*การตรวจแก้บรรทัดคำสั่ง (line editing)*. จะเห็นได้ว่าการตรวจแก้ไขบรรทัดคำสั่งจะคล้ายกับการแก้ไขเอกสารในบรรณาธิกรณ์, และ bash เชลล์ก็ใช้วิธีการแก้ไขบรรทัดคำสั่งเหมือนกับบรรณาธิกรณ์ที่เป็นที่นิยมได้แก่ emacs หรือ vi.

ตัวอย่างเช่นในบรรณาธิกรณ์ emacs เมื่อกดคีย์ Ctrl พร้อมกับ k จะหมายถึงการลบอักษรตั้งแต่ตำแหน่งเคอร์เซอร์ปัจจุบันไปจนสุดบรรทัด. ซึ่งใน bash เชลล์ก็ใช้การกดคีย์ชุดเดียวกัน. เมื่อกดคีย์เหล่านี้เชลล์จะกระทำการบางอย่างที่เตรียมไว้แล้ว, กรรมวิธีนี้เรียกว่า *key binding* คือผูกการกระทำอย่างใดอย่างหนึ่งไว้กับการกดคีย์ที่กำหนดไว้. ในหนังสือหรือเอกสารที่เกี่ยวกับ emacs มักแสดงการกดคีย์ Ctrl ร่วมกับคีย์อื่นเป็น C-. เช่น C-k หมายถึงการกดคีย์ Ctrl ค้างไว้แล้วกด k ตาม. นอกจากคีย์ Ctrl แล้วยังมีคีย์อื่น ๆได้แก่คีย์ Esc. key binding ที่ใช้ Esc จะมีวิธีการใช้แตกต่างจากคีย์ Ctrl คือกดคีย์ Esc ก่อน (ไม่ต้องกดค้าง) แล้วกดคีย์อื่นตาม. เขียนแทนด้วย M- เช่น M-f หมายถึงกดคีย์ Esc หนึ่งครั้งแล้วกดคีย์ f ตาม.

เราสามารถปรับแต่ง bash เชลล์เลือกวิธีการตรวจแก้บรรทัดคำสั่งได้ว่าจะให้เป็นแบบบรรณาธิกรณ์ emacs หรือ vi ด้วยคำสั่ง set. bash เชลล์จะใช้การแก้บรรทัดคำสั่งแบบ emacs โดยปริยายแต่ถ้าต้องการจะให้เป็นแบบ vi ก็สามารถแก้ไขได้ดังนี้.

ตัวอย่างที่ 2.67: การแก้วิธีตรวจแก้บรรทัดคำสั่งให้เป็นแบบ vi

```
$ set -o vi↵
```

ในที่นี้จะแนะนำการตรวจแก้บรรทัดคำสั่งแบบ emacs เท่านั้น.

ตารางที่ 2.8: key binding ที่ใช้บ่อยในการตรวจแก้บรรทัดคำสั่ง (แบบ emacs)

key binding	ชื่อคำสั่ง	การกระทำ
C-a	beginning-of-line	เลื่อนเคอร์เซอร์ไปต้นบรรทัด
C-e	end-of-line	เลื่อนเคอร์เซอร์ไปท้ายบรรทัด
C-f	forward-char	เลื่อนเคอร์เซอร์ไปข้างหน้าหนึ่งตัวอักษร
C-b	backward-char	เลื่อนเคอร์เซอร์ไปข้างหลังหนึ่งตัวอักษร
M-f	forward-word	เลื่อนเคอร์เซอร์ไปข้างหน้าหนึ่งคำ
M-b	backward-word	เลื่อนเคอร์เซอร์ไปข้างหลังหนึ่งคำ
C-l	clear-screen	ลบ สิ่ง ที่ แสดง อยู่ บน หน้า จอ หมด แล้ว ขึ้น เชลล์พรอมต์ที่บรรทัดแรก
C-p	previous-history	เรียกคำสั่งที่สั่งไปแล้วก่อนคำสั่งปัจจุบันหนึ่งคำสั่ง
C-n	next-history	เรียกคำสั่งหลังจากคำสั่งปัจจุบันหนึ่งคำสั่ง
M-<	beginning-of-history	เรียกคำสั่งแรกที่สั่งไป
M->	end-of-history	เรียกคำสั่งสุดท้ายที่สั่งไป
C-r	reverse-search-history	ค้นหาคำสั่งที่เคยสั่งไปแล้ว
C-s	forward-search-history	ค้นหาคำสั่งต่อไปจากคำสั่งปัจจุบัน
C-g	abort	ยกเลิกการตรวจแก้บรรทัดคำสั่ง
C-d	delete-char	ลบหนึ่งตัวอักษร
M-d	kill-word	ลบคำ “คำ” ในที่นี้หมายถึงตัวอักษรตั้งแต่เคอร์เซอร์ถึงตัวแบ่งคำซึ่งได้แก่ช่องไฟ
C-w	unix-word-rubout	ลบตัวอักษร(คำ)ที่อยู่ก่อนหน้าเคอร์เซอร์จนถึงตัวแบ่งคำ
C-k	kill-line	ลบ ตัว อักษร ตั้ง แต่ ตำแหน่งเ คอร์เ ซอร์ปัจจุบันจนถึงสุดบรรทัด
C-u	unix-line-discard	ลบ ตัว อักษร ตั้ง แต่ ตำแหน่งเ คอร์เ ซอร์ปัจจุบันจนถึงต้นบรรทัด
C-y	yank	นำตัวอักษรที่ลบ(เช่นจากคำสั่ง M-d, C-w, C-k, C-u)ไปแล้วกลับคืนมา

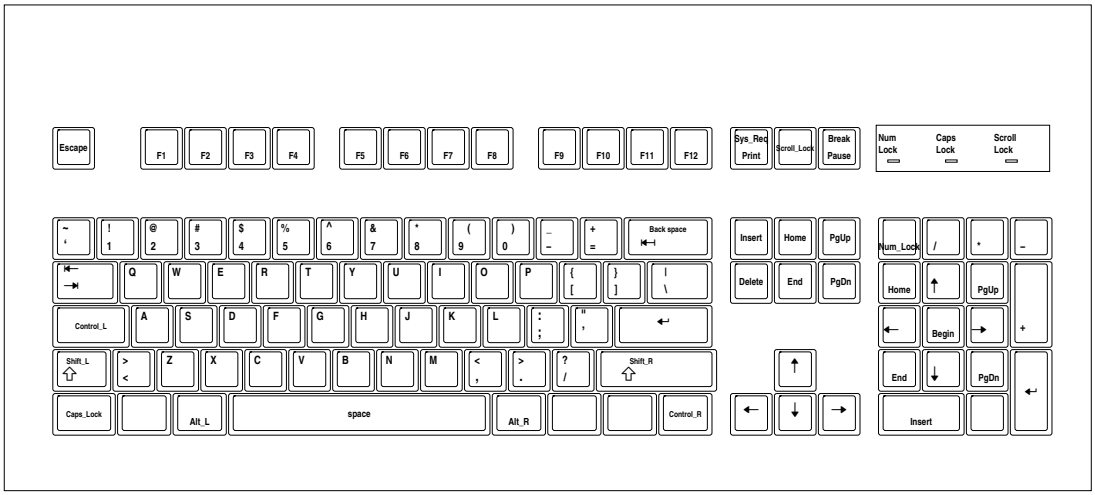
ต่อไปนี้เป็นตัวอย่างการตรวจแก้บรรทัดคำสั่งด้วย key binding ต่าง ๆที่แสดงในตารางที่ 2.8

ตัวอย่างที่ 2.68: การแก้ไขบรรทัดคำสั่งด้วย key binding ต่าง ๆ

```
$ echo This is a very long long command↵
This is a very long long command
$ ■
↓ Ctrl+p หรือ ↑ เรียกคำสั่งที่สั่งไปก่อนหน้านี้
$ echo This is a very long long command■
↓ Ctrl+a เลื่อนเคอร์เซอร์ไปต้นบรรทัด
$ echo This is a very long long command
↓ Esc-f เลื่อนเคอร์เซอร์ไปหนึ่งคำ
$ echo■ This is a very long long command
```

```
↓ Esc-d ลบคำหนึ่งคำ
$ echo is a very long long command
↓ พิมพ์เพิ่ม
$ echo That is a very long long command
↓ Enter กด Enter ได้เลยโดยไม่ต้องไปหลังบรรทัด
That is a very long long command
$
```

จะเห็นได้ว่า key binding ต่างๆที่ใช้จะต้องกดคีย์ Ctrl บ่อยมากแล้วต้องงอนิ้วก้อยของมือซ้ายกดทำให้ไม่สะดวก. จริงๆแล้วคีย์บอร์ดของเครื่องเวิร์กสเตชันยูนิกซ์จะต่างกับคีย์บอร์ดของเครื่องคอมพิวเตอร์ส่วนบุคคลคือตำแหน่งของคีย์ Ctrl กับคีย์ Caps Lock จะสลับกันตามรูปที่ 2.7. คีย์บอร์ดแบบนี้จะกดคีย์ Ctrl ได้ง่ายกว่าคีย์บอร์ดของคอมพิวเตอร์ส่วนบุคคลเพราะตำแหน่งของคีย์ Ctrl จะตรงกดนิ้วก้อยพอดี. วิธีการสลับตำแหน่งของคีย์ Ctrl กับคีย์ Caps Lock จะแนะนำในบทที่ ??.



รูปที่ 2.7: คีย์บอร์ดที่มี Control สลับกับ Caps Lock

2.5.16 การควบคุมเทอร์มินอล

เชลล์เป็นโปรแกรมที่กระทำการอยู่ในเทอร์มินอล, เพราะฉะนั้นจึงมีความจำเป็นที่ผู้ใช้ควรจะรู้จักการควบคุมเทอร์มินอลด้วยเพื่อที่จะใช้งานได้คล่องยิ่งขึ้น.

บางครั้งเราอาจเผลอกดคีย์ C-s ในเทอร์มินอลแล้วทำให้พิมพ์เทอร์มินอลค้างทำอะไรต่อไม่ได้. key binding เช่น C-s ที่กล่าวถึงนี้ไม่เกี่ยวกับเชลล์แต่เป็นคีย์ที่เทอร์มินอลรับรู้และหยุดแสดงผลทางหน้าจอชั่วคราว. เนื่องจากจอจะหยุดนิ่งไม่ว่าเราพิมพ์อะไรก็ตามทำให้ดูเหมือนเทอร์มินอลนั้นไม่ทำงาน. ผู้ใช้สามารถยกเลิกการหยุดแสดงผลได้โดยการกดคีย์ C-q. สิ่งที่เราพิมพ์ระหว่างการหยุดการแสดงผลก็จะปรากฏหลังจากกด C-q. ตัวอย่างการใช้ C-s กับ C-q เช่นเมื่อสั่งคำสั่งที่แสดงผลยาวมากๆและสิ่งที่แสดง

ผ่านไปอย่างรวดเร็วดูไม่ทันก็สามารถใช้ C-s หยุดดูผลชั่วคราวแล้วกด C-q ให้แสดงผลต่อไป.

เทอร์มินอลทั่วไปมักจะมี key binding อื่น ๆอีกเช่น Shift+PgUp เลื่อนหน้าจอขึ้นแสดงผลที่แสดงไปแล้ว, Shift+PgDn เลื่อนหน้าจอลง. คีย์เหล่านี้อำนวยความสะดวกตรงที่ไม่ต้องใช้เมาส์เลื่อน scroll ของเทอร์มินอลเอมิวเลเตอร์ก็สามารถใช้คีย์บอร์ดแทนได้ซึ่งจะเร็วกว่า. ถ้าเราใช้ลินุกซ์คอนโซล, Shift+PgUp มีประโยชน์มากเพราะใช้เมาส์เลื่อนหน้าจอไม่ได้เหมือนเทอร์มินอลเอมิวเลเตอร์.

บางครั้งผู้ใช้อาจจะสั่งคำสั่งที่แสดงตัวอักษรที่อ่านไม่ได้หรือไฟล์แบบไบนารีออกทางหน้าจอแล้วทำให้พรอมท์กลายเป็นอักษรประหลาดอ่านไม่ออก. ในกรณีนี้ก็ให้พิมพ์คำสั่ง reset ทั้ง ๆที่อ่านไม่ออกไปเลย. คำสั่งนี้จะรีเซ็ตเทอร์มินอลทำให้เทอร์มินอลอยู่ในสภาพปรกติ. คำสั่งที่ใช้ควบคุมเทอร์มินอลอื่น ๆได้แก่ tset, stty, clear, tput เป็นต้น.

reset อ้างอิงหน้า 414

2.5.17 ประวัติคำสั่ง

เชลล์มีความความสามารถช่วยผู้ใช้ในการจำคำสั่งที่สั่งไปแล้วที่เรียกว่า*ประวัติคำสั่ง (history)*. ประวัติการสั่งคำสั่งมีประโยชน์ช่วยลดภาระในการพิมพ์คำสั่งโดยเฉพาะคำสั่งที่ยาว ๆไม่ต้องพิมพ์คำสั่งทั้งหมดใหม่. ผู้ใช้อาจจะกดคีย์บางคีย์เพื่อเรียกคำสั่งที่เคยสั่งไปแล้วในอดีตมาใช้อีกครั้ง.

แนวคิดของการจำประวัติคำสั่งในเชลล์จะคล้ายกับการบันทึกคำสั่งบรรทัดต่อบรรทัดลงในไฟล์. คำสั่งทุกคำสั่งจะมีหมายเลขประวัติคำสั่งกำกับ. หมายเลขประวัติคำสั่งจะเก็บในตัวแปรเชลล์ชื่อ HISTCMD. และเชลล์มีคำสั่งภายใน history เอาไว้แสดงประวัติคำสั่งที่สั่งไปแล้ว.

history อ้างอิงหน้า ??

ตัวอย่างที่ 2.69: คำสั่ง history

```
$ history 4↵
 1005  ls
 1006  man bash
 1007  help history
 1008  history 4
$ echo $HISTCMD↵ ← เลขประวัติคำสั่งของคำสั่งนี้เป็น 1009
1010 ← หมายเลขประวัติคำสั่งของคำสั่งที่จะสั่งต่อไป
$
```

จากตัวอย่างข้างบน, "history 4" หมายถึงแสดงประวัติการสั่งคำสั่งรวมถึงคำสั่งที่กำลังสั่งอยู่ด้วย 4 บรรทัด. จำนวนคำสั่งที่เชลล์จะจำนั้นขึ้นอยู่กับค่าที่เราตั้งซึ่งโดยปริยายเชลล์จะจำคำสั่งในประวัติคำสั่งไว้ 500 คำสั่ง.

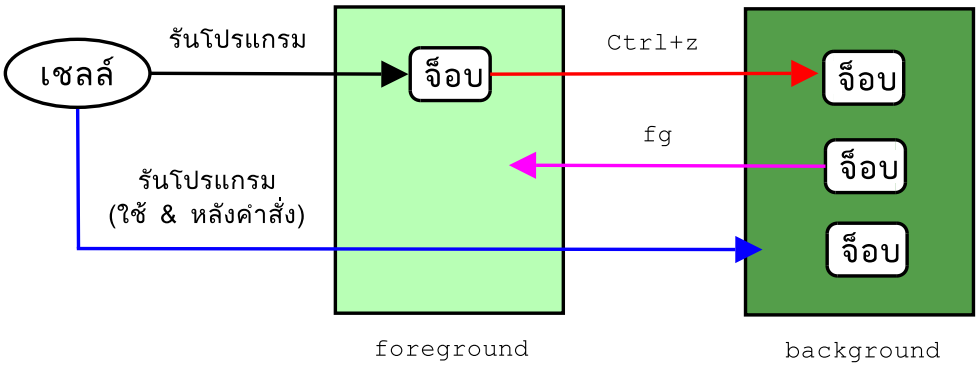
การดูคำสั่งที่สั่งไปแล้วไม่เกิดประโยชน์อะไรถ้าไม่สามารถเรียกใช้ได้และการเรียกใช้คำสั่งที่สั่งไปแล้วต้องรวดเร็วด้วยจึงจะสะดวก. เราดูตัวอย่างที่ผ่านมาแล้วเกี่ยวกับการเรียกคำสั่งที่แล้วมาใช้อีกโดยการกดคีย์ C-p หรือ ↑. สมมติว่าคำสั่งที่สั่งไปแล้วและเราต้องการเรียกกลับมาใช้เป็นคำสั่งที่สั่งไปนานมากแล้ว, ผู้ใช้ไม่ต้องกดคีย์ C-p หลายครั้งเพราะเชลล์สามารถหาคำสั่งในประวัติคำสั่งได้ด้วยคีย์ C-r. ถ้าต้องการยกเลิกการค้นหากลางครรให้ใช้คีย์ C-g.

ตัวอย่างที่ 2.70: การค้นหาคำสั่งที่เคยสั่งไปแล้วด้วย C-r

```
$ ■ Ctrl+r
↓
(reverse-i-search)'': ■
↓ หาคำในประวัติคำสั่งที่มี ac อยู่
(reverse-i-search)'ac': /usr/local/Acrobat5/bin/acroread linux.pdf
↓ กด Ctrl+r เพื่อหาคำสั่งที่มี ac ต่อไป
(reverse-i-search)'ac': emacs /home/poonlap/.bashrc &
↓ ถ้าต้องการรันคำสั่งก็กด Enter ได้ทันที. หรือจะแก้ไขบรรทัดคำสั่งก่อนก็ได้.
↓ กด Enter เพื่อรันคำสั่งนี้.
$ emacs /home/poonlap/.bashrc &
[7] 10338
$ ■
```

2.5.18 จ็อบและการควบคุมคำสั่ง

จากรูปวงจรการรันคำสั่งของเชลล์, จะเห็นว่าผู้ใช้ไม่สามารถสั่งคำสั่งต่อไปได้ถ้าคำสั่งที่สั่งไปก่อนหน้านี้ยังทำงานไม่เสร็จสมบูรณ์. ในกรณีนี้เราเรียกคำสั่ง (หรือโปรแกรม) ที่กำลังทำงานอยู่ว่า*จ็อบแบบ foreground (foreground job)*. ในทางกลับกัน, คำสั่งที่กระทำการอยู่และเชลล์สามารถสั่งคำสั่งต่อไปได้ทันทีโดยที่ไม่ต้องรอคำสั่งที่กระทำการอยู่จบการทำงานเรียกว่า*จ็อบแบบ background (background job)*. วิธีการสั่งคำสั่งให้เป็นจ็อบแบบ background ทำได้โดยใส่เครื่องหมาย "&" ท้ายคำสั่ง.



รูปที่ 2.8: ความสัมพันธ์ระหว่างจ็อบแบบ foreground และ background

ตัวอย่างต่อไปนี้เป็นการหาไฟล์ที่ชื่อ core ที่อยู่ใต้*รูทไดเรกทอรี (root directory)* ด้วยคำสั่ง `find`. คำสั่ง `find` จะแสดงชื่อไฟล์ที่หาเจอและ*ข้อผิดพลาด (error)* ทางหน้าจอ. ในตัวอย่างจะแสดงวิธีเก็บผลที่แสดงบนหน้าจอลงในไฟล์ชื่อ `result.txt` และแยกบันทึกข้อผิดพลาดลงในไฟล์ `error.txt`.

core ▶
core หรือเรียกอีกอย่างว่า core dump เป็นไฟล์ที่บันทึกเนื้อหาของหน่วยความจำเวลาเวลาโปรแกรมทำงานล้มเหลว (crash). ผู้พัฒนาสามารถใช้ข้อมูลที่ได้จากไฟล์นี้ในการหาข้อบกพร่องของโปรแกรมได้.

ตัวอย่างที่ 2.71: การสั่งคำสั่งแบบ background

```
$ find / -name core > result.txt 2> error.txt &↵
[1] 6514 ← หมายเลขจ็อบ (1) และหมายเลขโปรเซส (6514)
$ ■ ← สามารถสั่งคำสั่งอื่นต่อได้ทันที, ไม่ต้องรอให้ find ทำงานเสร็จ
```

การหาไฟล์ตั้งแต่รูทไดเรกทอรีตามตัวอย่างที่แสดงจะกินเวลานานกว่าจะจบทำให้ไม่สามารถสั่งคำสั่งอื่นได้ทันที, การสั่งคำสั่งแบบ background จะช่วยให้ผู้ใช้สั่งคำสั่งที่ต้องการต่อไปได้ทันทีโดยที่คำสั่ง `find` ยังทำงานอยู่เบื้องหลัง. เชลล์จะแสดงหมายเลข*จ็อบ (job)* และหมายเลข*โปรเซส (process)* หลังจากที่สั่งคำสั่งแบบ background. ผู้ใช้สามารถควบคุมคำสั่งโดยอาศัยหมายเลขจ็อบหรือหมายเลขโปรเซส. หมายเลขจ็อบกับหมายเลขโปรเซสต่างกันที่หมายเลขจ็อบเป็นหมายเลขที่ออกให้โดยเชลล์ที่สั่งคำสั่งนั้น ๆ, ส่วนหมายเลขโปรเซสเป็นหมายเลขเฉพาะที่ออกโดยตัวระบบปฏิบัติการไว้อ้างอิงโปรแกรมที่กระทำการอยู่.

ในกรณีที่ผู้ใช้ไม่ได้ใส่เครื่องหมาย “&” เวลาสั่งคำสั่ง (จ็อบแบบ foreground) และต้องการทำให้คำสั่งที่สั่งไปแล้วเป็นจ็อบแบบ background, ผู้ใช้ต้องหยุดจ็อบนั้นก่อนชั่วคราวโดยการส่ง*สัญญาณ (signal)* SIGSTOP ด้วยการกดคีย์ `C-z`. หลังจากนั้นสั่งคำสั่ง “bg” เพื่อเปลี่ยนสภาพจ็อบที่หยุดชั่วคราวให้ทำงานแบบ background.

ในที่นี้จะเรียกคำสั่งที่กำลังทำงานอยู่ว่า “จ็อบ” เนื่องจากช่วงนี้เป็นเนื้อหาที่เกี่ยวกับเชลล์เป็นส่วนใหญ่. จริง ๆ แล้วจะเรียกว่าโปรเซสก็ได้แล้วแต่มุมมองโปรแกรมที่กำลังกระทำการอยู่ว่ามองจากเชลล์หรือมองจากระบบปฏิบัติการ.

bg อ้างอิงหน้า 420

ตัวอย่างที่ 2.72: การสั่งคำสั่งแบบ foreground ก่อนแล้วเปลี่ยนเป็น background

```
$ find / -name core > result.txt 2> error.txt↵
^Z (Ctrl+Z) ← ส่งสัญญาณ SIGSTOP โดยการกดคีย์
[1]+  Stopped                 find / -name core >result.txt 2>error.txt
$ bg↵
[1]+ find / -name core >result.txt 2>error.txt &
$
```

jobs อ้างอิงหน้า 422

คำสั่ง `jobs` เป็นคำสั่งภายในเชลล์ใช้ในแสดงรายการจ็อบและสภาพของจ็อบที่เชลล์นั้น ๆกระทำการอยู่.

ตัวอย่างที่ 2.73: การใช้คำสั่ง `jobs` ดูโปรแกรมที่ทำงานอยู่ในเชลล์

```
$ jobs↵
[1]   Running                 find / -name core >result.txt 2>error.txt &
[2]+  Stopped                 emacs
[3]-  Running                 mozilla &
$
```

จากตัวอย่างข้างบนจะเห็นว่าในเชลล์ที่ใช้อยู่มีจ็อบ 3 รายการ, โปรแกรม `find`, `mozilla` ซึ่งกำลังกระทำการอยู่และโปรแกรม `emacs` หยุดทำงานชั่วคราว.

fg อ้างอิงหน้า 422

คำสั่งที่ตรงข้ามกับ bg ได้แก่คำสั่ง `fg`. คำสั่ง `fg` จะทำให้จ็อบที่หยุดชั่วคราวอยู่กลับมาเป็นจ็อบแบบ foreground อีกครั้ง, หรือทำให้จ็อบแบบ background เปลี่ยนมาเป็นจ็อบแบบ foreground. ตัวอย่างต่อไปนี้แสดงการทำให้จ็อบหมายเลข 1 เปลี่ยนสภาพจาก background เป็น foreground. “%1” เป็นวิธีการเจาะจงหมายจ็อบ. ถ้าไม่เจาะจงหมายเลขจ็อบแล้วสั่งคำสั่ง `fg` หรือ bg เดี่ยว, จะถือว่าจ็อบที่มีเครื่องหมาย “+” เวลาแสดงรายการจ็อบเป็นเป้าหมาย.

ตัวอย่างที่ 2.74: การใช้ `fg` เปลี่ยนจ็อบจาก background ให้เป็น foreground

```
$ fg %1↵
find / -name core >result.txt 2>error.txt
                                  ← กลับมาสู่ foreground อีกครั้ง
```

2.6 คู่มือการใช้งาน

โปรแกรมใช้งานในลินุกซ์มีมากมายให้เลือกใช้. ปัญหาอย่างหนึ่งของผู้ใช้ใหม่คือรู้ว่าต้องการทำอะไรบางอย่างแต่ไม่รู้ว่าควรจะใช้โปรแกรมหรือคำสั่งอะไร. ปัญหาอีกอย่างคือรู้คำสั่งแต่ไม่รู้ว่าคำสั่งนั้นทำอะไรได้บ้างนอกเหนือจากที่สิ่งที่ตัวเองใช้เป็น. ผู้ใช้เริ่มต้นไม่ควรกลัวในการทดลองสั่งคำสั่ง. แต่ก่อนที่จะทดลองควรจะดูคู่มือใช้งานที่มากับคำสั่งหรือโปรแกรมที่จะใช้. จะทำให้ลดความอันตรายต่อระบบหรือป้องกันผลลัพธ์ที่ไม่คาดคิดได้.

หลังจากที่แนะนำการใช้เชลล์เบื้องต้นแล้ว, ในช่วงนี้จะแนะนำการใช้หาข้อมูลต่าง ๆ เช่นเอกสารเกี่ยวกับโปรแกรมที่ต้องการใช้, คู่มือการใช้งาน, แหล่งข้อมูลที่เกี่ยวข้อง ฯลฯ เพื่อเตรียมความพร้อมในการแก้ปัญหาด้วยตัวเองและเป็นพื้นฐานสำหรับเนื้อหาที่จะแนะนำต่อไป.

2.6.1 คู่มือใช้งาน

คำสั่งมาตรฐานของลินุกซ์และโปรแกรมโดยทั่วไปมักจะมี*คู่มือใช้งาน (manual)* ติดมากับตัวโปรแกรมด้วย. สำหรับโปรแกรมแบบ CLI มันจะมีคู่มือการใช้งานในรูปของ *on-line manual* หรือเรียกง่าย ๆว่า *man page (manual page)*. โปรแกรมแบบ GUI อาจจะมี man page ด้วยและมีเมนูช่วยเหลือ (Help menu) ให้คลิ้กจากหน้าต่างหลักของโปรแกรม.

Online manual

การอ่านคู่มือแบบ man page จะใช้คำสั่ง `man` โดยที่มีชื่อโปรแกรมหรือคำสั่งที่ต้องการดูคู่มือเป็นอาร์กิวเมนต์. ตัวอย่างต่อไปนี้เป็นการดูคู่มือการใช้งานของคำสั่ง `man`.

man อ้างอิงหน้า 414

ตัวอย่างที่ 2.75: ดูคู่มือการใช้งานด้วยคำสั่ง man

```
$ man man↵                          ← โปรแกรมจะเคลียร์หน้าจอแล้วแสดงคู่มือใช้งานเต็มหน้า
man(1)                                                          man(1)


NAME
       man - format and display the on-line manual pages
       manpath - determine user's search path for man pages

SYNOPSIS
       man  [-acdfFhkKtwW]  [--path]  [-m system] [-p string]
       [-C config_file] [-M pathlist] [-P pager] [-S section_list]
       [section] name ...


DESCRIPTION
       man formats and displays the on-line manual pages.  If you specify
       section,  man  only looks in that section of the manual.  name is
       normally the name of the manual page, which is typically the name
       of a  command, function,  or  file.  However,  if  name contains
       a slash (/) then man interprets it as a file specification, so that
       you can do  man  ./foo.5 or even man /cd/foo/bar.1.gz.
```

```
        See  below  for  a  description  of where man looks for the manual
:■                                ← พรอมต์ของเพจเจอร์ (less) สำหรับควบคุมหน้าจอดูคู่มือ
```

โปรแกรม man จะแสดงคู่มือที่เก็บอยู่ในรูปของไฟล์แบบ *roff*. ข้อมูลของคู่มือจะเก็บเป็นไฟล์ไว้ใต้ไดเรกทอรีที่กำหนดไว้ในตัวแปรสภาพแวดล้อม MANPATH. โปรแกรม man จะประมวลผลจัดหน้าจอแล้วให้โปรแกรม*เพจเจอร์ (pager)* เช่น less หรือ more ช่วยควบคุมหน้าจอต่อ.

จากตัวอย่างที่ 2.75 บรรทัดแรกของ man page ได้แก่ชื่อของโปรแกรมที่ต้องการดูคู่มือและมีตัวเลขในวงเล็บต่อท้าย, man(1). ตัวเลขนี้คือหมวดหมู่ (section) ของคู่มือแบ่งไว้ตามตารางที่ 2.9. ตัวเลขหมวดมีประโยชน์เวลาคู่มือที่ต้องการหามีชื่อซ้ำกันแต่อยู่คนละหมวดเช่น printf เป็นทั้งชื่อคำสั่งของยูสเซอร์ทั่วไปและในขณะเดียวกัน printf ก็เป็นชื่อฟังชันของไลบรารีในภาษา C ด้วย. เพื่อที่จะแยกแยะความแตกต่าง, เอกสารบางฉบับจะเขียน printf(1) และ printf(3) เพื่อความชัดเจน.

ตารางที่ 2.9: หมวดหมู่ของ man page

หมวดหมู่ที่	คำอธิบาย
1	คู่มือเกี่ยวกับคำสั่งหรือโปรแกรมสำหรับผู้ใช้ทั่วไป.
2	คู่มือเกี่ยวกับ system call สำหรับเขียนโปรแกรมติดต่อกับระบบปฏิบัติการ.
3	คู่มืออธิบายฟังชันของไลบรารีต่าง ๆสำหรับการเขียนโปรแกรม.
4	คู่มืออธิบายเกี่ยวกับไฟล์พิเศษเช่นไฟล์ที่อยู่ในไดเรกทอรี /dev.
5	คู่มืออธิบายเกี่ยวกับฟอร์แมต, รูปแบบของไฟล์เช่นไฟล์สำหรับปรับแต่งระบบ.
6	คู่มือสำหรับเกมส์หรือโปรแกรมสนุกสนานอื่น ๆ.
7	คู่มืออธิบายเกี่ยวกับข้อตกลง, มาตรฐาน, โปรโตคอล, ข้อกำหนดต่าง ๆ.
8	คู่มือเกี่ยวกับคำสั่งหรือโปรแกรมสำหรับผู้ดูแลระบบ.
9	คู่มืออื่น ๆที่เจาะจงสำหรับเคอร์เนล.
n	เอกสารใหม่ (new document) เช่นฟังชันของภาษา Tcl/Tk. (ไม่ค่อยใช้)
o	เอกสารเก่า (old document) (ไม่ค่อยใช้)
l	เอกสารท้องถิ่น (local document) เฉพาะแต่ละระบบ. (ไม่ค่อยใช้)

ผู้อ่านสามารถอ่านคำแนะนำของแต่ละหมวดจากลินุกซ์ที่ใช้อยู่ได้ดังนี้

ตัวอย่างที่ 2.76: การอ่านคำแนะนำของ man page หมวดที่ 1

```
$ man 1 intro↵
```

ต่อจากตัวอย่างที่ 2.75, ในคู่มือที่แสดงจะแบ่งเป็นช่วง ๆหลักได้แก่ ชื่อ (NAME), สรุปความสำคัญ (SYNOPSIS), คำอธิบาย (DESCRIPTION), อธิบายตัวเลือก (OPTION-

ในความเป็นจริงแล้วจะมีการบีบอัดไฟล์คู่มือด้วย.

roff ▶ ระบบประมวลผลเอกสารที่พัฒนาบนระบบปฏิบัติการยูนิกซ์ตั้งแต่เริ่มต้น. ฟอร์แมตของเอกสารนี้ในปัจจุบันเป็นฟอร์แมตของ man page ที่ใช้กันทั่วไป. ต่อมามีการพัฒนาหลายแขนงได้แก่ nroff, troff, groff เป็นต้น.

ผู้ใช้สามารถกำหนดไดเรกทอรีที่มีเก็บไฟล์คู่มือได้ด้วยตัวแปรสภาพแวดล้อม MANPATH และเลือกโปรแกรมเพจเจอร์ได้โดยการตั้งค่าตัวแปรสภาพแวดล้อม MANPAGER.

pager ▶ *เพจเจอร์*. โปรแกรมที่ใช้แสดงข้อมูลเท็กซ์ทางหน้าจอเทอร์มินอลเป็นหน้า ๆ. มีความสามารถเลื่อนข้อมูลตามต้องการและมีความสามารถค้นหาคำที่ต้องการได้. เพจเจอร์ที่นิยมใช้ได้แก่ less, more, lv ฯลฯ.

more เป็นโปรแกรมเพจเจอร์ทมีมาก่อน less. เนื่องจาก less มีความสามารถเหนือกว่า more, ในปัจจุบันจึงนิยมใช้ less เป็นเพจเจอร์โดยปริยาย.

man อ้างอิงหน้า 414

S), ตัวแปรสภาพแวดล้อมที่เกี่ยวข้อง (ENVIRONMENT), ไฟล์ปรับแต่ง (FILES), ข้อมูลอื่น ๆที่เกี่ยวข้อง (SEE ALSO) ฯลฯ.

ในช่วงสรุปความสำคัญเป็นช่วงสำคัญที่อธิบายว่าโปรแกรมที่ต้องการใช้นั้นมีไวยกรณ์, ตัวเลือกอย่างไร. เราสามารถรู้ได้ว่าตัวเลือกใดบังคับหรือไม่บังคับ, ตัวเลือกใดรวมกันได้, เรารู้ได้จากช่วงนี้. จากตัวอย่างที่ 2.75, คำที่อยู่ใน bracket (`[]`) หมายความว่าไม่ใช่เป็นส่วนที่บังคับ เช่น `-acdfFhkKtwW` เป็นตัวเลือกที่ไม่บังคับ, ไม่ต้องพิมพ์ในบรรทัดคำสั่งก็ได้. ส่วนอาร์กิวเมนต์บังคับจะไม่อยู่ใน bracket (`[]`).

วิธีการเขียนแบบ `[-acdfFhkKtwW]` เป็นการแสดงตัวเลือกที่รวมกันได้ไว้ด้วยกัน, หมายความว่าคำสั่ง `man` นี้มีตัวเลือก `-a`, `-c` ฯลฯ แยกกัน. ส่วนตัวเลือกเหล่านี้มีความหมายอย่างไร, ใช้ทำอะไรต้องไปดูที่ช่วงอธิบายตัวเลือกที่อยู่ข้างล่างต่อไป. ส่วนตัวเลือกเช่น `[-m system]` เป็นตัวเลือกที่ต้องการค่าประกอบตามเป็นอาร์กิวเมนต์ของตัวเลือก. คือจะใช้ตัวเลือก `-m` เดี่ยว ๆไม่ได้. ส่วนค่าประกอบนั้นคืออะไรต้องไปอ่านที่ช่วงอธิบายตัวเลือกข้างล่างต่อไป.

ตัวเลือก `[section]` เป็นตัวเลือกแบบไม่ต้องมีเครื่องหมาย `-` นำหน้า. ตรงนี้เป็นที่ใส่ชื่อหมวดของคู่มือที่ต้องการดู, ไม่จำเป็นต้องใส่ก็ได้. ในช่วงสรุปความสำคัญสุดท้ายเป็น `...` หมายความว่าซ้ำได้ต่อไป. กล่าวคือใส่ชื่อคู่มือที่ต้องการดูได้หลายอันในทีเดียว เช่น “`man cp rm mkdir`” เป็นต้น.

นอกจากช่วงสรุปความสำคัญแล้ว, ช่วงที่สำคัญอื่น ๆได้แก่ช่วง ENVIRONMENT ซึ่งอธิบายตัวแปรสภาพแวดล้อมที่มีผลกระทบต่อโปรแกรมนั้น ๆ, ช่วง FILES ซึ่งบอกที่อยู่และชื่อของไฟล์ปรับแต่ง และ SEE ALSO ซึ่งแนะนำโปรแกรมอื่น ๆที่ทำหน้าที่คล้าย ๆกันหรือเกี่ยวข้องกัน.

ตามที่ได้แนะนำไปแล้วว่าคำสั่ง `man` เป็นโปรแกรมนำข้อมูลมาประมวลผลจัดแต่งหน้าจอแล้วส่งให้เพจเจอร์เป็นตัวแสดงผลทางเทอร์มินอล. ในช่วงนี้จะแนะนำการใช้เพจเจอร์ซึ่งได้แก่ `less` โดยสังเขป. นอกจากจะใช้ `less` ดู man page แล้ว, เรายังสามารถใช้ `less` ดูข้อมูลยาว ๆที่ไม่สามารถแสดงได้ในหน้าจอเทอร์มินอลหน้าจอเดียวได้ด้วย.

จากตัวอย่างที่ 2.75 ตรงบรรทัดสุดท้าย, “`:■`” เป็นพรอมต์ที่ระบุว่าสิ่งที่แสดงบนหน้าจอ (ในกรณีนี้คือ man page) ยังไม่จอ. และรอรับคำสั่งจากคีย์บอร์ดอยู่. ตารางที่ 2.10 สรุป key binding ที่ใช้ควบคุมการทำงานของโปรแกรม `less`. Key binding เหล่านี้จะเหมือนกับบรรณาธิกรณ์ `vi`.

key binding ▶ การสร้างความสัมพันธ์ของโปรแกรมระหว่างคีย์ที่ผู้ใช้กดและการกระทำของโปรแกรมที่เตรียมไว้.

คำสั่ง `man` มีตัวเลือก `-k` สำหรับค้นหา man page ด้วยคีย์เวิร์ดที่ต้องการ. แต่ก่อนที่ตัวเลือกนี้จะทำงานถูกต้อง, ผู้ดูแลระบบจะต้องสร้างฐานข้อมูลสำหรับค้นหาด้วยคีย์เวิร์ดก่อนด้วยคำสั่ง `makewhatis`. ถ้าไม่สร้างฐานข้อมูลไว้ก่อนอาจจะใช้ตัวเลือก `-K` เพื่อค้นหา man page ได้เช่นกัน แต่จะช้ากว่าและไม่สะดวกในการใช้งาน.

▢ `makewhatis` อ้างอิงหน้า 406

คำสั่งที่ใช้ค้นหาคู่มือด้วยคีย์เวิร์ดนอกจาก `man -k` แล้วยังมีคำสั่ง `whatis` และ `apropos`. คำสั่งเหล่านี้จะใช้ฐานข้อมูลที่สร้างด้วยคำสั่ง `makewhatis` ร่วมกัน. คำสั่ง `apropos` จะให้ผลเหมือนคำสั่ง `man -k`, ส่วนคำสั่ง `whatis` จะหาชื่อคำสั่งที่ตรงกับคีย์เวิร์ดที่ต้องการเท่านั้น. การใช้คำสั่งเหล่านี้จะช่วยให้ผู้ใช้รู้จักคำสั่งมากขึ้น.

▢ `whatis` อ้างอิงหน้า 418

▢ `apropos` อ้างอิงหน้า 409

ตารางที่ 2.10: key binding ของโปรแกรม less

คีย์	คำอธิบาย
Space หรือ PgDn	เลื่อนเนื้อหาต่อไป(ลง)หนึ่งหน้า
b หรือ PgUp	เลื่อนหน้ากลับ(ขึ้น)หนึ่งหน้า
j หรือ Enter หรือ ↓	เลื่อนเนื้อหาต่อไป(ลง)หนึ่งบรรทัด
k หรือ ↑	เลื่อนเนื้อหากลับ(ขึ้น)หนึ่งบรรทัด
q	เลิกการทำงาน
h	แสดงหน้าจอช่วยเหลือ (help)
/	เริ่มการค้นหาคำ. หลังจากที่กดคีย์นี้แล้วให้พิมพ์คำที่ต้องการหาต่อแล้วกด Enter
?	เริ่มการค้นหาแบบย้อนหลัง. หลังจากที่กดคีย์นี้แล้วให้พิมพ์คำที่ต้องการหาต่อแล้วกด Enter
n	หาคำที่ต้องการหาต่อไป (next)
1G	กระโดดไปบรรทัดแรก (หน้าแรก)
G	กระโดดไปบรรทัดสุดท้าย (หน้าสุดท้าย)

ตัวอย่างที่ 2.77: หาคำสั่งที่ต้องการใช้ด้วยคีย์เวิร์ด

```
$ man -k fstab↵
endfsent [getfsent]  (3)  - handle fstab entries
fstab                (5)  - static information about the filesystems
getfsent             (3)  - handle fstab entries
getfsfile [getfsent] (3)  - handle fstab entries
getfsspec [getfsent] (3)  - handle fstab entries
nfs                  (5)  - nfs fstab format and options
setfsent [getfsent]  (3)  - handle fstab entries
```

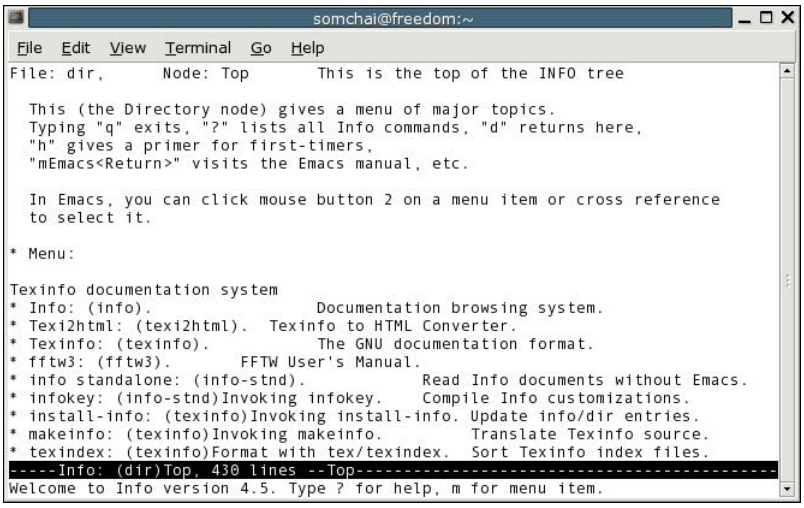
GNU info

info อ้างอิงหน้า 410

info เป็นคำสั่งที่ใช้แสดงคู่มือการใช้งานที่เก็บเป็นไฟล์ฟอร์แมต info. เอกสารแบบนี้มักเป็นโปรแกรมของโครงการ GNU เช่น emacs, bash. ระบบจัดการเอกสาร GNU info จะแตกต่างจาก man page ตรงที่ว่าเป็นระบบที่ยืดหยุ่นกว่าและสามารถสร้างเอกสารที่มีโครงสร้างซับซ้อน, มีเนื้อหาที่ละเอียดกว่า. ลักษณะของเอกสารจะเหมือนหนังสือมากกว่า man page ที่เป็นคู่มืออธิบายการใช้งานหน้าเดียว.

คำสั่ง info โดยไม่มีตัวเลือกจะเข้าสู่หน้าหลักของเอกสาร info ซึ่งมีรายการคู่มือเอกสารต่าง ๆให้เลือกต่อไป. หรือจะใช้ชื่อโปรแกรมที่ต้องการดูคู่มือเป็นอาร์กิวเมนต์ก็ได้. การอ่านคู่มือด้วย info จะคล้ายกับการใช้บราวเซอร์ (browser). มีการกระโดดข้าม node, กลับไปมาได้โดยใช้คีย์ลูกศรและ Enter. โปรแกรม info จะมี key binding แบบบรรณาธิกรณ์ emacs ซึ่งสรุปไว้ในตารางที่ 2.11.

node ของ info จะเหมือนกับบทหรือส่วนย่อยในหนังสือ.



รูปที่ 2.9: โปรแกรม `info` ใน `gnome-terminal`.

ตารางที่ 2.11: key binding ของโปรแกรม `info`

คีย์	คำอธิบาย
Space หรือ PgDn	เลื่อนเนื้อหาต่อไป(ลง)หนึ่งหน้า
BackSpace หรือ PgUp	เลื่อนหน้ากลับ(ขึ้น)หนึ่งหน้า
C-n หรือ ↓	เลื่อนเนื้อหาต่อไป(ลง)หนึ่งบรรทัด
C-p หรือ ↑	เลื่อนเนื้อหากลับ(ขึ้น)หนึ่งบรรทัด
q	เลิกการทำงาน
C-h หรือ ?	แสดงหน้าจอช่วยเหลือ (help)
C-s หรือ /	เริ่มการค้นหาคำ.
C-r	เริ่มการค้นหาแบบย้อนหลัง.
M-<	กระโดดไปบรรทัดแรก (หน้าแรก)
M->	กระโดดไปบรรทัดสุดท้าย (หน้าสุดท้าย)

คู่มือการใช้งานแบบ GUI

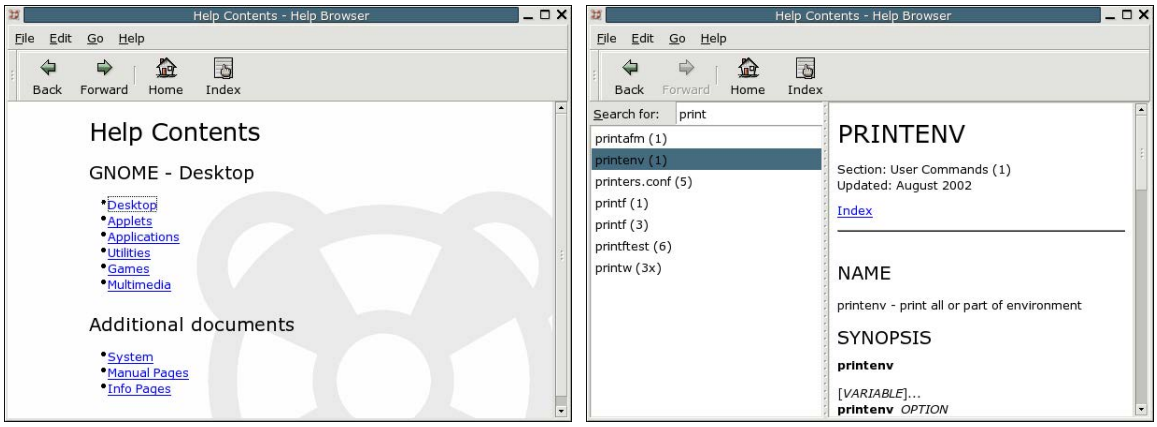
ในระบบ desktop environment เช่น GNOME หรือ KDE จะมี Help browser หรือ Help center เตรียมไว้สำหรับผู้ใช้. คู่มือแบบ GUI นี้นอกจากดู Help ของโปรแกรมที่เกี่ยวกับ GNOME และ KDE แล้วยังใช้ดู man page และ info ได้ด้วย. นอกจากนี้ยังมีหน้าจอหาคู่มือ, อธิบายคำศัพท์ ฯลฯ. Help browser ของ GNOME ได้แก่โปรแกรม `yelp`, ส่วน Help center ของ KDE ได้แก่โปรแกรม `khelpcenter`.

`yelp`
Help browser ของ GNOME.

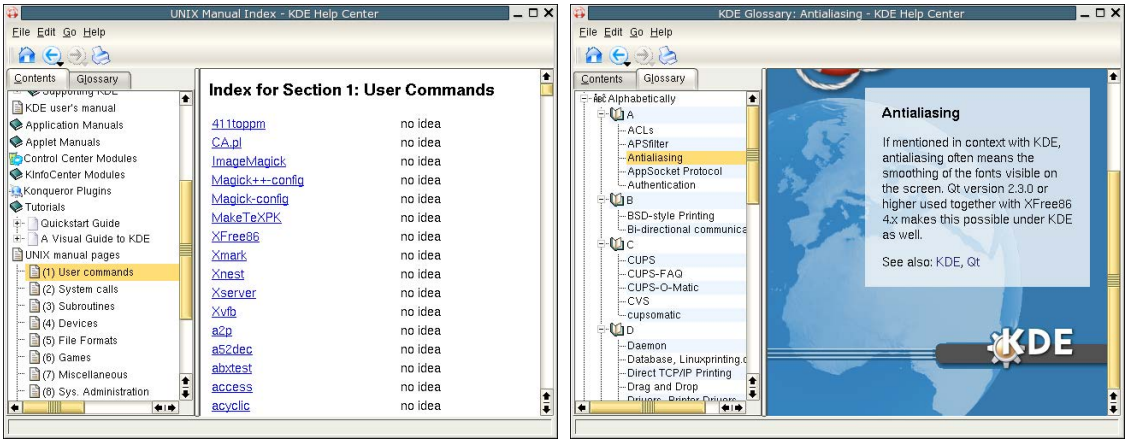
`khelpcenter`
Help center ของ KDE.

Help ของ bash เชลล์

จากช่วงที่ผ่านมาเรารู้แล้วว่าคำสั่งที่สั่งในเชลล์พรอมต์มีทั้งคำสั่งภายในซึ่งเชลล์รับรู้กระทำการเองได้และคำสั่งภายนอกซึ่งได้แก่การเรียกโปรแกรมมาใช้งาน. คู่มือการใช้คำ



รูปที่ 2.10: GNOME Help browser



รูปที่ 2.11: KDE Help center

สั่งภายในของเชลล์สามารถอ่านได้ด้วยคำสั่ง “`man bash`” หรือ “`info bash`”. คู่มือการใช้ `bash` มีเนื้อหาดี, ผู้ใช้ใหม่ควรจะอ่านคู่มือนี้หนึ่งรอบ.

help อ้างอิงหน้า 422

นอกจากการใช้ `man` หรือ `info` แล้ว, bash เชลล์มีคำสั่ง `help` แสดงวิธีใช้คำสั่งและคำอธิบายอย่างง่าย ๆเมื่อให้ชื่อคำสั่งเป็นอาร์กิวเมนต์. ถ้าสั่งคำสั่ง `help` เดี่ยว ๆจะแสดงรายชื่อคำสั่งที่สามารถอธิบายได้. ตัวอย่างเช่น

ตัวอย่างที่ 2.78: การใช้ `help` อธิบายคำสั่งภายในของเชลล์

```
$ help cd↵
cd: cd [-L|-P] [dir]
    Change the current directory to DIR.  The variable $HOME is the
    default DIR.  The variable CDPATH defines the search path for
    the directory containing DIR.  Alternative directory names in CDPATH
    are separated by a colon (:).  A null directory name is the same as
    the current directory, i.e. `.'.  If DIR begins with a slash (/),
    then CDPATH is not used.  If the directory is not found, and the
    shell option `cdable_vars' is set, then try the word as a variable
    name.  If that variable has a value, then cd to the value of that
```

```
variable.  The -P option says to use the physical directory structure
instead of following symbolic links; the -L option forces symbolic
links to be followed.
```

ถ้าคำอธิบายยาวเกินหนึ่งหน้าจอ, เราอาจจะใช้ `less` ช่วยในการแสดงผล.

ดูสรุปการใช้ `less` ที่หน้า 68.

ตัวอย่างที่ 2.79: การใช้ `help` ร่วมกับ `less`.

```
$ help set | less↵
```

2.6.2 เอกสารกำกับโปรแกรมใช้งาน

เวลาติดตั้งโปรแกรมต่าง ๆจะมีเอกสารนอกเหนือจาก man page มาด้วย. เอกสารเหล่านี้อาจจะเป็นเอกสารแนะนำโปรแกรม, หนังสืออนุญาติ, ข้อมูลเพิ่มเติม ฯลฯ อยู่ที่ `/usr/share/doc`. บางครั้งอาจจะมีเอกสารการใช้งานแบบ HTML ใต้ไดเรกทอรีนั้น ๆด้วย. ถ้าเป็นเอกสารแบบ HTML ก็ใช้*บราวเซอร์ (browser)* เช่น `mozilla`, `konqueror` เปิดอ่านใน X วินโดว์ได้. หรือถ้าต้องการอ่านในเทอร์มินอลก็ใช้บราวเซอร์แบบเท็กซ์โหมดเช่น `lynx` หรือ `w3m`.

HTML (Hyper Text Markup Language) ▶
มาตรฐานสำหรับเขียนเอกสารที่เผยแพร่ทาง World Wide Web. รูปแบบของไฟล์เป็นเนื้อหาเท็กซ์ที่มนุษย์อ่านแล้วเข้าใจได้. ใช้การเขียนกำกับส่วนที่เป็นหัวข้อ, เนื้อหา, รูปภาพ ฯลฯ. โดยปรกติจะใช้เว็บบราวเซอร์ (web browser) ดูไฟล์ HTML. เว็บบราวเซอร์จะทำหน้าที่จัดหน้าจอภาพตามที่เขียนกำกับไว้ในไฟล์.

2.6.3 ข้อมูลทางอินเทอร์เน็ต

เว็บไซต์อย่างเป็นทางการของดิสทริบิวชันต่าง ๆ

ดิสทริบิวชันชั้นนำต่าง ๆมักจะมีเว็บไซต์เผยแพร่เอกสารที่เกี่ยวกับดิสทริบิวชันเช่น คู่มือการติดตั้ง (Installation Guide), คู่มือเริ่มต้นใช้งาน (User's Guide), คู่มือสำหรับผู้ดูแลระบบ (System Administrator's Manual) ฯลฯ. ผู้ที่สนใจสามารถไปที่เว็บไซต์ของดิสทริบิวชันที่ใช้อยู่และหาเอกสารที่ต้องการ.

นอกจากเว็บไซต์ของดิสทริบิวชันแล้วยังมีเว็บไซต์ The Linux Document Project (TLDP)[27] ที่รวบรวมเอกสารต่าง ๆที่เรียกว่า HOWTO, Guide, FAQ ฯลฯ. เอกสารที่เผยแพร่และรวบรวมโดย TLDP นี้ไม่ขึ้นกับดิสทริบิวชันใด ๆและเขียนโดยอาสาสมัครทั่วโลก. เอกสารหลักเป็นภาษาอังกฤษและมีการแปลเป็นภาษาอื่น ๆด้วย. เอกสาร HOWTO เป็นเอกสารที่เจาะจงรายละเอียดตั้งแต่เรื่องทั่ว ๆไปจนถึงเรื่องเฉพาะด้านเช่น วิธีการเซ็ตเชลล์พรอมต์, การสร้าง VPN ฯลฯ. เอกสารอีกประเภทที่น่าอ่านของ TLDP อีกประเภทได้แก่เอกสารยาวที่เป็นหนังสือเรียกว่า Guide ต่าง ๆเช่น Advanced Bash-Scripting Guide, The Linux System Administrators' Guide เป็นต้น. เอกสารเหล่านี้เผยแพร่หลายฟอร์แมตเช่น HTML, PDF, PostScript ฯลฯ.

แหล่งข้อมูลที่มีประโยชน์อีกแห่งได้แก่เว็บไซต์บริการหาข้อมูลเช่น Google, Yahoo ฯลฯ. หากผู้ใช้มีปัญหาการใช้โปรแกรมเช่นเกิด error ขึ้นตอนใช้งาน, ก็สามารถนำข้อความ error นั้นไปค้นหาบนอินเทอร์เน็ตได้. นอกจากค้นหาจากเว็บไซต์แล้ว, การหาข้อมูลจากนิวส์กรุปเช่นจาก `http://groups.google.com` ในบางครั้งจะได้ข้อมูลที่ไม่มีในเว็บก็ได้.

`w3m` จะมี key binding ทั้งแบบ `vi` และ `emacs` ให้ใช้ด้วย. จะให้ความรู้สึกเหมือนกับใช้เพจเจอร์ `less` แต่เป็นบราวเซอร์. จริง ๆแล้ว `w3m` สามารถใช้เป็นเพจเจอร์ได้ด้วย.

FAQ (Frequently Asked Questions ▶
การรวมคำถามและคำตอบที่ถามกันบ่อยสรุปไว้เพื่อเป็นข้อมูลสำหรับคนอื่นที่มีคำถามเดียวกัน.

VPN (Virtual Private Network) ▶
การใช้เครือข่ายสาธารณะเช่นอินเทอร์เน็ตในการรับส่งข้อมูลโดยมีวิธีการรักษาความปลอดภัยของข้อมูลด้วย. พูดง่าย ๆคือการสร้างเครือข่ายที่มีความปลอดภัยสูง (เช่นการลงรหัส) บนเครือข่ายสาธารณะ.

แหล่งข้อมูลอื่น ๆที่ผู้ใช้อาจจะถามคำถามได้เช่นเว็บบอร์ด, ฟอรัม (forum), mailing list เป็นต้น. เว็บไซต์ภาษาไทยที่ผู้อ่านสามารถถามตอบปัญหาต่าง ๆได้แสดงไว้ในตารางที่ 2.12.

URL (Uniform Resource Locator) ► วิธีการอ้างอิงแหล่งข้อมูลเช่นเอกสารทางอินเทอร์เน็ต. ตัวอย่างเช่น http://linux.thai.net:80/plone ประกอบด้วยส่วนต่าง ๆได้แก่ โปรโตคอล (http), ชื่อโดเมน (linux.thai.net), หมายเลขพอร์ต (80) และ path (plone).

ตารางที่ 2.12: เว็บไซต์ที่บุคคลถามตอบปัญหาได้.

เว็บไซต์	URL
Thai Linux Working Group	`http://linux.thai.net`
Linux Siam	`http://www.linuxsiam.com`
Thai Linux Cafe	`http://thailinuxcafe.com`
Tux Crazy	`http://www.tuxcrazy.com`
Grandlinux Solution	`http://www.grandlinux.com`

2.7 การปรับแต่งเชลล์

โดยปรกติแล้วลินุกซ์ดิสทริบิวชันที่ผู้อ่านติดตั้งนั้นมักจะปรับแต่งเชลล์ไว้ให้เรียบร้อยแล้ว. บ้างก็ตั้งค่าตัวแปรสภาพแวดล้อมที่จำเป็นต่าง ๆไว้ให้, บ้างก็ปรับแต่งพรอมต์ให้ดูสวยงามใช้ง่าย, บ้างก็สร้าง alias ไว้ให้ ฯลฯ. ในช่วงนี้เราจะมาดูวิธีการปรับแต่งเชลล์เพื่อให้การใช้งานเชลล์สะดวกขึ้น. และเรียนรู้เกี่ยวกับระบบให้ลึกซึ้งยิ่งขึ้น.

2.7.1 ไฟล์ตั้งค่าเริ่มต้น

ในที่นี้จะครอบคลุมเนื้อหาของ bash เท่านั้น.

จากตัวอย่างการสร้าง alias (หน้า 51) ที่ผ่านมา, alias ที่สร้างนั้นจะใช้ได้ในเชลล์ที่ทำงานอยู่เท่านั้น. ถ้าเราจบการใช้งานเชลล์แล้ว alias ที่เราสร้างหรือสิ่งที่เราปรับค่าไว้ก็จะหายไป. ในระบบยูนิกซ์, มักจะมี*ไฟล์ตั้งค่าเริ่มต้น (initialization file)* ซึ่งเป็นเท็กซ์ไฟล์ที่อ่านได้, ใช้สำหรับเขียนคำสั่งเริ่มต้น, ปรับแต่งค่าตัวแปร, ปรับแต่งพฤติกรรมของโปรแกรม. เชลล์ก็เช่นกันมีไฟล์ตั้งค่าเริ่มต้นของระบบ (system wide) และส่วนบุคคล.

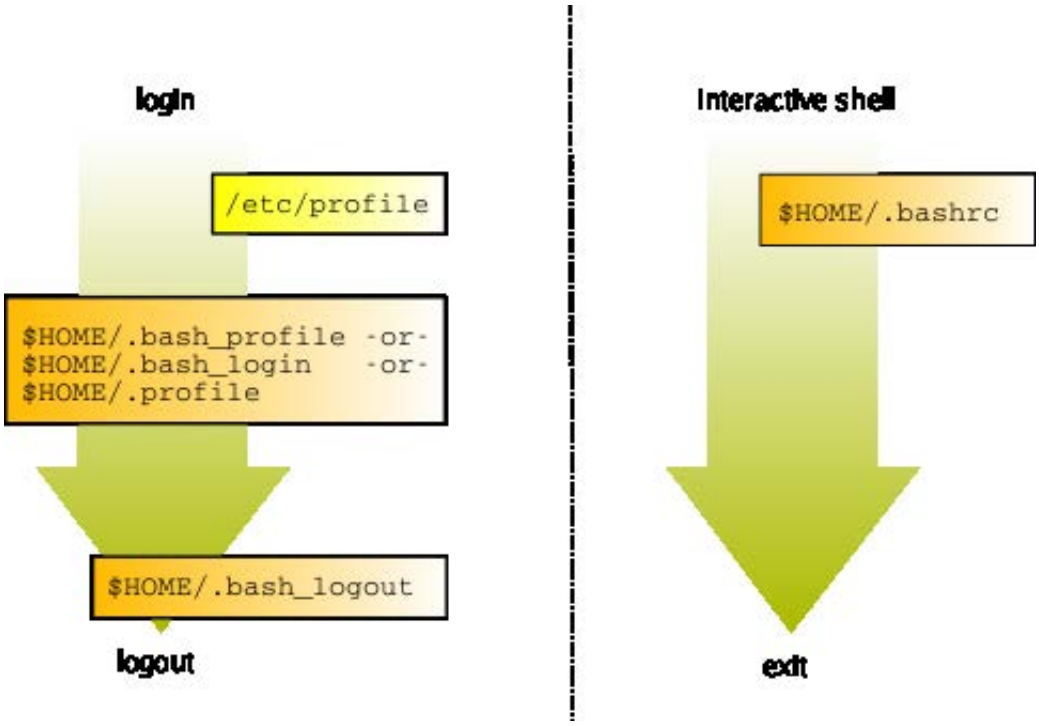
ในภาษาอังกฤษ, บ้างก็เรียกไฟล์ตั้งค่าเริ่มต้นว่า init file, rc file หรือ dot file. ไฟล์ตั้งค่าเริ่มต้นมักจะเป็นไฟล์ที่นำหน้าด้วยจุด (`.`) ทำให้เวลาสั่งคำสั่ง `ls` แล้วมองไม่เห็น, จึงเรียกว่า dot file. บางทีมีการตั้งชื่อให้ชัดเจน เช่น `.bashrc` โดยการเติม "rc". จากการบอกเล่ากันมา, "rc" ย่อมาจาก run command. กล่าวคือเป็นไฟล์ที่เขียนคำสั่งที่ต้องการทำไว้เฉพาะเวลาโปรแกรมนั้น ๆเริ่มทำงาน.

ไฟล์ที่เชลล์อ่านเมื่อเพื่อตั้งค่าเริ่มต้นของระบบนั้นจะแตกต่างกันขึ้นอยู่กับประเภทของเชลล์ที่ใช้ว่าเป็นล็อกอินเชลล์หรือเชลล์เชิงโต้ตอบ. ในกรณีที่เป็นล็อกอินเชลล์, เชลล์จะอ่านค่าเริ่มต้นจากไฟล์ `/etc/profile` ซึ่งเป็นไฟล์ตั้งค่าเริ่มต้นของระบบก่อน. หลังจากนั้นจะตรวจดูว่าในโฮมไดเรกทอรีมีไฟล์ `.bash_profile` หรือ `.bash_login` หรือ `.profile` หรือไม่ตามลำดับ. ถ้ามีเจอไฟล์ใดไฟล์หนึ่งที่กล่าวไปแล้วก็จะอ่านเนื้อหาจากไฟล์นั้นและเลิกค้นหาไฟล์อื่น ๆ. ถ้ามีไฟล์ `.bash_logout` ในโฮมไดเรกทอรี, เชลล์จะอ่านไฟล์นี้ตอนที่เลิกการทำงาน.

เชลล์ที่ใช้ในเทอร์มินอลเอมิวเลเตอร์บางชนิดเช่น `gnome-terminal` สามารถเลือกได้ว่าจะใช้ล็อกอินเชลล์หรือเชลล์เชิงโต้ตอบ.

ในกรณีที่เป็นเชลล์เชิงโต้ตอบเช่น เชลล์ที่อยู่ใน `xterm` เวลาเริ่มต้นจะอ่านค่าเริ่มต้นในไฟล์ `.bashrc` ที่อยู่ในโฮมไดเรกทอรีเท่านั้น.

เนื่องจากการใช้เชลล์ประเภทต่างกันจะอ่านไฟล์ตั้งค่าเริ่มต้นที่แตกต่างกันไปด้วย. ดังนั้นถ้าจะตั้งค่าหรือปรับแต่งเชลล์ควรจะระวังด้วยว่าเชลล์ที่กำลังใช้อยู่นั้นอ่านไฟล์ตั้งค่า



รูปที่ 2.12: ไฟล์ตั้งค่าเริ่มต้นของ bash

เริ่มต้นจากที่ไหน. ลินุกซ์ดิสทริบิวชันบางค่ายปรับแต่งไฟล์เช่น .bash_profile ให้เชลล์อ่านไฟล์ .bashrc อีกต่อด้วย. หมายความว่าไม่ว่าผู้ใช้จะใช้ล็อกอินเชลล์หรือเชลล์เชิงโต้ตอบ, การปรับแต่งเชลล์ที่เขียนไว้ในไฟล์จะถูกอ่านเสมอ.

2.7.2 ไฟล์ /etc/profile

หลังจากที่แนะนำการใช้เชลล์และคำสั่งต่าง ๆไปพอสมควรแล้ว, เราลองมาดูตัวอย่างไฟล์ตั้งค่าเริ่มต้นเชลล์ /etc/profile ที่ใช้ในลินุกซ์ดิสทริบิวชัน Gentoo กัน. เนื้อหาของไฟล์นี้จะแตกต่างกันตามดิสทริบิวชันค่ายต่าง ๆแต่ก็มีหน้าที่เหมือนกันคือปรับแต่งค่าเริ่มต้นของล็อกอินเชลล์ในระบบ. ไฟล์นี้เป็นไฟล์ที่มีอยู่แล้วไม่ต้องเขียนเองและไม่ควรแก้ไขถ้าไม่มีความรู้เกี่ยวกับเชลล์อย่างเพียงพอเพราะไฟล์นี้จะมีผลกระทบกับทุกคนในระบบ. ถ้าต้องการปรับแต่งค่าเริ่มต้นเชลล์เฉพาะบุคคลให้แก้ไฟล์ $HOME/.bash_profile.


เครื่องหมาย ⤸ ใช้แสดงว่าเนื้อหาที่เขียนยังอยู่ในบรรทัดเดียวกันแต่ขึ้นบรรทัดใหม่ให้อ่านง่ายขึ้นเท่านั้น.

ตัวอย่างที่ 2.80: ไฟล์ตั้งค่าเริ่มต้นของเชลล์

```
$ cat -n /etc/profile↵
     1  # /etc/profile:
     2  # $Header: /home/cvsroot/gentoo-src/rc-scripts/etc/profile,v ⤸
1.23 2003/04/29 21:23:18 azarah Exp $
     3
     4  if [ -e "/etc/profile.env" ]
     5  then
     6          . /etc/profile.env
```

```
 7  fi
 8
 9  # 077 would be more secure, but 022 is generally quite realistic
10  umask 022
11
12  if [ ‘/usr/bin/whoami‘ = ’root’ ]
13  then
14          # Do not set PS1 for dumb terminals
15          if [ "$TERM" != ’dumb’  ] && [ -n "$BASH" ]
16          then
17                  export PS1=’\[\033[01;31m\]\h \[\033[01;34m\]\W
\$ \[\033[00m\]’
18          fi
19          export PATH="/bin:/sbin:/usr/bin:/usr/sbin:$ROOTPATH"
20  else
21          # Do not set PS1 for dumb terminals
22          if [ "$TERM" != ’dumb’  ] && [ -n "$BASH" ]
23          then
24                  export PS1=’\[\033[01;32m\]\u@\h \[\033[01;34m\]
\W \$ \[\033[00m\]’
25          fi
26          export PATH="/bin:/usr/bin:$PATH"
27  fi
28  unset ROOTPATH
29
30  if [ -z "$INPUTRC" -a ! -f "$HOME/.inputrc" ]
31  then
32          export INPUTRC="/etc/inputrc"
33  fi
34
35  # Extract the value of EDITOR
36  [ -z "$EDITOR" ] && EDITOR="‘. /etc/rc.conf 2>/dev/null;
echo $EDITOR‘"
37  [ -z "$EDITOR" ] && EDITOR="‘. /etc/conf.d/basic 2>/dev/null;
echo $EDITOR‘"
38  [ -z "$EDITOR" ] && EDITOR="/bin/nano"
39  export EDITOR
```

คอมเมนต์

บรรทัดที่ 1 และ 2 ขึ้นต้นบรรทัดด้วยเครื่องหมาย number (#) ซึ่งเชลล์จะไม่แปลความสิ่งที่เขียนหลังเครื่องหมาย number จนสุดบรรทัด. สองบรรทัดแรกนี้เรียกว่า*คอมเมนต์ (comment)* ใช้เขียนอธิบายเนื้อหาที่อยู่ในไฟล์นี้.

คำว่า comment แปลเป็นภาษาไทยตรงๆได้ว่า “หมายเหตุ”. แต่ในที่นี้จะใช้ทับศัพท์ว่าคอมเมนต์.

ตรวจสอบไฟล์ด้วย `if`

บรรทัดที่ 4 ถึง 7 เป็นการตรวจสอบว่ามีไฟล์ `/etc/profile.env` หรือไม่ด้วยคำสั่ง `if`.

`if` เป็นคำสั่งผสมที่ใช้ในเชลล์เพื่อตรวจสอบกรณีต่าง ๆ.

```
if COMMAND                          ← ถ้าสถานะการจบคำสั่งเป็น 0 จะถือว่าเป็นจริง
then
    COMMAND                         ← คำสั่งที่ใช้ในกรณีที่เป็นจริง
    ...
```

```
elif COMMAND                                    ← ตรวจสอบด้วย if อีกครั้ง
then
    COMMAND
    ...
else
    COMMAND                                     ← คำสั่งที่ใช้ในกรณีอื่นๆ (เท็จ)
    ...
fi                                              ← จบการตรวจสอบด้วย if
```

ให้สังเกตว่าสิ่งที่พิมพ์หลังคำสั่ง if เป็นคำสั่งและคำสั่งนี้จะถูกกระทำการจริง. ถ้าคำสั่งจบบริบูรณ์โดยไม่มีข้อผิดพลาด (สถานะการจบเป็น 0) ก็จะถือว่าการตรวจสอบเป็นจริง. จะกระทำการคำสั่งที่อยู่ในช่วง then ต่อไป. บรรทัดที่ 4 คำสั่งที่ใช้ในการตรวจสอบคือคำสั่ง [ซึ่งเป็นคำสั่งภายในและเหมือนกับคำสั่ง test (หน้า 416). คำสั่ง [ต้องการอาร์กิวเมนต์ตัวสุดท้ายเป็น] เสมอเพื่อให้ดูเรียบร้อยเหมือนกับไวยกรณ์ของโปรแกรม. กล่าวคือ “[-e "/etc/profile.env"]” เขียนได้อีกแบบคือ “test -e "/etc/profile.env"”.

ให้สังเกตว่าหลัง [และก่อนหน้า] ต้องมีช่องไฟเพราะ [เป็นคำสั่งและ] เป็นอาร์กิวเมนต์ของคำสั่ง.

อ่านเนื้อหาไฟล์อื่นเข้ามาในเชลล์

ถ้ามีไฟล์ /etc/profile.env อยู่ให้เชลล์อ่านเนื้อหาคำสั่งที่อยู่ในไฟล์นั้น. เครื่องหมายจุด (.) ในบรรทัดที่ 6 ก็เป็นคำสั่งภายเชลล์อย่างหนึ่งทำหน้าที่อ่านคำสั่งที่อยู่ในไฟล์ที่เป็นอาร์กิวเมนต์. เมื่อใช้คำสั่ง . กับไฟล์ที่เป็นอาร์กิวเมนต์ก็เหมือนกับการนำคำสั่งที่เขียนอยู่ในไฟล์นั้นมากระทำการในเชลล์ที่ใช้อยู่. คำสั่ง . ดูผิวเผินดูไม่เหมือนกับเป็นคำสั่ง, จึงมีคำสั่งที่อ่านแล้วเข้าใจคือ source ซึ่งเป็นคำสั่งที่ทำหน้าที่เหมือนกับ . ทุกประการ.

ในไฟล์ /etc/profile.env มีการประกาศค่าตัวแปรสภาพแวดล้อมต่าง ๆที่ Gentoo ตั้งค่าไว้.

source อ้างอิงหน้า 424

บรรทัดที่ 10 เป็นการตั้งค่าสิทธิ์การใช้ไฟล์โดยใช้คำสั่งภายในเชลล์ umask. จะอธิบายคำสั่งนี้ในช่วงที่แนะนำเรื่องสิทธิ์การใช้ไฟล์.

ตั้งค่าพรอมต์

บรรทัดที่ 12 ถึง 27 เป็นการตั้งค่าพรอมต์และ PATH. จะมีการตรวจสอบว่าถ้าผู้ใช้คือ root ก็จะตั้งค่าพรอมต์และ PATH ให้ต่างจากผู้ใช้ทั่วไป. ตรงนี้มีการใช้ test ตรวจสอบผลของคำสั่ง whoami ว่าได้ผลเป็น root หรือไม่. ก่อนที่จะตั้งค่าพรอมต์มีการตรวจสอบชนิดของเทอร์มินอลว่าไม่ใช่*เทอร์มินอลแบบ dumb (dumb terminal)*.

test อ้างอิงหน้า 416

whoami อ้างอิงหน้า 419

dump terminal ▶
เป็นเทอร์มินอลโบราณรุ่นแรก ๆ. ไม่มีความสามารถในการจัดแต่งหน้าจอ, ไม่มีสี (ขาวดำ) หรือมีขีดจำกัดอื่น ๆ. กล่าวคือเป็นเทอร์มินอลแบบง่าย ๆ, มีเพียงคุณสมบัติเบื้องต้นที่พอจะใช้งานได้เท่านั้น.

ค่าของพรอมต์เก็บอยู่ในตัวแปรสภาพแวดล้อม PS1 ดูซับซ้อนแต่ความจริงไม่ยากที่จะทำความเข้าใจ. พรอมต์ที่ตั้งนี้เป็นพรอมต์ที่มีสีซึ่งใช้คุณสมบัติของ *เทอร์มินอลมาตรฐาน ANSI (ANSI terminal)* ในการตกแต่งสีต่าง ๆ. ค่าของพรอมต์ในบรรทัดที่ 17 และ 24 แบ่งออกได้เป็น 3 ส่วนได้แก่.

1. escape sequence ของเชลล์
   escape ของเชลล์จะนำหน้าด้วยเครื่องหมาย backslash (\). จากตัวอย่างไฟล์ /etc/profile บรรทัดที่ 17 ได้แก่ \[, \], \h, \W และ \@. ตารางที่ 2.13 แสดง escape sequence ที่ใช้ได้และคำอธิบาย.

ตารางที่ 2.13: escape sequence ที่ใช้กับเชลล์พรอมต์

escape sequence	ความหมาย
\a	อักขระ ASCII กระดิ่ง. จะไม่แสดงอะไรบนหน้าจอแต่จะมีเสียงกระดิ่งออกทางลำโพง.
\d	แสดงวันที่ในพรอมต์ในรูปของ “วัน เดือน วันที่”.
\D{*FORMAT*}	แสดงวันโดยใช้รูปแบบที่ระบุด้วย *FORMAT*. strftime(3) เพิ่มเติมเกี่ยวกับ *FORMAT*.
\e	อักขระ ESC.
\h	ชื่อโฮสซึ่งจะแสดงชื่อโฮสถึงเครื่องหมายจุด (.) ที่เจอตัวแรกเท่านั้น. เช่นถ้าชื่อโฮสเป็น localhost.localdomain, \h ก็จะเป็น localhost.
\H	ชื่อโฮส.
\j	จำนวนจ็อบที่ควบคุมโดยเชลล์นั้น.
\l	basename ของดีไวส์เทอร์มินอลที่ใช้. เช่นเทอร์มินอลที่ใช้คือดีไวส์คือ /dev/pst/6, \l จะเป็น 6.
\n	อักขระ newline ขึ้นบรรทัดใหม่.
\r	อักขระ cariiage return ขึ้นบรรทัดใหม่.
\s	ชื่อเชลล์ที่ใช้.
\t	เวลาปัจจุบันแบบ 24 ชั่วโมงเช่น 23:59:59.
\T	เวลาปัจจุบันแบบ 12 ชั่วโมงเช่น 11:59:59.
\@	เวลาปัจจุบันแบบ 12 ชั่วโมงเช่น 11:59 PM.
\A	เวลาปัจจุบันแบบ 24 ชั่วโมงเช่น 23:59.
\u	ชื่อผู้ใช้.
\v	เวอร์ชันของ bash เชลล์ที่ใช้เช่น 2.00.
\V	รีลีสของ bash เชลล์ที่ใช้เช่น 2.00.0 (เวอร์ชัน + patch level).
\w	ไดเรกทอรีที่ทำงานอยู่เช่น ~ (โฮมไดเรกทอรี).
\W	ไดเรกทอรีที่ทำงานอยู่เช่น somchai (โฮมไดเรกทอรีของ somchai).
\!	หมายเลขประวัติคำสั่งของคำสั่งนี้.
\#	หมายเลข (จำนวน) คำสั่ง.
\$	แสดงพรอมต์ # ถ้าเป็น root หรือ $ อื่นๆ.
nnn	เลขฐานแปด *nnn* ใช้เขียนแทนอักขระที่ไม่สามารพิมพ์ได้.
\\	backslash (\)
\[	เริ่มอักขระที่แสดงทางหน้าจอไม่ได้ใช้เพื่อพิมพ์อักขระควบคุมเทอร์มินอล.
\]	จบอักขระที่แสดงทางหน้าจอไม่ได้.

2. ANSI escape sequence
   ANSI escape sequence เป็น escape sequence ที่ใช้ควบคุมเทอร์มินอลที่มีคุณสมบัติตามที่ ANSI กำหนด. เทอร์มินอลเอมิวเลเตอร์ที่ใช้ใน X วินโดว์โดยทั่วไปก็จัดอยู่ในพวกนี้. จากบรรทัดที่ 17, ช่วงที่เป็น ANSI escape sequence คือ "`\033[01;31m`".

3. อักขระทั่วไป
   อักขระทั่วไปคือเครื่องหมายหรืออักษรที่มองเห็นได้ทางเทอร์มินอลไม่มีความหมายพิเศษสำหรับเชลล์. จากบรรทัดที่ 17 ได้แก่ช่องไฟ. จากบรรทัดที่ 24 ได้แก่เครื่องหมาย `@`.

ตั้งค่าตัวแปรสภาพแวดล้อม

ตั้งแต่บรรทัดที่ 28 จนจบไฟล์เป็นการตั้งค่าตัวแปรสภาพแวดล้อมต่าง ๆ. บรรทัดที่ 28 จะลบตัวแปรสภาพแวดล้อม `ROOTPATH` ซึ่งถูกตั้งค่าจากการอ่านไฟล์ `/etc/profile.env` บรรทัดที่ 6 ด้วยคำสั่ง `unset`.

`unset` อ้างอิงหน้า 425

บรรทัดที่ 30 ถึง 33 เป็นตั้งค่าตัวแปรสภาพแวดล้อม `INPUTRC` ซึ่งมีการตรวจก่อนว่าถ้าตัวแปร `INPUTRC` มีมีความยาวเป็น 0 (`-z $INPUTRC`) และ (`-a`) ไม่มีไฟล์ `$HOME/.inputrc` (`! -f $HOME/.inputrc`) จึงจะตั้งค่าตัวแปรสภาพแวดล้อม `INPUTRC` เป็น `/etc/inputrc`. ตัวแปรสภาพแวดล้อม `INPUTRC` เป็นตัวแปรที่เก็บชื่อไฟล์ที่เป็นไฟล์ตั้งค่าเริ่มต้นของ *ไลบรารี readline (readline library)*. ไลบรารีนี้ใช้ใน bash เชลล์สำหรับช่วยในการแก้ไขบรรทัดคำสั่ง, การเติมเต็มคำสั่ง. ไฟล์ตั้งค่าเริ่มต้นโดยปริยายจะเป็นไฟล์ `.inputrc` ที่อยู่ในโฮมไดเรกทอรีของผู้ใช้ดังนั้นจึงมีการตรวจสอบไฟล์นี้ว่ามีอยู่หรือไม่. ถ้าไม่มีตัวระบบก็จะตั้งค่าให้เป็นไฟล์ที่ระบบเตรียมไว้.

ตัวแปรมีความยาวเป็น 0 หมายถึงไม่มีค่าหรือตัวแปรไม่มีตัวตน.

`test` อ้างอิงหน้า 416

บรรทัดที่ 36 จนจบไฟล์เป็นการตั้งค่าตัวแปรสภาพแวดล้อม `EDITOR`. ตัวแปร `EDITOR` จะใช้เมื่อโปรแกรมที่ใช้งานอยู่ต้องการให้ผู้ใช้แก้ไขไฟล์แต่โปรแกรมนั้นไม่ใช่บรรณาธิกรณ์จึงไม่สามารถแก้ไขไฟล์ได้. กรณีนี้โปรแกรมนั้นจะเรียกบรรณาธิกรณ์ที่ระบุอยู่ในตัวแปรสภาพแวดล้อม `EDITOR` ขึ้นมาให้ผู้ใช้แก้ไขไฟล์ที่ต้องการ. โปรแกรมที่ทำอย่างนี้เช่น `less` (เมื่อต้องแก้ไขไฟล์ที่ดูอยู่), `cvs` (เมื่อต้องการแก้ไขไฟล์หรือเขียนหมายเหตุ) ฯลฯ.

2.7.3 สีที่ใช้ในเทอร์มินอลแบบ ANSI

ANSI escape sequence เป็น escape sequence ที่ใช้ควบคุมหรือปรับแต่พฤติกรรมของเทอร์มินอลที่มีมาตรฐานแบบ ANSI. escape sequence ที่เกี่ยวกับการปรับแต่งสีมีรูปแบบเป็น

```
<ESC>[attr1;...;attrnm
```

- <ESC> คืออักขระ ESC. เนื่องจากอักขระ ESC ไม่สามารถแสดงได้ในเทอร์มินอลได้เลยใช้ `\033` ซึ่งเป็นค่าของอักขระ (เลขฐานแปด) แทนในบรรทัดคำสั่ง.

- [เป็นการเริ่มต้นคุณลักษณะของอักษรที่ต้องการแสดง.
- ตัวเลขที่ถัดจาก [คือค่าที่ระบุเช่น 01 หมายถึงใช้ตัวหนา. 33 หมายถึงใช้ตัวอักษรสีเหลือง. ในการระบุค่ามากกว่าสองอย่างให้ใช้เครื่องหมาย semi-colon (;) ขั้น.
- m เป็นการสั่งตั้งค่าที่ระบุ.

สี Magenta คือสีที่เกิดจากการผสมแม่สี, สีแดง (Red) และสีฟ้า (Blue). สี Cyan คือสีที่เกิดจากการผสมแม่สี, สีเขียว (Green) และสีฟ้า (Blue).

ตารางที่ 2.14: ANSI escape sequence ที่เกี่ยวกับสี. [1]

ค่า	คำอธิบาย
0	ยกเลิกค่าที่ตั้งไว้ทั้งหมด
1	ทำอักษรที่แสดงให้เข้มขึ้น
2	ทำอักษรที่แสดงให้จางลง
4	ขีดเส้นใต้
5	ทำตัวอักษรให้กระพิบ
7	สลับเปลี่ยนสีฉากหลังกับสีตัวอักษร
8	ซ่อนอักษรที่แสดง
30	ใช้สีตัวอักษรสีดำ (Black)
31	ใช้สีตัวอักษรสีแดง (Red)
32	ใช้สีตัวอักษรสีเขียว (Green)
33	ใช้สีตัวอักษรสีเหลือง (Yellow)
34	ใช้สีตัวอักษรสีฟ้า (Blue)
35	ใช้สีตัวอักษรสีแดงม่วง (Magenta)
36	ใช้สี Cyan
37	ใช้สีตัวอักษรสีขาว (White)
40	ใช้ฉากหลังสีตัวอักษรสีดำ (Black)
41	ใช้ฉากหลังสีตัวอักษรสีแดง (Red)
42	ใช้ฉากหลังสีตัวอักษรสีเขียว (Green)
43	ใช้ฉากหลังสีตัวอักษรสีเหลือง (Yellow)
44	ใช้ฉากหลังสีตัวอักษรสีฟ้า (Blue)
45	ใช้ฉากหลังสีตัวอักษรสีแดงม่วง (Magenta)
46	ใช้ฉากหลังสี Cyan
47	ใช้ฉากหลังสีตัวอักษรสีขาว (White)

การตั้งคุณลักษณะพิเศษของสิ่งที่แสดงบนเทอร์มินอลขึ้นอยู่กับความสามารถของเทอร์มินอลจึงมีการตรวจสอบประเภทของเทอร์มินอลก่อนตั้งค่านี้. เราอาจจะทดลองใช้ echo พิมพ์ ANSI escape sequence ดูทางเทอร์มินอลด้วยตัวเลือก -e.

echo อ้างอิงหน้า 420

ตัวอย่างที่ 2.81: การใช้ echo พิมพ์ ANSI escape sequence.

```
echo -e "\033[1;4mBold and Underline\033[0m Normal \033[1mBold again"↵
```

Bold and Underline Normal **Bold again**

2.7.4 เชลล์พรอมต์

เชลล์พรอมต์ที่ใช้กันทั่วไปมีตั้งแต่แบบง่ายสุดตั้งแต่การใช้เครื่องหมายดอลลาร์ $ จนถึงการใช้สคริปต์และฟอนต์พิเศษช่วย [28]. พรอมต์ที่ใช้ใน bash มีอยู่ 4 ชนิดเก็บค่าไว้ในตัวแปรสภาพแวดล้อมต่าง ๆกันได้แก่

1. PS1
   เก็บค่าพรอมต์หลักที่ใช้ในล็อกอินเชลล์หรือเชลล์เชิงโต้ตอบ. เราจะคุ้นเคยกับพรอมต์นี้เพราะเป็นพรอมต์ที่ใกล้ตัวที่สุด. และการปรับแต่งพรอมต์ก็มักจะปรับแต่งค่า PS1. ค่าโดยปริยายของ PS1 คือ "\s-\v\$ ".

   หลังเครื่องหมาย $ มีช่องไฟหนึ่งตัว.

   ตัวอย่างที่ 2.82: ค่าปริยายของพรอมต์หลัก PS1.

   ```
   -bash-2.05b$ echo $PS1↵
   \s-\v$
   ```

   โดยปรกติแล้วดิสทริบิวชันจะปรับแต่งค่า PS1 ไว้ให้ในไฟล์ /etc/profile.

2. PS2
   เก็บค่าพรอมต์รองใช้แสดงเป็นพรอมต์เมื่อสั่งคำสั่งไม่จบภายในหนึ่งบรรทัดแล้วขึ้นบรรทัดใหม่. ค่าโดยปริยายของพรอมต์รองนี้คือ "> ".

   ตัวอย่างที่ 2.83: ค่าปริยายของพรอมต์รอง PS2.

   ```
   -bash-2.05b$ "Primary prompt $PS1↵
   > Secondary prompt $PS2"↵
   Primary prompt \s-\v$
   Secondary prompt >
   ```

3. PS3
   เป็นพรอมต์ที่ใช้กับคำสั่ง select. คำสั่ง select เป็นคำสั่งแบบไวยกรณ์ของเชลล์ที่ใช้แสดงตัวเลือกเป็นข้อ ๆให้เลือกตอบคำถาม.

   select อ้างอิงหน้า 423

   ตัวอย่างที่ 2.84: ใช้ select ถามคำถาม.

   ```
   $ echo What is your favorite character in One Piece?;\↵
   > select name in Luffy Nami Zoro Sanji Usopp Choper Robin;↵
   > do echo I like $name also!↵
   > break↵
   > done↵
   What is your favorite character in One Piece?
   1) Luffy
   2) Nami
   3) Zoro
   4) Sanji
   5) Usopp
   6) Choper
   7) Robin
   #? 6↵
   I like Choper also!
   ```

```
$ echo $name↵
Chopper
```

ถ้าตั้งค่า PS3, คำสั่ง `select` ก็จะใช้ค่าที่ตั้งไว้เป็นพรอมต์แทน “#?”.

4. PS4
   เป็นพรอมต์ที่แสดงเมื่อมีการติดตาม (trace) คำสั่ง. การติดตามคำสั่งนี้ทำได้โดยตั้งค่า `xtrace` หรือรันเชลล์ด้วยตัวเลือก `-x`. ค่าปริยายของ PS4 คือ `+`.

ตัวอย่างที่ 2.85: การติดตามคำสั่งในเชลล์

```
$ set -o xtrace↵                                          ← หรือ set -x
++ echo -ne '\033]0;somchai@toybox:~\007'
$ echo Hello↵
+ echo Hello
Hello
++ echo -ne '\033]0;somchai@toybox:~\007'
$
```

การติดตามคำสั่งในเชลล์มีประโยชน์อย่างยิ่งเมื่อใช้ debug เชลล์สคริปต์ดูการสั่งคำสั่งต่าง ๆในสคริปต์.

หลังจากที่ได้รู้จักกับพรอมต์ประเภทต่าง ๆแล้วเราลองมาดูตัวอย่างการตั้งค่าพรอมต์จากสิ่งที่เรียนรู้ไปแล้ว.

ตัวอย่างที่ 2.86: การปรับค่าพรอมต์จากสิ่งที่เรียนมา.

```
$ export PS1="-\[\033[32m\d, \t\033[0m\]- [\w]\n\u@\h \$  "↵
-Tue May 18, 01:38:54- [~]
somchai@toybox $ cd /usr/src/linux/Documentation/filesystems↵
-Tue May 18, 01:39:04- [/usr/src/linux/Documentation/filesystems]
somchai@toybox $
```

จากตัวอย่างเป็นการสร้างพรอมต์ให้มีสองบรรทัด. บรรทัดแรกบอกเวลาและไดเรกทอรีที่ทำงานอยู่ (full path). ส่วนบรรทัดที่สองเป็นพรอมต์ธรรมดาบอกชื่อผู้ใช้และชื่อโฮส. สำหรับผู้ที่สนใจเรื่องการตั้งค่าพรอมต์โดยละเอียด, สามารถอ่านได้จากเอกสาร Bash Prompt HOWTO [28].

2.7.5 ไลบรารี readline

ตามที่ได้แนะนำไปข้างต้นแล้วว่าเชลล์ใช้ไลบรารี readline ช่วยในการปรับแต่งบรรทัดคำสั่งและจัดการประวัติคำสั่ง. เนื่องจาก readline เป็นไลบรารี, โปรแกรมอื่น ๆที่มีการโต้ตอบกับผู้ใช้แบบ CLI ก็สามารถใช้ไลบรารี readline ได้ด้วย. ตัวอย่างเช่นโปรแกรม `gnuplot` ซึ่งมีอินเทอร์เฟสแบบ CLI ก็สามารถใช้ key binding เหมือนกับที่ใช้กับ bash เชลล์ เช่น `C-p` สั่งคำสั่งที่สั่งไปแล้ว เป็นต้น.

`gnuplot` เป็นโปรแกรมเขียนกราฟ.

ไลบรารี readline จะอ่านไฟล์ที่ระบุโดยตัวแปรสภาพแวดล้อม INPUTRC หรือถ้าไม่มีการตั้งค่าตัวแปรนี้ก็จะอ่านค่าเริ่มต้นจากไฟล์ $HOME/.inputrc. เราลองมาดูเนื้อหาของไฟล์ตั้งค่าเริ่มต้นของไลบรารี readline กัน.

ตัวอย่างที่ 2.87: ไฟล์ตั้งค่าเริ่มต้นของไลบรารี readline.

```
$ cat -n ~/.inputrc↵
     1  $include /etc/inputrc
     2
     3  # be quiet
     4  set bell-style none
     5  set meta-flag on
     6  set input-meta on
     7  set convert-meta off
     8  set output-meta on
     9
    10  "\C-x\C-r": re-read-init-file
    11  "\M-m": "LANG=ja_JP XMODIFIERS='@xim=kinput2' mozilla &"
    12
    13 $if gnuplot
    14  "\M-m": "plot sin(x)\r"
    15  $endif
    16  "\M-xf": dump-functions
    17  "\M-xv": dump-variables
    18  "\M-xm": dump-macros
```

ก่อนที่จะอธิบายเนื้อหาในไฟล์นี้, เราพอจะสังเกตุได้ว่าคอมเมนต์ของไฟล์นี้คือเครื่องหมาย number (#) และไวยกรณ์ที่ใช้ในไฟล์นี้แบ่งออกเป็น 3 ชนิดใหญ่ ๆคือ

1. ตั้งค่าตัวแปร
   การตั้งค่าตัวแปรในไฟล์ ~/.inputrc มีไวยกรณ์เป็น

   ```
   set variable-name value
   ```

   "set *variable-name value*" ในที่นี้ไม่ใช่คำสั่งเชลล์แต่เป็นไวยกรณ์ของไฟล์ตั้งค่าเริ่มต้น readline, จะนำไปใช้ในเชลล์ไม่ได้. ชื่อตัวแปรที่สำคัญและค่าโดยปริยายที่ใช้ในไฟล์ ~/.inputrc ได้แก่.

   - bell-style (audible)
     รูปแบบของกระดิ่งที่ใช้ในเทอร์มินอล. audible คือใช้เสียงกระดิ่งออกทางลำโพง. ถ้าใช้ค่า visible จะไม่มีเสียงแต่หน้าจอจะกระพิบแทน. ส่วน none คือไม่มีอะไรเกิดขึ้น.
   - completion-ignore-case (off)
     ถ้าตั้งค่าเป็น on จะเติมเต็มคำสั่งหรือชื่อไฟล์โดยที่ไม่แยกแยะอักษรตัวใหญ่และอักษรตัวเล็ก.
   - completion-query-items (100)

ค่าที่อยู่ในวงเล็บคือค่าโดยปริยาย.

จำนวนที่เป็นขีดแบ่งว่าให้ถามว่าจะแสดงส่วนเติมเต็มหรือไม่ในกรณีที่ส่วนเติมเต็มเกินจำนวนที่ตั้งไว้.

- `convert-meta` (on)
  ถ้ามีค่าเป็น on, readline จะแปลงอักขระที่เป็น 8 บิตเป็น 7 บิตโดยบิตที่แปดจะใช้อักขระ ESC เป็นตัวแทน. ขึ้นอยู่กับความสามารถของเทอร์มินอลด้วย.
- `disable-completion` (off)
  ถ้าตั้งค่าเป็น off จะไม่มีการเติมเต็มให้.
- `editing-mode` (emacs)
  ตั้งค่าวิธีการปรับแต่งบรรทัดคำสั่งแบบ emacs หรือ vi.
- `expand-tilde` (off)
  เครื่องหมาย tilde (~) ใช้แทนโฮมไดเรกทอรี. ถ้าตั้งค่าตัวแปรนี้เป็น on, เวลาเติมเต็มจะกระจายเครื่องหมาย tilde เป็น full path.
- `input-meta` (off)
  ถ้าตั้งค่าเป็น on, readline จะอนุญาตให้ข้อมูลเข้าเป็นอักขระ 8 บิต. `meta-flag` เป็นตัวแปรที่มีหน้าที่เหมือนตัวแปรตัวนี้.
- `mark-directories` (on)
  ถ้าค่าเป็น on, เวลาเติมเต็มชื่อไฟล์จะใส่เครื่องหมาย slash (/) หลังชื่อไดเรกทอรีเพื่อแยกแยะระหว่างไฟล์ให้ชัดเจน.
- `match-hidden-files` (on)
  readline จะแสดงไฟล์ที่เป็นส่วนเติมเต็มทั้งหมดรวมทั้งไฟล์ที่ขึ้นต้นด้วยจุด (.) ด้วย. ถ้าตั้งค่าเป็น off และต้องการเติมเต็มไฟล์ที่ขึ้นต้นด้วยจุด, ต้องพิมพ์จุดก่อนแล้ว readline จะเติมเต็มไฟล์ที่ขึ้นต้นด้วยจุดให้.
- `output-meta` (off)
  ถ้าตั้งค่าเป็น on, readline จะแสดงอักขระแบบ 8 บิตให้โดยที่ไม่พยายามเปลี่ยนเป็น 7 บิต. ขึ้นอยู่กับเทอร์มินอลที่ใช้ด้วย.
- `page-completions` (on)
  readline จะใช้เพจเจอร์แบบ more ในการแสดงส่วนเติมเต็มที่มีมากกว่าที่จะแสดงบนหน้าจอเดียวได้. ถ้าตั้งค่าเป็น off และมีส่วนเติมเต็มมากจะทำให้ดูยาก, แต่จะไม่ถูกรบกวนด้วยเพจเจอร์. อาจจะใช้คีย์ Shift+PgUp ย้อมกลับไปดูส่วนเติมเต็มก็ได้.

2. Key bindings

   key binding ที่ใช้อยู่ในเชลล์เช่น C-p, C-k เหล่านี้กำหนดโดยไลบรารี readline ทั้งนั้น. ไลบรารี readline จะเชื่อมคีย์ที่กำหนดไว้หรือผู้ใช้กำหนดไว้เข้ากับคำสั่งของไลบรารี readline. การกำหนดว่าคีย์ไหนจะให้ทำอะไรมีไวยกรณ์ดังนี้.

```
keys: action
```

keys อาจจะเป็นชื่อคีย์ที่ readline ได้กำหนดไว้แล้วได้แก่ DEL, ESC, ESCAPE ฯลฯ. ชื่อคีย์เหล่านี้สามารถอ่านได้จาก on-line manual หรือ info ของ readline. *keys* อีกแบบคือชุดคีย์ที่ต้องกดคีย์ Ctrl ค้างแล้วตามด้วยคีย์อื่น (\C-), กดคีย Esc ก่อนแล้วตามด้วยคีย์อื่น (M-), หรือ กดคีย์ Esc ก่อนแล้วตามด้วยการกดคีย์ Ctrl ค้างไว้แล้วตามด้วยคีย์อื่น (M-C-). การกดคีย์เหล่านี้มีอิทธิพลมาจากบรรณาธิกรณ์ Emacs.

action เป็นการกระทำที่เกิดจากการกดคีย์ที่กำหนดไว้. การกระทำนี้อาจจะเป็นฟังชันของ readline ตามตัวอย่างบรรทัดที่ 10 เช่นถ้ากดคีย์ C-x C-r แล้ว, การกระทำคือฟังชัน re-read-init-file. readline จะอ่านไฟล์ตั้งค่าเริ่มต้นที่กำหนดโดยตัวแปรสภาพแวดล้อม INPUTRC อีกครั้ง. โดยปรกติฟังชันเหล่านี้จะมี key binding โดยปริยายอยู่แล้ว. ถ้าต้องการใช้คีย์อื่นหรือต้องการระบุให้ชัดเจนก็เขียนลงในไฟล์ตั้งค่าเริ่มต้นได้. ในตารางที่ 2.15 แสดง key binding โดยปริยายและชื่อฟังชันที่สัมพันธ์กัน.

key binding บางตัวได้อธิบายไปแล้วในตารางที่ 2.1 และ 2.8

บรรทัดที่ 16-18 เป็นการกำหนด key binding สำหรับฟังชันพิเศษเช่น แสดงรายการของ key binding (M-x f) และฟังชันทั้งหมดที่ใช้ (dump-functions) เป็นต้น. ฟังชัน dump เหล่านี้โดยปรกติไม่มีการตั้งค่า key binding ไว้โดยปริยายนอกจากจะตั้งค่าเองตามตัวอย่าง. ผู้ใช้สามารถดูผลของ key binding นี้ได้จากตัวอย่างที่ 2.88.

ตัวอย่างที่ 2.88: การใช้ฟังชัน dump-functions ตามที่กำหนดใน .inputrc.

```
$ Esc x f

abort can be found on "\C-g", "\C-x\C-g", "\M-\C-g".
accept-line can be found on "\C-j", "\C-m".
alias-expand-line is not bound to any keys
arrow-key-prefix is not bound to any keys
backward-byte is not bound to any keys
backward-char can be found on "\C-b", "\M-[D".
...
```

key binding ยังสามารถใช้เรียก*แมโคร (macro)* แทนคำหรือประโยคที่เรากำหนดไว้ได้ตามตัวอย่างที่ 2.87 บรรทัดที่ 11. คำหรือประโยคที่เรากำหนดต้องเขียนอยู่ในเครื่องหมาย double quote ("). ตามตัวอย่างถ้ากดคีย์ M-m, คำสั่งที่เขียนเตรียมไว้ซึ่งได้แก่ "LANG=ja_JP XMODIFIERS='@xim=kinput2' mozilla &" จะแทรกอยู่ในบรรทัดคำสั่งให้เหมือนพิมพ์จากแป้นพิมพ์.

macro ▶
แมโคร. ชื่อที่กำหนดไว้และกระจายต่อไปคำอื่น ๆต่อไป. แมโครใช้กันอย่างกว้างขวางเช่นในภาษาซี, LaTeX ฯลฯ.

3. เงื่อนไข
   บรรทัดที่ขึ้นต้นด้วยเครื่องหมายดอลลาร์ ($) เป็นไวยกรณ์ที่ระบุเงื่อนไขหรือคำสั่ง

ตารางที่ 2.15: key binding โดยปริยายและฟังชันไลบรารี readline ที่สัมพันธ์กัน.

key binding แบบ C-	
C-@	set-mark
C-a	beginning-of-line
C-b	backward-char
C-d	delete-char
C-e	end-of-line
C-f	forward-char
C-g	abort
C-h	backward-delete-char
C-i	complete
C-j	accept-line
C-k	kill-line
C-l	clear-screen
C-m	accept-line
C-n	next-history
C-p	previous-history
C-q	quoted-insert
C-r	reverse-search-history
C-s	forward-search-history
C-t	transpose-chars
C-u	unix-line-discard
C-v	quoted-insert
C-w	unix-word-rubout
C-y	yank
C-]	character-search
C-_	undo
C-?	backward-delete-char

key binding แบบ M-	
M-C-g	abort
M-C-h	backward-kill-word
M-C-i	tab-insert
M-C-j	vi-editing-mode
M-C-m	vi-editing-mode
M-C-r	revert-line
M-C-y	yank-nth-arg
M-C-[	complete
M-C-]	character-search-backward
M-space	set-mark
M-#	insert-comment
M-&	tilde-expand
M-*	insert-completions
M--	digit-argument
M-.	yank-last-arg
M-<	beginning-of-history
M-=	possible-completions
M->	end-of-history
M-?	possible-completions
M-b	backward-word
M-c	capitalize-word
M-d	kill-word
M-d	forward-word
M-l	downcase-word
M-n	non-incremental-forward-search-history
M-p	non-incremental-reverse-search-history
M-r	revert-line
M-t	transpose-words
M-u	upcase-word
M-y	yank-pop
M-\	delete-horizontal-space
M-~	tilde-expand
M-C-?	backward-kill-word
M-_	yank-last-arg

พิเศษ. บรรทัดที่ 1 เป็นการรวมค่าเริ่มต้นจากไฟล์ `/etc/inputrc` เข้ามาในไฟล์. ส่วนบรรทัดที่ 13 เป็นการระบุเงื่อนไขแยก key binding ตามโปรแกรมที่ใช้. ในตัวอย่างถ้าโปรแกรมที่ใช้คือ `gnuplot`, `M-m` จะแทน "`plot sin(x)\r`" ในบรรทัดคำสั่งให้.

นอกจากจะสร้างเงื่อนไขตามโปรแกรมที่ใช้แล้ว, ยังสามารถสร้างเงื่อนไขแบ่งตาม mode (`emacs` หรือ `vi`) และประเภทของเทอร์มินอลได้ด้วย. ให้อ่านรายละเอียดเพิ่มเติมจาก online manual หรือ info ของ readline.

โปรแกรมนี้ใช้ readline หรือไม่?

โปรแกรมที่มีอินเทอร์เฟสแบบ CLI เช่น `bash`, `wc`, `gnuplot` ฯลฯ มักจะใช้ไลบรารี readline. ดังนั้นจะตรวจสอบดูว่าโปรแกรมนั้นใช้ไลบรารี readline หรือไม่ก็ลองใช้ key binding เช่น `C-p`, `C-b` ฯลฯ หรือมีความสามารถเติมเต็มชื่อไฟล์หรือไม่. ถ้าใช้ได้

แสดงว่าโปรแกรมนั้นน่าจะใช้ไลบรารี readline.

วิธีตรวจอีกวิธีหนึ่งคือดูจาก *shared library* ที่โปรแกรมนั้นใช้. โดยปรกติโปรแกรมที่ใช้ในลินุกซ์มันเป็นโปรแกรมแบบ dynamic, กล่าวคือจะเรียกรวมภาษาเครื่องกลส่วนที่เป็นไลบรารีเมื่อเรียกใช้. วิธีตรวจสอบว่าโปรแกรมนั้นใช้ shared library อะไรได้ด้วยคำสั่ง `ldd`. โปรแกรม `ldd` เป็นโปรแกรมที่ใช้แสดง shared library ที่โปรแกรมใช้.

บ้างก็เรียก shared library ว่า dynamic library.

shared library ▶ ไฟล์ไบนารีที่เป็นส่วนของไลบรารี. โปรแกรมที่สร้างไม่ต้องรวมส่วนที่เป็นไลบรารีเข้าไปในตัวโปรแกรม. เรียกอีกอย่างหนึ่งว่า dynamic library.

`ldd` อ้างอิงหน้า 409

ตัวอย่างที่ 2.89: ใช้ `ldd` ตรวจสอบ shared library

```
$ ldd /usr/bin/gnuplot↵
        linux-gate.so.1 =>  (0xffffe000)
        lib3dkit.so.1 => /usr/lib/lib3dkit.so.1 (0x40015000)
        libvgagl.so.1 => /usr/lib/libvgagl.so.1 (0x4002a000)
        libplot.so.2 => /usr/lib/libplot.so.2 (0x40038000)
        libreadline.so.4 => /lib/libreadline.so.4 (0x4016b000)
...
```

จะเห็นว่าผลลัพธ์ของคำสั่ง `ldd` มีบรรทัด `libreadline` อยู่หมายความว่าโปรแกรมนั้นใช้ไลบรารี readline.

2.7.6 ไฟล์ `$HOME/.bash_profile`

ตัวอย่างที่ 2.90: เนื้อหาของไฟล์ `.bash_profile`

```
     1  # /etc/skel/.bash_profile:
     2  # $Header: /home/cvsroot/gentoo-src/rc-scripts/etc/skel/.bash_profile,v
1.10 2002/11/18 19:39:22 azarah Exp $
     3
     4  #This file is sourced by bash when you log in interactively.
     5  [ -f ~/.bashrc ] && . ~/.bashrc
```

ไฟล์ตั้งค่าเริ่มต้นของล็อกอินเชลล์อีกไฟล์หนึ่งได้แก่ไฟล์ `.bash_profile`. ในตัวอย่าง 2.7.6 เป็นไฟล์ที่สร้างขึ้นโดยระบบตอนที่สร้างบัญชีผู้ใช้. ถ้ามีสิ่งที่ต้องการทำหรือตั้งค่าพิเศษสำหรับล็อกอินเชลล์เฉพาะบุคคลก็ให้เขียนในไฟล์นี้. ในตัวอย่างไม่มีการปรับแต่งอะไรเป็นพิเศษนอกจากอ่านไฟล์ตั้งค่าเริ่มต้นจากไฟล์ `.bashrc` เท่านั้น.

2.7.7 ไฟล์ `$HOME/.bashrc`

ตัวอย่างที่ 2.91: เนื้อหาของไฟล์ `.bashrc`

```
     1  # /etc/skel/.bashrc:
     2  # $Header: /home/cvsroot/gentoo-src/rc-scripts/etc/skel/.bashrc,v
1.8 2003/02/28 15:45:35 azarah Exp $
     3
     4  # This file is sourced by all *interactive* bash shells on startup.  This
     5  # file *should generate no output* or it will break the scp and rcp commands.
     6
     7  # colors for ls, etc.
     8  eval ‘dircolors -b /etc/DIR_COLORS‘
     9  alias d="ls --color"
    10  alias ls="ls --color=auto -F"
    11  alias ll="ls --color -l"
```

```
12
13  alias rm='rm -i'
14  alias cp='cp -i'
15  alias mv='mv -i'
16
17  # Change the window title of X terminals
18  case $TERM in
19          xterm*|rxvt|Eterm|eterm)
20                  PROMPT_COMMAND='echo -ne "\033]0;${USER}@${HOSTNAME%%.*}:
${PWD/$HOME/~}\007"'
21                  ;;
22          screen)
23                  PROMPT_COMMAND='echo -ne "\033_${USER}@${HOSTNAME%%.*}:
${PWD/$HOME/~}\033\\"'
24                  ;;
25  esac
```

ไฟล์ .bashrc นี้ก็เป็นตัวอย่างไฟล์ที่เอามาจากลินุกซ์ Gentoo เช่นกัน. ไฟล์นี้จะเป็นไฟล์ตั้งค่าเริ่มต้นเชลล์เชิงโต้ตอบ. และเนื่องจากในไฟล์ .bash_profile มีการระบุให้อ่านไฟล์นี้เข้าไปด้วย, เพราะฉะนั้นค่าทั้งหลายที่ปรับแต่งในไฟล์นี้จะมีผลกับเชลล์ล็อกอินด้วย. การปรับแต่งเชลล์ไฟล์นี้มักจะเป็นการตั้งค่าตัวแปรสภาพแวดล้อมและนามแฝง, ไม่ควรสั่งคำสั่งที่ทำให้เกิดข้อมูลแสดงออกทางหน้าจอ. มิเช่นนั้นจะทำให้ scp และ rcp ไม่ทำงาน.

scp และ rcp เป็นโปรแกรมก็อปปี้ไฟล์ผ่านทางเน็ตเวิร์ก.

dircolors อ้างอิงหน้า 410

บรรทัดที่ 8 ตั้งให้คำสั่ง ls ใช้สีในการแสดงไฟล์หรือไดเรกทอรี. คำสั่ง dircolors ปรับแต่งสีที่จะใช้กับคำสั่ง ls. คำสั่ง dircolors จะแสดงเนื้อหาการตั้งค่าตัวแปรสภาพแวดล้อม LS_COLORS ทาง stdout. ตัวแปรสภาพแวดล้อมนี้จะมีผลเมื่อใช้คำสั่ง ls กับตัวเลือก --color. เนื่องจากคำสั่ง dircolors แสดงวิธีการตั้งค่าตัวแปรสภาพแวดล้อมทาง stdout เท่านั้นจึงต้องใช้คำสั่ง dircolors ใน back quote (‘) และใช้ eval?? ตีความกระทำการผลลัพธ์ของ dircolors ให้เชลล์รับรู้.

บรรทัดที่ 9-15 เป็นการสร้าง alias ของคำสั่งต่างๆ. ในที่นี้มีการสร้าง alias ของคำสั่ง ls ซ้ำตัวเอง (บรรทัดที่ 10). จะเห็นได้ว่ามีการใช้ตัวเลือก --color กับคำสั่ง ls และก่อนหน้านี้ได้ทำการปรับแต่งสีที่ใช้กับคำสั่ง ls ไปแล้ว, ถ้าเราสั่งคำสั่ง ls เดี่ยวๆก็จะแสดงสีแยกตามประเภทของไฟล์ที่แสดงให้ด้วย.

บรรทัดที่ 18-25 เป็นการตั้งค่าตัวแปรสภาพแวดล้อม PROMPT_COMMAND. ค่าของตัวแปรนี้เป็นคำสั่งที่จะถูกกระทำการทุกครั้งก่อนที่จะแสดงเชลล์พรอมต์. บรรทัดที่ 20 เป็นการตั้งค่า PROMPT_COMMAND ให้ตั้งชื่อ title bar ของเทอร์มินอลทุกครั้งก่อนที่จะแสดงพรอมต์. การตั้งค่า title bar มีรูปแบบดังนี้.

```
echo -ne "\033]0;TITLE\007"
```

การตั้งชื่อ title bar จะคล้ายกับการปรับแต่งสีที่ใช้ในพรอมต์คือมีการใช้อักขระ ESC (\033) แต่วลีหลังจากนั้นจะต่างกัน. \007 คืออักขระ BELL ที่เขียนอยู่ในรูปของเลขฐานแปด. ถ้ามีการติดตามคำสั่งในเชลล์เหมือนในตัวอย่างที่ 2.85 (หน้า 80) ก็จะเห็นว่ามีการสั่งคำสั่ง echo นี้ก่อนที่จะแสดงเชลล์พรอมต์ทุกครั้ง.

2.8 สรุปท้ายบท

- ลินุกซ์เป็นระบบปฏิบัติการที่แยกอินเทอร์เฟสอเช่น เชลล์หรือ X วินโดว์ออกจากตัวระบบปฏิบัติการ. อินเทอร์เฟสเหล่านี้เป็นเพียงโปรแกรมตัวกลางที่ใช้ติดต่อระหว่างผู้ใช้กับเคอร์เนล.

- เชลล์เป็นโปรแกรมแปลคำสั่งและใช้อินเทอร์เฟสที่เรียกว่า CLI (Command Line Interface). คำสั่งในที่นี้แยกออกได้เป็นสองประเภทใหญ่ ๆคือ คำสั่งภายใน (build-in command) และคำสั่งภายนอกหรือที่เรียกกันว่าโปรแกรม.

- เชลล์ bash มีความสามารถในการสร้างนามแฝง (alias), ควบคุมจ็อบ (job), บันทึกประวัติคำสั่ง (history), การเติมเต็มไฟล์หรือคำสั่ง (completion), แก้ไขบรรทัดคำสั่ง (line editing) ฯลฯ. ความสามารถบางอย่างเป็นความสามารถที่เกิดจากการใช้ไลบรารี readline ซึ่งโปรแกรมอื่น ๆที่มีอินเทอร์เฟสแบบ CLI บางโปรแกรมก็ใช้ไลบรารีนี้ด้วย. การทำความคุ้นเคยกับไลบรารี readline จะช่วยให้ใช้งานบรรทัดคำสั่งคล่องและสะดวกขึ้น.

- เชลล์ที่ใช้กันอยู่โดยทั่วไปแบ่งได้เป็น 2 ชนิดใหญ่ได้แก่ เชลล์ล็อกอิน (login shell) และเชลล์เชิงโต้ตอบ (interactive shell). เชลล์ทั้งสองแบบมีไฟล์ตั้งค่าเริ่มต้นที่ต่างกัน. เวลาใช้งานควรจะตระหนักเสมอว่าเชลล์ที่ใช้อยู่เป็นเชลล์ประเภทใด.

- ข้อมูลที่ประมวลโดยเชลล์มักเป็นข้อมูลแบบเท็กซ์. ผู้ใช้สามารถป้อนผลลัพธ์ของโปรแกรมหนึ่งให้อีกโปรแกรมหนึ่งได้โดยไปป์และบันทึกหรืออ่านข้อมูลจากไฟล์ได้โดยการรีไดเรก.

- คำสั่งหลักในลินุกซ์มักจะรับข้อมูลจาก stdin (คีย์บอร์ด) ถ้าไม่มีชื่อไฟล์เป็นอาร์กิวเมนต์. และมักจะแสดงผลที่ประมวลออกทาง stdout (หน้าจอ). ถ้าคำสั่งที่ใช้เกิดข้อผิดพลาด, จะแสดงข้อผิดพลาดทาง stderr (หน้าจอ).

- file descriptor คือตัวเลขจำนวนเต็มที่เฉพาะในแต่ละโปรเซสใช้อ้างอิงไฟล์ที่ใช้งาน. file descriptor ของ stdin, stdout และ stderr ได้แก่ 0, 1 และ 2 ตามลำดับ. เราสามารถใช้ file descriptor เหล่านี้รีไดเรกข้อมูลหาซึ่งกันและกันได้.

- ลินุกซ์ (ยูนิกซ์ก็เช่นกัน) จะมียูทิลิตี้หรือเครื่องมือในการทำงานต่าง ๆ. โปรแกรมเหล่านี้จะทำงานเฉพาะอย่าง, และทำงานอย่างดี. การแก้ปัญหาที่ไม่อาจใช้โปรแกรมเดียวแก้ได้อาจจะแก้ได้ด้วยการใช้โปรแกรมเล็ก ๆร่วมกันแก้ปัญหาเช่นใช้ไปป์หรือเชลล์สคริปต์.

บทที่ 3
ไฟล์

"Everything in the Unix system is a file."
Brian W. Kernighan, Rob Pike [29]

ในหนังสือ The Unix Programming Environment [29] ซึ่งจัดเป็นหนังสือต้องมีเล่มหนึ่งสำหรับผู้ใช้ยูนิกซ์กล่าวไว้ว่า "ทุกอย่างในยูนิกซ์คือไฟล์". ประโยคยังเป็นความจริงสำหรับระบบปฏิบัติการลินุกซ์ด้วย. ในการใช้ลินุกซ์ในแต่ละวัน, การทำงานกับไฟล์เป็นสิ่งที่หลีกเลี่ยงไม่ได้ ตั้งแต่การสร้างไฟล์, แก้ไขไฟล์ ฯลฯ. ดังนั้นจึงมีความจำเป็นที่จะต้องเรียนรู้เกี่ยวกับไฟล์ในลินุกซ์เป็นอย่างดีในบทนี้.

3.1 พาร์ทิชัน, และระบบไฟล์

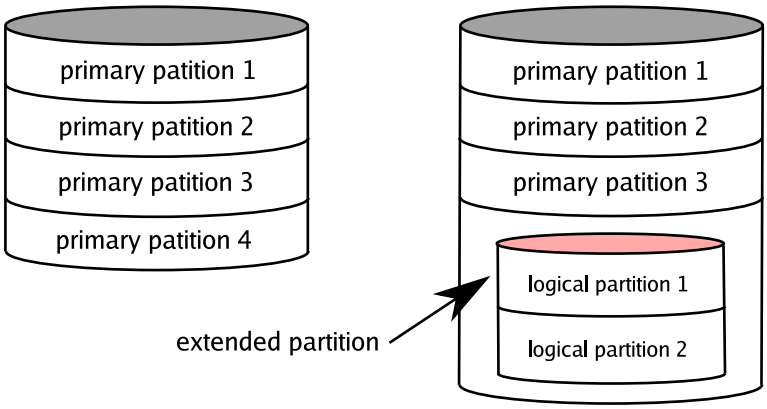
ปรกติแล้ว, ไฟล์ที่ใช้ในระบบปฏิบัติการลินุกซ์จะอยู่ในหน่วยความถาวรเช่นฮาร์ดดิสก์. และก่อนที่จะสร้างไฟล์ในฮาร์ดดิสก์ได้นั้นต้องสร้างพาร์ทิชันและสร้างระบบไฟล์ให้กับพาร์ทิชันนั้นก่อนจึงจะสามารถกระทำการต่าง ๆกับไฟล์ได้.

3.1.1 พาร์ทิชัน

พาร์ทิชัน (partition) คือหน่วยย่อยของฮาร์ดดิกส์, หรือฮาร์ดดิสก์ย่อยแบบ logical ที่แบ่งมาจากฮาร์ดดิกส์ที่มีอยู่จริง. พาร์ทิชันยังแบ่งได้เป็นสองประเภทคือ*พาร์ทิชันหลัก (primary partition)* และ*พาร์ทิชันเสริม (extended partition)*. เราสามารถแบ่งพาร์ทิชันหลักได้มากที่สุด 4 ส่วน. ถ้าต้องการแบ่งพาร์ทิชันมากกว่านั้นต้องให้พาร์ทิชันหลักหนึ่งในสี่นั้นเป็นพาร์ทิชันเสริมตามที่แสดงในรูปที่ 3.1. ในพาร์ทิชันสามารถสร้างพาร์ทิชันที่เรียกว่าพาร์ทิชัน logical ต่อไปได้อีกเรื่อย ๆ.

ข้อมูลตำแหน่งของพาร์ทิชันเหล่านี้จะเก็บอยู่ในส่วนที่เรียกว่า *partition table* และโปรแกรมที่ทำหน้าที่จัดการข้อมูลพาร์ทิชันของฮาร์ดดิสก์ได้แก่โปรแกรม `fdisk`. โปรแกรมนี้จะใช้โดยผู้ดูแลระบบหรือ root เพื่อจัดการแบ่งพาร์ทิชันและกำหนดประเภทของพาร์ทิชัน.

❑ `fdisk` อ้างอิงหน้า 405



รูปที่ 3.1: พาร์ทิชันหลักและพาร์ทิชันเสริม.

ตัวอย่างที่ 3.1: ใช้ `fdisk` แสดง partition table.

```
# fdisk /dev/hdb↵                ← ฮาร์ดดิสก์แบบ EIDE ตัวที่สอง. ตัวแรกจะมีชื่อเป็น hda

The number of cylinders for this disk is set to 38166.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): p↵

Disk /dev/hdb: 40.0 GB, 40020664320 bytes
64 heads, 32 sectors/track, 38166 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hdb1               1       19074    19531760   83  Linux
/dev/hdb2           19075       37672    19044352   83  Linux
/dev/hdb3           37673       38166      505856   82  Linux swap

Command (m for help): q↵

# ▮
```

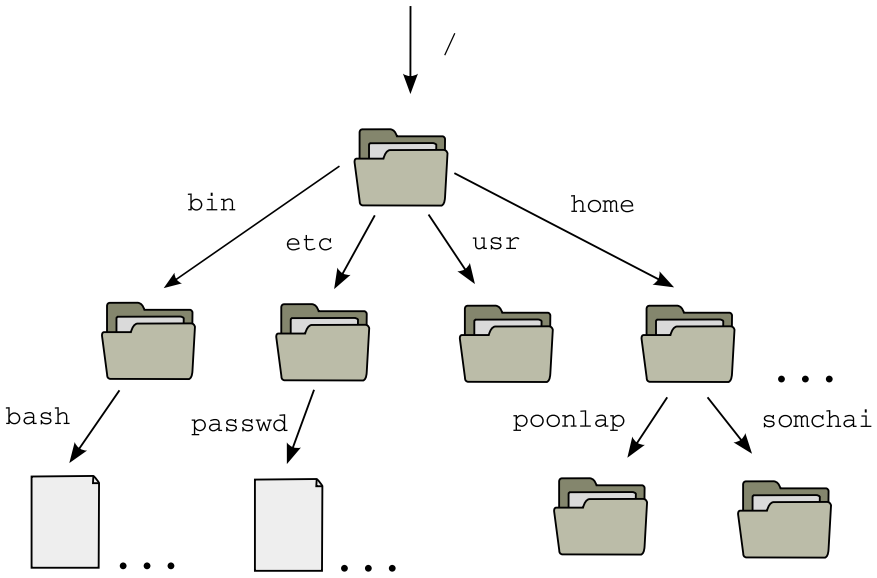
สำหรับระบบปฏิบัติการลินุกซ์แล้ว, ควรจะแบ่งพาร์ทิชันอย่างน้อย 2 ส่วนขึ้นไป. ส่วนแรกใช้สำหรับเป็นพื้นที่ที่จะสร้างระบบไฟล์เพื่อเก็บข้อมูลที่จำเป็นต่อไป. ส่วนพาร์ทิชันทีเหลือใช้เป็นพื้นที่สำหรับ virtual memory ซึ่งมักจะเรียกกันว่า *swap partition*. swap partition มีไว้สำหรับให้เคอร์เนลเก็บข้อมูลบางส่วนที่อยู่ในหน่วยความจำชั่วคราว (memory) ลงในฮาร์ดดิสก์ได้. และเรียกข้อมูลส่วนนั้นที่อยู่ในฮาร์ดดิสก์กลับมาในหน่วยความจำใหม่เมื่อต้องการ. วิธีการนี้ทำให้จำนวนหน่วยความจำโดยรวมไม่ขาดแคลน, และในบางกรณีสามารถช่วยให้ประสิทธิภาพการทำงานของโปรแกรมบางอย่างดีขึ้นด้วย.

virtual memory ▶

ผู้ใช้สามารถระบบที่ไม่มี swap ก็ได้ แต่ไม่แนะนำ. พื้นที่ที่เป็น swap นี้ไม่จำเป็นต้องเป็นพาร์ทิชันแต่อาจจะเป็นไฟล์ก็ได้. swap ที่เป็นไฟล์นั้นมักจะสร้างเมื่อต้องการเพิ่มจำนวน swap ภายหลังที่สร้างพาร์ทิชันไปเรียบร้อยแล้ว.

3.1.2 ระบบไฟล์

การสร้างพาร์ทิชันเป็นเพียงแค่การแบ่งฮาร์ดดิสก์ให้มีจำนวนและขนาดตามที่ต้องการ. พาร์ทิชันที่สร้างขึ้นมานั้นต้องผ่านการสร้าง*ระบบไฟล์ (file system)* ก่อนจึงจะใช้งาน, อ่านเขียนไฟล์ได้. การสร้างระบบไฟล์ในพาร์ทิชันเป็นการสร้างข้อมูลพื้นฐานเพื่อที่จะเก็บข้อมูลเป็นไฟล์, จัดโครงไดเรกทอรีต่อไป. การจัดเก็บไฟล์และไดเรกทอรีในระบบปฏิบัติการลินุกซ์และยูนิกซ์จะมีโครงสร้างเป็นแผนภาพต้นไม้ตามรูปที่ 3.2.



รูปที่ 3.2: โครงสร้างไฟล์และไดเรกทอรี

ระบบไฟล์ที่ออกแบบมาสำหรับลินุกซ์เพื่อสร้างโครงสร้างไฟล์และไดเรกทอรีมีหลายประเภทเช่น ext2 (Second Extended Filesystem), ext3, xfs, Reiserfs ฯลฯ. ระบบไฟล์เหล่านี้จะเก็บข้อมูลเกี่ยวกับตำแหน่งของไฟล์ต่าง ๆและข้อมูลที่เกี่ยวข้องอื่น ๆเช่น ข้อมูลเกี่ยวกับเจ้าของไฟล์, สิทธิ์ในการใช้ไฟล์, เวลาบันทึกไฟล์ เป็นต้น.

การสร้างระบบไฟล์ให้กับพาร์ทิชันที่แบ่งไว้แล้วจะใช้คำสั่ง `mkfs` ซึ่ง root มักจะเป็นผู้ใช้คำสั่งนี้สร้างระบบไฟล์.

mkfs อ้างอิงหน้า 406

ตัวอย่างที่ 3.2: การสร้างระบบไฟล์.

```
# mkfs -t ext2 /dev/fd0↵
```

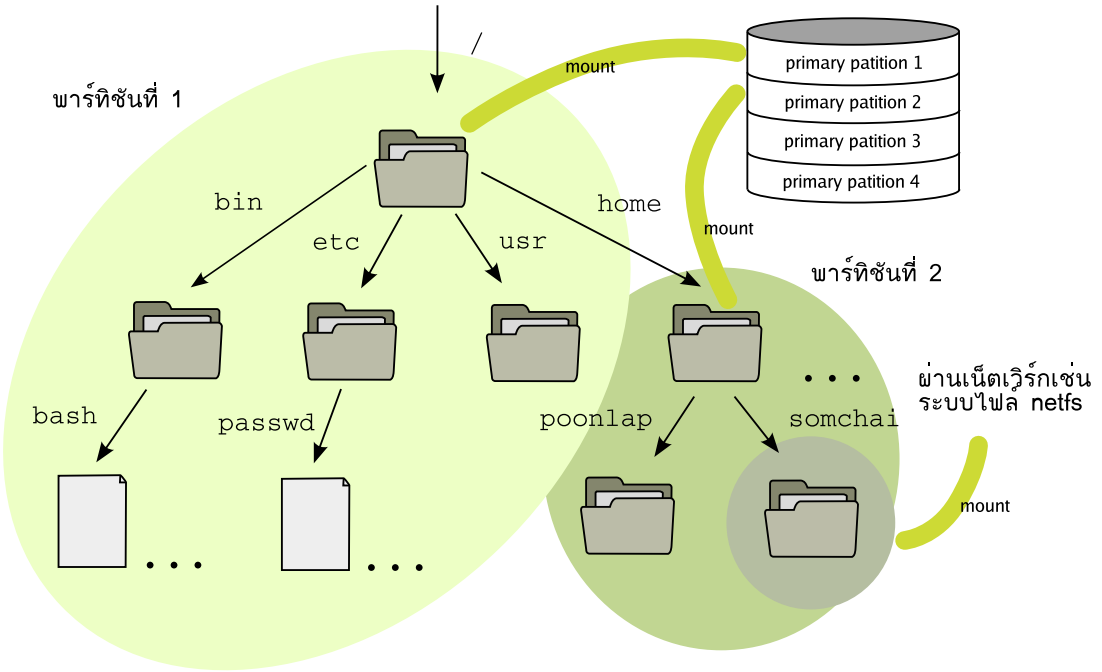
ในความเป็นจริงแล้วคำสั่ง `mkfs` เป็นเพียง front-end ของโปรแกรมที่สร้างระบบไฟล์อื่น ๆเช่น `mkfs.ext2`, `mkfs.xfs` ฯลฯ. ดังนั้นเราอาจจะเรียกใช้โปรแกรม `mkfs.ext2` โดยตรงก็ได้.

front-end ▶ โปรแกรมที่เรียกใช้โปรแกรมเฉพาะต่ออีกทีหนึ่ง. เช่นโปรแกรม `mkfs` เรียกใช้โปรแกรม `mkfs.ext2` หรือ `mkfs.xfs` อีกทอดหนึ่งแล้วแต่ตัวเลือกที่ใช้กับ `mkfs`. โปรแกรม front-end เหล่านี้เป็นเพียงอินเทอร์เฟส, ไม่ได้เป็นตัวที่ทำงานหลักเอง.

นอกจากระบบไฟล์ที่ออกแบบมาสำหรับลินุกซ์แล้ว, เรายังสามารถสร้างหรือใช้ระบบไฟล์อื่น ๆเช่นระบบไฟล์สำหรับ DOS, vfat (วินโดวส์), ntfs (วินโดวส์) ฯลฯ และนำมาใช้กับลินุกซ์ได้ด้วย. ปรกติเราจะสร้างระบบไฟล์เหล่านี้ให้กับฮาร์ดดิสก์พกพา, USB flash

memory, ฟล็อปปี้ดิสก์ เพื่อที่จะได้ใช้กับเครื่องคอมพิวเตอร์วินโดวส์ได้ด้วย.

3.1.3 mount และ unmount



รูปที่ 3.3: mount พาร์ทิชันเข้าในโครงสร้างไฟล์.

ขั้นตอนต่อไปเพื่อที่จะใช้พาร์ทิชันที่สร้างระบบไฟล์เรียบร้อยแล้วเก็บบันทึกไฟล์ได้ แก่การ *mount* ระบบไฟล์นั้นเข้าไปในโครงสร้างไฟล์และไดเรกทอรีที่แสดงในรูปที่ 3.2.

การ mount คือการนำระบบไฟล์ที่สร้างในพาร์ทิชัน (ดีไวซ์, ฮาร์ดดิสก์) ไปปะติดกับโครงสร้างไดเรกทอรีที่ในตำแหน่งที่ต้องการ. ตำแหน่งที่นำระบบไฟล์ไปปะติดนี้เรียกว่า *จุด mount (mount point)* ซึ่งจะเป็นไดเรกทอรีที่สร้างเตรียมไว้. ตัวอย่างเช่นรูปที่ 3.3, ระบบไฟล์ที่สร้างไว้บนพาร์ทิชันที่ 1 จะปะติดกับไดเรกทอรี / (รูท). ไดเรกทอรี `/home` สร้างในพาร์ทิชันที่ 1 แต่หลังจากที่ mount ระบบไฟล์ที่สร้างไว้บนพาร์ทิชันที่ 2 ปะติดกับไดเรกทอรี `/home` แล้ว, ข้อมูลที่อยู่ใต้ไดเรกทอรี `/home` จะไปเก็บอยู่ในพาร์ทิชันที่ 2. การ mount ไม่จำเป็นต้อง mount กับพาร์ทิชันหรือฮาร์ดดิสก์เสมอไป, อาจจะ mount ระบบไฟล์ที่อยู่ในเครื่องคอมพิวเตอร์อื่น ๆผ่านทางเน็ตเวิร์กก็ได้. ตอนที่เครื่องเริ่มทำงานจะ mount ระบบไฟล์ต่าง ๆตามที่กำหนดไว้ในไฟล์ `/etc/fstab`.

จะเป็นไดเรกทอรีที่ว่างเปล่าหรือไม่ว่างเปล่าก็ได้.

ระบบไฟล์ที่ใช้เน็ตเวิร์กเช่น smbfs (Samba filesystem), netfs (network filesystem) เป็นต้น.

mount อ้างอิงหน้า 406

คำสั่ง `mount` เป็นคำสั่งที่ใช้ mount ระบบไฟล์เข้าไปในไดเรกทอรีหรือดูระบบไฟล์และไดเรกทอรีที่ mount อยู่.

ตัวอย่างที่ 3.3: คำสั่ง `mount` แสดงระบบไฟล์ที่ mount อยู่.

```
$ mount↵
/dev/hdb1 on / type ext3 (rw,noatime)
```

```
devfs on /dev type devfs (rw)
none on /proc type proc (rw)
none on /sys type sysfs (rw)
none on /dev/pts type devpts (rw)
/dev/hdb2 on /home type ext3 (rw,noatime)
none on /dev/shm type tmpfs (rw)
none on /proc/bus/usb type usbfs (rw)
/dev/hda2 on /mnt/usb type ntfs (rw,uid=1000)
/dev/cdrom on /mnt/cdrom type iso9660 (ro)
```

เนื่องจากการ mount เป็นการนำระบบไฟล์เข้าไปปะติดกับไดเรกทอรีที่ต้องการในโครงสร้างไฟล์และไดเรกทอรี, ดังนั้นจากมุมมองของผู้ใช้แล้วเราจึงมองโครงสร้างไฟล์และไดเรกทอรีได้เป็นหนึ่งเดียวคือในรูปที่ 3.2, ไม่มีการบอกว่าไดเรกทอรีนี้เป็นของพาร์ทิชันนี้ชัดเจนเหมือนระบบปฏิบัติการวินโดวส์ที่ใช้ตัวอักษรภาษาอังกฤษกำกับพาร์ทิชัน.

คำสั่ง `mount` ที่ใช้ mount ระบบไฟล์ปะติดกับไดเรกทอรีต้องใช้ตัวเลือก `-t` (type) เพื่อระบุระบบไฟล์และมีชื่อไฟล์เป็นอาร์กิวเมนต์ดังต่อไปนี้.

ตัวอย่างที่ 3.4: การ mount ฟล็อปปี้ดิสก์.

```
# mount -t ext2 /dev/fd0 /mnt/floppy -o ro↵
```

จากตัวอย่างเป็นการ mount ฟล็อปปี้ปะติดกับไดเรกทอรี `/mnt/floppy`. ตัวเลือก `-o` (option) เป็นการระบุรายละเอียดปลีกย่อยของการ mount. ในตัวอย่าง “ro” คือ read-only ให้อ่านได้อย่างเดียวเท่านั้น, ไม่สามารถเขียนได้. รายละเอียดปลีกย่อยนี้บางอย่างจะขึ้นกับระบบไฟล์ที่ใช้, ให้อ่านรายละเอียดเพิ่มเติมจาก `mount(8)`. หลังจากที่ mount เรียบร้อยแล้วผู้ใช้ก็สามารถอ่านไฟล์ที่อยู่ในฟล็อปปี้ได้จากไดเรกทอรี `/mnt/floppy`.

ต้องสร้างไดเรกทอรีที่ต้องการ mount เตรียมไว้ก่อนเสมอ.

unmount คือการเอาระบบไฟล์ออกจากไดเรกทอรีที่ mount อยู่. คำสั่งที่ใช้สำหรับ unmount ได้แก่ `umount`. ตัวอย่างเช่น

`umount` อ้างอิงหน้า 408

ใหระวังชื่อคำสั่งคือ `umount` ไม่ใช่ `unmount`

ตัวอย่างที่ 3.5: การ unmount.

```
# umount /mnt/floppy↵
```

จากมุมมองของระบบปฏิบัติการ, ถ้ามีข้อมูลที่ต้องเขียนลงระบบไฟล์นั้นแต่ข้อมูลนั้นยังอยู่ในหน่วยความจำชั่วคราว, ก็จะทำการเขียนลงระบบไฟล์นั้น. การกระทำนี้เรียกว่า *sync* (synchronize) ข้อมูลเพื่อให้ข้อมูลถูกต้องก่อนจะเอาระบบไฟล์ออกจากโครงสร้างไดเรกทอรี. นอกจากจะ sync ข้อมูลแล้ว, เคอร์เนลยังตรวจสอบด้วยว่ามีไฟล์ใดบ้างที่ยังเปิดอยู่หรือมีโปรแกรมใช้ไฟล์ที่อยู่ในระบบไฟล์นั้นหรือไม่. ถ้ามีก็ไม่สามารถ unmount ได้และจะเกิด error ว่า “device is busy”. ในกรณีนี้ต้องกำจัดโปรเซสที่เปิดไฟล์ใต้ไดเรกทอรีนั้นให้หมดก่อนจึงจะ unmount ได้.

ตารางที่ 3.1: ระบบไฟล์ที่ใช้ในลินุกซ์.

ระบบไฟล์	คำอธิบาย
ext2	เป็นระบบไฟล์มาตรฐานสำหรับลินุกซ์. เดิมเป็นระบบไฟล์ที่นิยมใช้ในลินุกซ์. แต่ในปัจจุบันเริ่มใช้ extended 3 ซึ่งเป็นระบบไฟล์ที่พัฒนาต่อจาก extended 2 อีกที.
ext3	เป็นระบบไฟล์แบบ ext2 ที่มีคุณสมบัติ journal เพิ่มเติม.
fat, vfat	ระบบไฟล์ที่ใช้ใน DOS, วินโดวส์.
ntfs	ระบบไฟล์ที่ใช้ในวินโดวส์ 2000 หรือ XP.
iso9660	เป็นระบบไฟล์ที่มักใช้กับแผ่นซีดี.
smbfs	ระบบไฟล์แบบเน็ตเวิร์ก, ใช้สำหรับ mount ไดเรกทอรีที่แชร์จากเครื่องวินโดวส์หรือเซิฟร์เวอร์ที่ใช้ samba.
nfs	ระบบไฟล์ Network File System. สำหรับใช้ไฟล์ผ่านเน็ตเวิร์กระหว่างเครื่องยูนิกซ์หรือลินุกซ์.
reiserfs, xfs, jfs	ระบบไฟล์สมัยใหม่ที่ออกแบบโดยคำนึงถึงประสิทธิภาพการทำงานด้านต่าง ๆเช่นความเร็ว, ขีดจำกัดของขนาดไฟล์, ระบบ journal เป็นต้น.

3.2 ไฟล์

ในภาษาไทยอาจแปลคำว่า file เป็นแฟ้มข้อมูล.

ไฟล์ (file) คืออะไร? ไฟล์คือกลุ่มก้อนของข้อมูล. ข้อมูลนี้มีจุดเริ่มต้นและจุดจบ. ตั้งแต่จุดเริ่มต้นจนถึงจุดจบจะเป็นข้อมูลต่อเนื่องไปเรื่อย ๆเป็น*สายข้อมูล (data stream)*. สรุปแนวคิดของไฟล์สั้นได้ว่า, ไฟล์คือสายข้อมูล. สายข้อมูลนี้ไม่มีการแยกแยะประเภทของข้อมูลว่าจะเป็น*ไบนารี (binary)* หรือ*เท็กซ์ (text)*. ความแตกต่างของข้อมูลไบนารีและข้อมูลเท็กซ์เป็นเพียงความแตกต่างที่มนุษย์สังเกตุเห็น, คือเป็นข้อมูลอักขระที่มนุษย์อ่านได้หรืออ่านไม่ได้เท่านั้น. สำหรับคอมพิวเตอร์แล้ว, ข้อมูลคือค่าที่แสดงในหน่วยของ*ไบต์ (byte)* .จริงไม่มีการแยกแยะความแตกต่างระหว่างข้อมูลไบนารีกับข้อมูลเท็กซ์.

byte ▶
หน่วยของข้อมูล. จำนวนหนึ่งไบต์สามารถแสดงหรือเก็บค่าได้ตั้งแต่ 0 ถึง 255.

3.2.1 ชื่อไฟล์

ขีดจำกัดนี้ดูได้จาก `limit.h` ที่อยู่ในรหัสต้นฉบับของเคอร์เนล.

ไฟล์ต้องมีชื่อและจำนวนอักขระที่สามารถใช้เป็นชื่อไฟล์มีขนาดไม่เกิน 255 ตัวอักษร. ชื่อไฟล์สามารถใช้อักษรหรือเครื่องหมายได้แต่ในทางปฏิบัติควรจะหลีกเลี่ยงเครื่องหมายที่มีความหมายพิเศษสำหรับเชลล์เช่น ช่องไฟ, วงเล็บ () เป็นต้น. เนื่องจากช่องไฟเป็นตัวแบ่งอาร์กิวเมนต์ในเชลล์จึงไม่ควรใช้ช่องไฟในชื่อไฟล์ถ้าไม่มีความจำเป็นจริง ๆ. โดยทั่วไปมักจะใช้เครื่องหมาย underscore (_) แทนช่องไฟ. ตัวอย่างเช่นแทนที่จะใช้ชื่อไฟล์ “`important file`” ก็ใช้ “`important_file`” แทน. นอกจากนี้ลินุกซ์ยังแยกแยะความแตกต่างระหว่างอักษรตัวใหญ่ (capital letter) และอักษรตัวเล็ก (small letter) ด้วย.

internationalization ▶
ความสามารถที่โปรแกรมสามารถปรับตัวให้เข้ากับสภาพแวดล้อมภาษาและวัฒนธรรมที่ผู้ใช้กำหนด. ตัวอย่างเช่นโปรแกรมสามารถแสดงผลนอกเหนือจากภาษาอังกฤษได้ถ้าสภาพแวดล้อมของผู้ใช้ไม่ใช่ภาษาอังกฤษ. Internationalization มีคำย่อว่า i18n โดยตัวเลข 18 คือจำนวนอักษรระหว่างอักษร i และ n.

สำหรับระบบที่ปรับแต่งให้ใช้*ภาษานานาชาติ (internationalization, i18n)* ได้ก็สามารถใช้ภาษาอื่นนอกจากภาษาอังกฤษเป็นชื่อไฟล์ได้ด้วย. แต่อย่างไรก็ตามหากมีการก็อปปี้ไฟล์นั้นข้ามระบบ, หรือใช้กับคอมพิวเตอร์อื่น ๆก็ควรจะใช้ภาษาอังกฤษเป็นชื่อไฟล์เพื่อ

หลีกเลี่ยงปัญหาที่เกิดขึ้นจากเครื่องคอมพิวเตอร์อื่นที่อาจจะไม่สนับสนุนการใช้ภาษานานาชาติ.

ส่วนขยายชื่อไฟล์

เนื่องจากไฟล์คือสายข้อมูล, และชื่อไฟล์จะใช้ตัวอักษรหรือเครื่องหมายอะไรก็ได้, หมายความว่า*ส่วนขยายชื่อไฟล์ (filename extension)* เช่น `.txt`, `.zip` ฯลฯ เป็นเพียงแค่ส่วนหนึ่งของชื่อไฟล์และไม่มีความหมายพิเศษสำหรับโปรแกรมที่ใช้ไฟล์นั้น. กล่าวคือไม่ว่าไฟล์จะชื่อ `file.txt` หรือ `file` อย่างเดียว, ถ้าไม่เปิดดูเนื้อหาของไฟล์ข้างในก็จะไม่รู้ว่าไฟล์นั้นเป็นไฟล์แบบไหน.

ในลินุกซ์จะมีคำสั่ง `file` เพื่อใช้ตรวจสอบประเภทของไฟล์. ตัวอย่างเช่น,

`file` อ้างอิงหน้า 412

ตัวอย่างที่ 3.6: การใช้ `file` ตรวจสอบประเภทของไฟล์.

```
$ file /etc/passwd /bin/bash↵
/etc/passwd: ASCII text
/bin/bash:   ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/L
inux 2.4.1, dynamically linked (uses shared libs), stripped
$ file /boot/initrd-2.6.5-gentoo-r1↵
/boot/initrd-2.6.5-gentoo-r1: gzip compressed data, was "initrd-loop", from Unix
, max compression
$ file /dev/hda↵
/dev/hda: symbolic link to ‘ide/host0/bus0/target0/lun0/disc’
$ file /dev/ide/host0/bus0/target0/lun0/disc↵
/dev/ide/host0/bus0/target0/lun0/disc: block special (3/0)
$ file /dev/null↵
/dev/null: character special (1/3)
```

คำสั่งไฟล์มีประโยชน์เช่นในกรณีที่เรามีไฟล์ที่ไม่ส่วนขยายชื่อไฟล์และต้องการรู้ว่าไฟล์นั้นเป็นอะไร, หรือส่วนขยายชื่อไฟล์ไม่ตรงกับประเภทของไฟล์นั้น, ก็ใช้คำสั่ง `file` สำรวจประเภทของไฟล์ได้. คำสั่ง `file` จะมีฐานข้อมูลของไฟล์ประเภทต่าง ๆทำให้สามารถตรวจสอบประเภทของไฟล์ได้อย่างละเอียด. ในบางกรณีสามารถบอกได้ว่าเป็นไฟล์ที่สร้างโปรแกรมอะไรได้ด้วย. ตัวอย่างเช่น,

ตัวอย่างที่ 3.7: การตรวจสอบไฟล์ที่ไม่มีส่วนขยายชื่อไฟล์

```
$ file unknown_file↵
unknown_file: PDF document, version 1.2
```

ในกรณีนี้ก็ทำให้เรารู้ไว้ไฟล์นี้เป็นไฟล์ PDF และต้องใช้โปรแกรมเช่น `acroread`, `xpdf`, `ggv` ฯลฯ ในการเปิดอ่าน. นอกจากนั้นยังทำให้เรารู้ว่าควรเปลี่ยนชื่อโดยเพิ่มส่วนขยายชื่อไฟล์เป็น `.pdf` เพื่อความสะดวกของผู้ใช้ในโอกาสต่อไป.

จากตัวอย่างที่ 3.6 จะเห็นได้ว่าคำสั่ง `file` สามารถแยกประเภทของไฟล์ได้ละเอียด. เช่นถ้าเป็นไฟล์ข้อมูลก็จะบอกได้ว่าเป็นข้อมูลแบบไหน, ของโปรแกรมอะไรเป็นต้น. ถ้าเป็นไฟล์เท็กซ์, คำสั่ง `file` ก็อาจจะบอกได้ว่าไฟล์นั้นมีเนื้อหาเป็น ASCII หรืออย่างอื่น.

การที่คำสั่ง `file` สามารถบอกประเภทของไฟล์ได้ละเอียดเป็นเพราะคำสั่ง `file` ดู

ฐานข้อมูลนี้เก็บไว้ในไฟล์ `/usr/share/misc/file/magic`.

เนื้อหาข้างในแล้วเทียบกับฐานข้อมูลของไฟล์ต่าง ๆที่เตรียมไว้.

จริง ๆแล้วเราสามารถเปิดดูเนื้อหาไฟล์ไม่ว่าจะเป็นไฟล์เท็กซ์หรือไฟล์ไบนารี, และบางครั้งสามารถบอกได้ว่าไฟล์นั้นเป็นไฟล์อะไร. ไฟล์ไบนารีส่วนใหญ่จะมีอักขระ ASCII ปะปนอยู่ด้วย. ในบางกรณีอักขระที่อ่านได้เหล่านี้เป็นตัวบอกใบ้ให้เรารู้ว่าไฟล์นั้นเป็นไฟล์อะไร. คำสั่ง `file` ก็เช่นกันอาศัยตัวบอกใบ้ที่อยู่ในไฟล์ที่ตรวจสอบแล้วเทียบเอากับฐานข้อมูลประเภทของไฟล์ที่เตรียมไว้. ส่วนบอกใบ้นี้เรียกกันว่า *magic number*. โดยทั่วไป magic number จะอยู่ในช่วงต้น ๆของไฟล์เช่น ถ้าเป็นไฟล์รูปภาพแบบ png ก็จะมีข้อมูลหนึ่งไบต์มีค่าเป็น `0x89` แล้วตามด้วยอักขระ ASCII ว่า PNG เป็นต้น. ไฟล์แบบนี้ถ้าเป็นคนดูเองจะง่ายและพอเดาได้เพราะมีอักขระ ASCII เขียนสื่อความหมายไว้. แต่บางกรณี magic number อาจจะเป็นข้อมูลไบนารีล้วน ๆ.

magic number ▶ ข้อมูลหรือค่าที่อยู่ในไฟล์, เป็นเอกลักษณ์พิเศษที่บอกประเภทของไฟล์ได้. ตัวอย่างเช่น magic number ของสคริปต์ที่ใช้ในยูนิกซ์หรือลินุกซ์คืออักขระ `#!`.

3.2.2 ประเภทไฟล์

การแบ่งประเภทของไฟล์ขึ้นอยู่กับมุมมองว่าจะแบ่งด้วยเกณฑ์อย่างไร. ถ้าจะแบ่งด้วยเกณฑ์ของข้อมูลที่มนุษย์สามารถอ่านได้ (แสดงผลทางหน้าจอได้) ก็อาจจะแบ่งได้เป็น *ไฟล์เท็กซ์ (text file)* และ *ไฟล์ไบนารี (binary file)*. ถ้าจะแบ่งไฟล์ไบนารีให้ละเอียดกว่านี้ก็แบ่งได้เป็นไฟล์ภาษาเครื่องกลที่คอมพิวเตอร์สามารถกระทำการได้หรือที่เรียกว่า *executable file* และ *ไฟล์ข้อมูลไบนารี (binary data file)*.

ในหนังสือนี้ขอแบ่งประเภทไฟล์แต่ละประเภทดังนี้.

1. ไฟล์ธรรมดา
2. ไฟล์ดีไวซ์
3. ไดเรกทอรี
4. FIFO
5. UNIX domain socket

3.3 ไฟล์ธรรมดา

ไฟล์ธรรมดาคือไฟล์ที่มีข้อมูลจริงเก็บอยู่ในฮาร์ดดิสก์. ในที่นี้จะไม่แยกความแตกต่างของข้อมูลที่เก็บอยู่ในไฟล์ว่าเป็นเท็กซ์หรือไบนารี.

3.3.1 ไฟล์จุด

ไฟล์จุด (dot file) เป็นไฟล์ธรรมดาแต่มีความหมายพิเศษสำหรับคำสั่ง `ls` โดยที่คำสั่ง `ls` จะไม่แสดงไฟล์เหล่านี้ถ้าไม่ใช้ตัวเลือก `-a` หรือ `-A`. ไฟล์เหล่านี้มักจะเป็นไฟล์เท็กซ์หรือไดเรกทอรีใช้สำหรับเป็นไฟล์ตั้งค่าเริ่มต้นของโปรแกรมต่าง ๆ. ถ้าเป็นไดเรกทอรีก็จะเป็นที่สำหรับเก็บไฟล์ตั้งค่าเริ่มต้นหลาย ๆไฟล์ไว้ด้วยกัน. ตัวอย่างไฟล์จุดที่แนะนำไปแล้วได้แก่ `.bashrc` เป็นต้น.

3.3.2 ไฟล์สคริปต์

ไฟล์สคริปต์ (script) คือไฟล์ข้อมูลเท็กซ์ที่มีเนื้อหาเป็นคำสั่งของภาษาคอมพิวเตอร์แบบ*อินเทอร์พรีเตอร์ (interpreter)*บรรทัดแรกของไฟล์สคริปต์จะขึ้นต้นด้วยเครื่องหมาย “#!” และตามด้วยชื่อโปรแกรมแปลภาษาแบบ full path. เช่นไฟล์สคริปต์ที่ใช้เชลล์เป็นตัวแปลภาษาที่เรียกกันว่าเชลล์สคริปต์จะมีบรรทัดแรกเป็น

```
#!/bin/sh
```

และตั้งแต่บรรทัดที่สองเป็นต้นไปจะเป็นคำสั่งที่ใช้ในเชลล์. ไฟล์สคริปต์นี้ต้องตั้งค่าไฟล์ให้กระทำการได้ด้วยคำสั่ง “chmod +x” ก่อนจึงจะใช้งานได้เหมือนไฟล์โปรแกรมไบนารีทั่วไป.

interpreter ▶
โปรแกรมแปลคำสั่ง, ตัวแปรคำสั่ง. ภาษาคอมพิวเตอร์แบบหนึ่งซึ่งจะรับข้อมูลเท็กซ์, แปลความหมาย, แล้วกระทำการ. ภาษาคอมพิวเตอร์แบบนี้มีข้อดีที่พัฒนาได้รวดเร็ว, ไม่ต้องคอมไพล์ และมักจะมีวิธีการจัดการหน่วยความจำให้โดยอัตโนมัติ. ข้อเสียของภาษาคอมพิวเตอร์แบบนี้คือจะทำงานช้ากว่าโปรแกรมที่สร้างด้วยคอมไพเลอร์. โปรแกรมแปลภาษาที่นิยมใช้กันได้แก่ เชลล์, Perl, Python, Ruby ฯลฯ.

เกี่ยวกับคำสั่ง chmod ให้ดูที่หน้า 119

สำหรับการสร้างไฟล์สคริปต์ตัวแปลภาษาอื่น, ให้เปลี่ยน /bin/sh เป็นโปรแกรมแปลภาษาที่ต้องการเช่นถ้าเป็น perl script บรรทัดแรกของไฟล์จะเป็น “#!/usr/bin/perl”.

จะสังเกตเห็นว่าเราต้องเขียนชื่อโปรแกรมแปลภาษาด้วย full path ซึ่งบางครั้งโปรแกรมแปลภาษาอาจจะอยู่ในไดเรกทอรีที่ไม่มาตรฐานเช่น /usr/local/bin ดังนั้นการเขียนโปรแกรมแปลภาษาเช่น /usr/bin/perl อาจจะทำให้สคริปต์นั้นไม่ทำงานถ้า perl บังเอิญอยู่ในไดเรกทอรี /usr/local/bin. วิธีการแก้ปัญหานี้ทำได้โดยการใช้คำสั่ง env ช่วยโดยเขียนบรรทัดแรกของสคริปต์เป็น

env อ้างอิงหน้า 410

```
#!/bin/env perl
```

ในกรณีนี้ env จะใช้โปรแกรม perl ที่หาเจอตัวแรกจากตัวแปรสภาพแวดล้อม PATH ทำให้ไม้ต้องกังวลว่าตัวแปลภาษา perl จะอยู่ที่ /usr/bin, /usr/local/bin หรือที่อื่น ๆ. ถ้าในสภาพแวดล้อมของผู้ใช้นั้นใช้ perl ได้, สคริปต์นี้ก็จะทำงานได้.

3.4 ไฟล์ดีไวซ์

ไฟล์ดีไวซ์ (device file) เป็นไฟล์พิเศษแทนฮาร์ดแวร์ในคอมพิวเตอร์ต่าง ๆเช่น ฮาร์ดดิสก์, แป้นพิมพ์, เมาส์ ฯลฯ. ไฟล์เหล่านี้จะอยู่ใต้ไดเรกทอรี /dev. จากความหมายของไฟล์ที่ได้กล่าวไปแล้วว่าไฟล์คือสายของข้อมูล, ดังนั้นฮาร์ดแวร์ต่าง ๆเช่นฮาร์ดดิกส์ก็เป็นแหล่งข้อมูลอย่างหนึ่งและเคอร์เนลก็ถือให้ฮาร์ดแวร์เหล่านี้เป็นเสมือนไฟล์.

ตัวอย่างที่ 3.8: ตัวอย่างไฟล์ดีไวซ์

```
$ ls -l /dev/input/mouse0↵
crw-r--r--    1 root     root      13,  32 Jan  1  1970 /dev/input/mouse0
$ ls -l /dev/hda*↵
lr-xr-xr-x    1 root     root           32 Sep  5  2004 /dev/hda ->
ide/host0/bus0/target0/lun0/disc
lr-xr-xr-x    1 root     root           33 Sep  5  2004 /dev/hda1 ->
ide/host0/bus0/target0/lun0/part1
lr-xr-xr-x    1 root     root           33 Sep  5  2004 /dev/hda2 ->
```

```
ide/host0/bus0/target0/lun0/part2
lr-xr-xr-x    1 root     root           33 Sep  5  2004 /dev/hda3 ->
ide/host0/bus0/target0/lun0/part3
$ cd /dev/ide/host0/bus0/target0/lun0↵
$ ls -l↵
total 0
brw-------    1 root     root       3,   0 Jan  1  1970 disc
brw-------    1 root     root       3,   1 Jan  1  1970 part1
brw-------    1 root     root       3,   2 Jan  1  1970 part2
brw-------    1 root     root       3,   3 Jan  1  1970 part3
```

ถ้าใช้คำสั่ง `ls` ดูรายละเอียดของไฟล์ดีไวซ์จะเห็นว่าไฟล์ดีไวซ์บางไฟล์เช่น `/dev/hda` เป็นซอฟต์ลิงค์ ไปยังไฟล์ดีไวซ์จริง. ส่วนรายละเอียดของไฟล์ดีไวซ์จะมีอยู่สองกรณีคือ `crw-r--r--` และ `brw-------`. ตัวอักษร `c` และ `b` บ่งบอกว่าไฟล์ดังกล่าวเป็นไฟล์ดีไวซ์.

เรื่องของซอฟต์ลิงค์อยู่ที่หน้า 114.

ดูคำอธิบายเกี่ยวกับรายละเอียดของไฟล์ได้ที่หน้า 115.

ไฟล์ดีไวซ์ยังแบ่งเป็น 2 ประเภทใหญ่ ๆได้แก่*ดีไวซ์ character (character device)* และ *ดีไวซ์ block (block device)*. ดีไวซ์ character เป็นดีไวซ์ที่คอมพิวเตอร์ต้องรับข้อมูลมากจากดีไวซ์นั้น ๆเป็นลำดับไบต์ต่อไบต์, ไม่สามารถข้ามหรือกระโดดไปอ่านข้อมูลในตำแหน่งที่ต้องการได้. ส่วนดีไวซ์ block เป็นดีไวซ์ที่ส่งรับข้อมูลเป็นบล็อก. การส่งรับข้อมูลเป็นบล็อกจะมีการใช้ *buffer* ซึ่งเป็นพื้นที่หน่วยความจำที่กำหนดไว้คอรับเก็บข้อมูลเป็นบล็อกก่อนที่จะส่งไปดีไวซ์จริง ๆ. ตัวอย่างของดีไวซ์ block ได้แก่ฮาร์ดดิสก์, ส่วนตัวอย่างของดีไวซ์ character ได้แก่ เมาส์, เทป, แป้นพิมพ์ ฯลฯ.

จากตัวอย่างที่ 3.8 ให้สังเกตผลลัพธ์ของคำสั่ง `ls -l` ที่แสดงรายละเอียดของไฟล์ดีไวซ์เช่น `/dev/input/mouse0`. ตัวเลขที่อยู่หลังเจ้าของไฟล์และกลุ่ม (root) คือตัวเลขสองตัวได้แก่ 13 และ 32. ตัวเลขตัวแรกเรียกว่า *major device number*, และตัวเลขถัดมาเรียกว่า *minor device number*. major device number คือตัวเลขเฉพาะที่ใช้ในเคอร์เนลบ่งบอกไดร์เวอร์ (driver) ที่ใช้ควบคุมดีไวซ์นั้น. ตัวอย่างเช่นหมายเลข 1 เป็นไดร์เวอร์ที่ใช้เกี่ยวกับหน่วยความจำ. ชื่อไดร์เวอร์และหมายเลข major device นี้ดูได้จากไฟล์ `/proc/devices` หรือจากไฟล์ `/usr/src/linux/Documentation/devices.txt` ซึ่งเป็นเอกสารที่อยู่ในรหัสต้นฉบับของลินุกซ์เคอร์เนล. ดีไวซ์หลายตัวอาจจะใช้ไดร์เวอร์ตัวเดียวกัน, กล่าวคือมี major device number เป็นเลขตัวเดียวกัน. ตัวอย่างเช่นฮาร์ดดิสก์จะมี major device number ตัวเดียวกัน, แต่จะมี minor device number ต่างกันเพื่อแยกแยะว่าเป็นดีไวซ์.

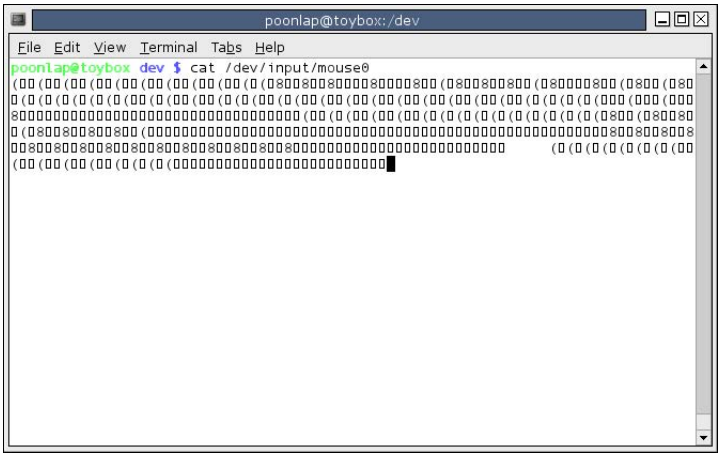
ไฟล์ดีไวซ์ก็มีคุณสมบัติเหมือนกันไฟล์ทั่วไปคือสามารถอ่านหรือเขียนลงไฟล์ได้. ตัวอย่างเช่นถ้าลองอ่านไฟล์ `/dev/input/mouse0` ด้วยคำสั่ง `cat` แล้วลากเมาส์ไปมาก็จะเห็นข้อมูลจากเมาส์ตามรูปที่ 3.4.

การเขียนข้อมูลลงไฟล์ก็ทำได้เหมือนกับไฟล์ทั่วไปเช่นกัน. ตัวอย่างเช่นเราสามารถเอาข้อมูลจากไฟล์เสียงฟอร์แมต au ใส่ใน `/dev/audio` ได้ตามตัวอย่างต่อไปนี้

au เป็นฟอร์แมตของไฟล์เสียงแบบง่าย ๆที่สร้างโดย Sun Microsystem แต่เป็นที่นิยมใช้แพร่หลายเท่าที่ควร. มักใช้ส่วนขยายชื่อไฟล์เป็น `.au`.

ตัวอย่างที่ 3.9: การรีไดเรกของไฟล์เสียงไปหา `/dev/audio`.

```
$ cat sound.au > /dev/audio↵
```



รูปที่ 3.4: ข้อมูลที่อ่านจากเมาส์โดยใช้ cat

การรีไดเรกไปหาไฟล์ดีไวซ์ไม่ใช่ทำได้ทุกกรณีโดยจะขึ้นอยู่กับไดรเวอร์ของดีไวซ์นั้น ๆ. เช่นถ้าเป็นการรีไดเรกไฟล์เสียง wave ให้รีไดเรกไปที่ /dev/dsp.

wave มักมีส่วนขยายชื่อไฟล์เป็น .wav

ตารางที่ 3.2 แสดงไฟล์ดีไวซ์ทั่วไปที่ใช้ในลินุกซ์. ไฟล์ดีไวซ์บางไฟล์อาจจะไม่ใช่ไฟล์จริง ๆแต่เป็นลิงก์ไปที่ไฟล์ดีไวซ์อื่น.

dsp ย่อมาจาก Digital Signal Processor.

ตารางที่ 3.2: ไฟล์ดีไวซ์ทั่วไปที่ใช้ในลินุกซ์

ไฟล์ดีไวซ์	คำอธิบาย
/dev/audio	ไฟล์ดีไวซ์ที่เกี่ยวกับเสียง. ไฟล์ที่เกี่ยวข้องอื่น ๆได้แก่ /dev/dsp, /dev/mixer เป็นต้น.
/dev/cdrom	ไฟล์ดีไวซ์ของ CD โดยปริยาย. โดยปรกติจะเป็นลิงก์ไปหาดีไวซ์ตัวจริงที่อื่น.
/dev/fd0	ไฟล์ดีไวซ์ของฟล็อปปี้ดิสก์.
/dev/hda	ไฟล์ดีไวซ์ของฮาร์ดดิสก์ตัวแรกที่ใช้อินเทอร์เฟสแบบ IDE.
/dev/sda	ไฟล์ดีไวซ์ของฮาร์ดดิสก์ตัวแรกที่มีอินเทอร์เฟสแบบ SCSI.

3.4.1 ไฟล์ /dev/null และ /dev/zero

ไฟล์ดีไวซ์ /dev/null และ /dev/zero เป็นไฟล์ดีไวซ์ที่ไม่ใช่ฮาร์ดแวร์ (ซอฟต์แวร์). ไฟล์ทั้งสองนี้น่าสนใจและมักใช้กับคำสั่งในเชลล์. /dev/null เป็นดีไวซ์ที่ไม่มีอะไร, ไม่ทำอะไร. ถ้าอ่านหรือเขียนข้อมูลลงในไฟล์นี้ก็ไม่มีอะไรเกิดขึ้น. ดังนั้นจึงมักใช้ /dev/null เป็นที่รองรับไม่ต้องการรีไดเรกสิ่งที่ไม่ต้องการแสดงไปหา /dev/null ตามที่แสดงในตัวอย่างที่ 2.43.

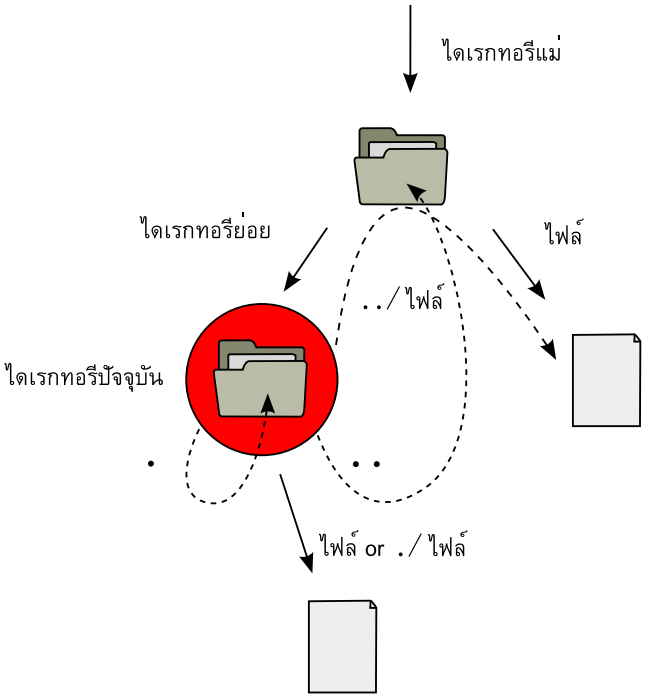
ไฟล์ดีไวซ์ /dev/zero ทำหน้าที่ตรงข้ามกับ /dev/null คือเป็นดีไวซ์ที่ให้ข้อมูลที่มีค่าเป็น NULL (อักขระ ASCII ตัวแรก) ให้กับโปรแกรมที่ต้องการไปเรื่อย ๆ. มักจะใช้กับคำสั่ง dd สร้างไฟล์ที่มีขนาดที่ต้องการ. ตัวอย่างต่อไปนี้เป็นการสร้างไฟล์ (ชื่อไฟล์ file) ที่มีขนาด 1 กิโลไบต์พอดี ๆด้วยคำสั่ง dd. ไฟล์ที่สร้างนี้สามารถเอาไปใช้เป็น swap ของระบบปฏิบัติการได้.

dd อ้างอิงหน้า 385

ตัวอย่างที่ 3.10: สร้างไฟล์ที่มีขนาดตามต้องการด้วย dd

```
$ dd if=/dev/zero of=file bs=1k count=1↵
1+0 records in
1+0 records out
$ ls -l file↵
-rw-r--r--    1 somchai  users        1024 Jun  7 00:32 file
```

3.5 ไดเรกทอรี



รูปที่ 3.5: ไฟล์, ไดเรกทอรี และความสัมพันธ์ต่าง ๆ

ไดเรกทอรี (directory) เป็นที่ที่เก็บรวมรวมไฟล์ต่าง ๆให้เป็นระบบระเบียบ. ในไดเรกทอรีหนึ่งสามารถเก็บไดเรกทอรีย่อย ๆออกไปได้อีกเรียกว่า sub-directory หรือเก็บไฟล์ต่าง ๆในไดเรกทอรีนั้น. ไดเรกทอรีที่เก็บไดเรกทอรีย่อยไว้เรียกว่า*ไดเรกทอรีแม่ (parent directory)* ของไดเรกทอรีย่อยนั้น. จากไดเรกทอรีย่อย, ถ้าต้องการอ้างอิงถึงไดเรกทอรีแม่จะใช้เครื่องหมายจุด (..) แทนไดเรกทอรีแม่. ความจริงแล้วไดเรกทอรีก็คือไฟล์แต่เป็นไฟล์พิเศษชนิดหนึ่งที่เก็บที่ชื่อของไฟล์และที่อยู่ของไฟล์ไว้ด้วยกันโดยมี ls เป็น

ในระบบปฏิบัติการ Windows จะเรียกว่า directory ว่า folder.

โปรแกรมที่อ่านเนื้อหาที่อยู่ในไดเรกทอรีแล้วแสดงผลทางหน้าจอ.

3.5.1 โฮมไดเรกทอรี

โฮมไดเรกทอรี (home directory) เป็นไดเรกทอรีพิเศษและเป็นที่เริ่มต้นของเชลล์. ผู้ใช้ทุกคนในระบบจะมีโฮมไดเรกทอรีเฉพาะแต่ละคนซึ่งโดยปรกติโฮมไดเรกทอรีมักจะอยู่ใต้ไดเรกทอรี `/home` และชื่อของโฮมไดเรกทอรีมักจะเป็นชื่อล็อกอินของผู้ใช้.

ถ้าผู้ใช้สั่งคำสั่ง `cd` โดยไม่มีตัวเลือก, จะเป็นการเปลี่ยนไดเรกทอรีไปที่โฮมไดเรกทอรีโดยปริยาย. การอ้างอิงตำแหน่งของโฮมไดเรกทอรีอาจทำได้โดยใช้เครื่องหมาย tilde (`~`). ตัวอย่างเช่น `~/.bashrc` เป็นการอ้างอิงถึงไฟล์ `.bashrc` ที่อยู่ในโฮมไดเรกทอรี. วิธีอื่นที่ใช้อ้างอิงไฟล์ที่อยู่ในโฮมไดเรกทอรีได้คือการใช้ตัวแปรสภาพแวดล้อม `HOME` เช่น `$HOME/.bashrc` เป็นการอ้างอิงไฟล์ให้ผลเหมือนกับ `~/.bashrc`.

3.5.2 ไดเรกทอรีปัจจุบัน

ไดเรกทอรีที่เชลล์ทำงานอยู่เรียกเป็นภาษาอังกฤษว่า working direcotory หรือ*ไดเรกทอรีปัจจุบัน (current working directory)*. ผู้ใช้จะใช้คำสั่ง `cd` เพื่อเปลี่ยนไดเรกทอรีปัจจุบันไปที่ไดเรกทอรีอื่น ๆและสั่งคำสั่งในไดเรกทอรีนั้น. คำสั่งที่ทำงานอยู่จะมีความสัมพันธ์กับไดเรกทอรีปัจจุบันโดยที่ไดเรกทอรีที่สั่งคำสั่งนั้นโดยจะเป็นสถานที่อ้างอิงไฟล์ของคำสั่ง.

ตัวอย่างที่ 3.11: ไดเรกทอรีที่ทำงานอยู่กับคำสั่ง

```
$ pwd↵
/home/somchai
$ ls file.txt↵
file.txt
$ cd ..↵
$ ls /home/somchai/file.txt↵
/home/somchai/file.txt
$ ls somchai/file.txt↵
somchai/file.txt
```

จากตัวอย่างจะเห็นได้ว่าถ้าการระบุชื่อไฟล์ให้เป็นอาร์กิวเมนต์ให้คำสั่งเช่น `file.txt` เป็นการระบุแบบนัย ๆว่าไฟล์ `file.txt` อยู่ในไดเรกทอรีปัจจุบัน. หมายความว่าคำสั่ง `ls` รับรู้ว่าไดเรกทอรีที่ทำงานอยู่นี้คือ `/home/somchai` แล้วไฟล์ `file.txt` อยู่ในไดเรกทอรีนี้. หลังจากที่เปลี่ยนไดเรกทอรีไปที่ไดเรกทอรีแม่ (`..`, `/home`) แล้วจะอ้างอิงถึงไฟล์เดิมก็ต้องเขียนชื่อไฟล์เต็ม `/home/somchai/file.txt` ที่เรียกว่าการเขียนชื่อไฟล์แบบ full path. หรืออ้างอิงไฟล์เป็น `somchai/file.txt` เรียกว่าการเขียนชื่อไฟล์แบบ relative path. relative หมายถึงการอ้างอิงจากไดเรกทอรีปัจจุบันนั่นเอง. ถ้าจะอ้างอิงไดเรกทอรีปัจจุบันให้ชัดเจนก็อาจจะเขียน `./somchai/file.txt` ก็ได้. เครื่องหมายจุด (`.`) เป็นเครื่องหมายแทนไดเรกทอรีปัจจุบัน.

บ่อยครั้งที่ผู้ใช้เชลล์ต้องเปลี่ยนไดเรกทอรีไปมาและต้องการเปลี่ยนไดเรกทอรีกลับไปไดเรกทอรีก่อนหน้าไดเรกทอรีปัจจุบัน. สมมติว่าไดเรกทอรีก่อนหน้าไดเรกทอรี

ปัจจุบันเป็นไดเรกทอรีย่อย ๆหลายชั้น, การที่จะเปลี่ยนไดเรกทอรีกลับไปนั้นผู้ใช้อาจจะใช้การเติมเต็มชื่อไฟล์ช่วย. เทคนิคหนึ่งที่นิยมใช้กันคือใช้เครื่องหมาย "-" เป็นอาร์กิวเมนต์ของคำสั่ง cd จะเป็นการเปลี่ยนไดเรกทอรีไปไดเรกทอรีล่าสุดที่เคยอยู่. ทำให้ประหยัดเวลาในการเปลี่ยนไดเรกทอรีไปหาไดเรกทอรีก่อนหน้าปัจจุบัน.

ตัวอย่างที่ 3.12: การเปลี่ยนไดเรกทอรีไปหาไดเรกทอรีก่อนหน้าปัจจุบัน.

```
$ pwd↵
/usr/src/linux/Documentation/filesystems
$ cd /tmp↵
$ cd -↵                                    ← ไม่ต้องเขียนชื่อไดเรกทอรีก่อนหน้านี้ให้เสียเวลา
$ pwd↵
/usr/src/linux/Documentation/filesystems
```

คำสั่งเฉพาะที่ใช้จำไดเรกทอรีที่เคยอยู่ได้แก่คำสั่ง pushd และคำสั่งที่เปลี่ยนไดเรกทอรีไปไดเรกทอรีจำไว้ได้แก่ popd แต่จากประสบการณ์ของผู้เขียนแล้วไม่ค่อยได้ใช้คำสั่งเหล่านี้มากเท่าไหร่นัก.

3.5.3 โครงสร้างไฟล์และไดเรกทอรี

ในระบบปฏิบัติการยูนิกซ์และลินุกซ์จะมีโครงสร้างไดเรกทอรีแสดงด้วยแผนภาพต้นไม้ได้ในรูปที่ 3.2. ไดเรกทอรีเริ่มต้นของไดเรกทอรีอื่น ๆและไฟล์ทั้งหมดได้แก่ *ไดเรกทอรีรูท (root directory)* ซึ่งใช้เครื่องหมาย slash (/) แทนไดเรกทอรีนี้. ใต้ไดเรกทอรีรูทประกอบด้วยไดเรกทอรีย่อย ๆต่าง ๆซึ่งแต่ละดิสทริบิวชันอาจจะมีไดเรกทอรีย่อยไม่เหมือนกัน. ในบางครั้งไฟล์ที่ที่ทำหน้าที่เหมือนกันอาจจะอยู่ในไดเรกทอรีที่ต่าง ๆกันขึ้นอยู่กับดิสทริบิวชันที่ใช้. ด้วยเหตุนี้เองจึงมีความพยายามสร้างมาตรฐานโครงสร้างระบบไฟล์ไดเรกทอรีที่เรียกว่า *Filesystem Hierarchy Standard (FHS)* [30]. FHS เป็นเอกสารอ้างอิงมีประโยชน์สำหรับผู้ใช้ทั่วไป, ผู้พัฒนาซอฟต์แวร์, หรือตัวโปรแกรมเองสามารถคาดเดาได้ว่าไฟล์หรือไดเรกทอรีที่ต้องการใช้อยู่ที่ไหน.

ไดเรกทอรีรูท, /

FHS ได้กำหนดไดเรกทอรีต่าง ๆใต้ไดเรกทอรีรูทไว้ [30], แต่ดิสทริบิวชันทั่วไปก็ไม่ได้มีไดเรกทอรีทั้งหมดที่ FHS กำหนดไว้. ไดเรกทอรีทอที่อยู่ใต้ไดเรกทอรีรูทที่สำคัญ ๆ ได้แก่.

- bin
  เป็นไดเรกทอรีที่เก็บโปรแกรมหลักที่จำเป็นต่าง ๆเช่น bash, ls, pwd เป็นต้น. ชื่อ bin ย่อมาจาก binaries หมายถึงเป็นไดเรกทอรีที่เก็บไฟล์ไบนารีซึ่งหมายถึงโปรแกรมนั่นเอง.

- boot
  ไดเรกทอรีที่เก็บไฟล์ที่เกี่ยวข้องกับ boot loader เคอร์เนล ฯลฯ.

boot loader ▶

- `dev`
  ชื่อ `dev` ย่อมากจาก devices เป็นไดเรกทอรีที่เก็บไฟล์ดีไวซ์ต่าง ๆ.

- `etc`
  เป็นไดเรกทอรีสำคัญที่เก็บไฟล์หรือไดเรกทอรีย่อยที่เกี่ยวกับการปรับแต่งหรือติดตั้งระบบ. ชื่อ `etc` มาจากคำว่า etcetera.

- `home`
  เป็นไดเรกทอรีที่มีหรือไม่มีก็ได้. เป็นที่ของโฮมไดเรกทอรีของผู้ใช้แต่ละคน.

- `lib`
  ไดเรกทอรีเก็บไลบรารี (library) หลักที่จำเป็นและโมดูลของเคอร์เนล.

- `mnt`
  ไดเรกทอรีสำหรับ mount ไฟล์ซิสเต็มต่าง ๆชั่วคราว. ตัวอย่างเช่นเมื่อต้องการใช้ซีดีรอมก็จะ mount ไว้ที่ `/mnt/cdrom` แล้วไฟล์ต่าง ๆที่อยู่ในซีดีรอมก็จะอยู่ใต้ไดเรกทอรี `/mnt/cdrom`.

mount ▶
การนำดีไวซ์หน่วยความจำข้อมูลเช่นฮาร์ดดิสก์มาปะติดในระบบโครงสร้างไดเรกทอรี. เช่นการนำซีดีรอมมาปะติด (mount) ไว้กับไดเรกทอรี `/mnt/cdrom` เป็นต้น. การ mount นี้สามารถระบุไฟล์ซิสเต็มของดีไวซ์ได้ด้วยเมื่อ mount ไฟล์ซิสเต็มเข้าสู่โครงสร้างไดเรกทอรีแล้วก็จะใช้งานได้อย่างโปร่งใส, อาจจะไม่สังเกตุเห็นความแตกต่างระหว่างไฟล์ซิสเต็ม.

- `opt`
  ไดเรกทอรีเก็บแพกเกจโปรแกรมเพิ่มเติมอื่น ๆที่ไม่ใช่โปรแกรมมาตรฐานเช่นแพกเกจโปรแกรมที่ไม่ใช่ซอฟต์แวร์เสรีเป็นต้น.

- `root`
  เป็นไดเรกทอรีที่ไม่บังคับ. ใช้เป็นโฮมไดเรกทอรีของ root.

ในสมัยก่อน, โฮมไดเรกทอรีของ root ในระบบปฏิบัติการยูนิกซ์คือ `/` ซึ่งหมายถึงโครงสร้างไฟล์ไดเรกทอรีทั้งหมด. การที่โฮมไดเรกทอรีของ root เป็น `/` อาจเป็นอันตรายได้ถ้าเผลอลบไฟล์ที่จำเป็นต่อระบบโดยไม่ตั้งใจ.

- `sbin`
  ไดเรกทอรีที่เก็บโปรแกรมใช้งานเกี่ยวกับระบบ (system) เช่น `shutdown`, `reboot` ฯลฯ. โปรแกรมคำสั่งเหล่านี้มักใช้โดยผู้ดูแลระบบ (root). ผู้ใช้ทั่วไปอาจไม่มีความจำเป็นที่ต้องรวมไดเรกทอรีนี้ในตัวแปรสภาพแวดล้อม `PATH`.

- `tmp`
  ไดเรกทอรีที่เก็บไฟล์ชั่วคราว. ไฟล์ชั่วคราวเหล่านี้ไม่ได้สร้างโดยผู้ใช้โดยตรงแต่มักสร้างโดยโปรแกรมที่มีความจำเป็นต้องสร้างไฟล์ชั่วคราวก็จะสร้างไว้ที่นี่. ใครก็ได้สามารถสร้างไฟล์ในไดเรกทอรีนี้ได้แต่ไม่สามารถลบไฟล์ของคนอื่นที่สร้างได้ถ้าไม่มีสิทธิ์ในไฟล์นั้น. เนื่องจากเป็นไดเรกทอรีสำหรับไฟล์ชั่วคราวจึงมีการลบข้อมูลที่อยู่ใต้ไดเรกทอรีนี้บ้างตามความเหมาะสม.

- `usr`
  เป็นโครงสร้างระบบไฟล์ไดเรกทอรีลำดับที่สอง. กล่าวคือใต้ไดเรกทอรีนี้จะมีโครงสร้างคล้ายกับ `/` ที่เป็นโครงสร้างระบบไฟล์ไดเรกทอรีหลัก, มีไดเรกทอรี `bin`, `sbin` และ `lib` เหมือนกัน. นอกจากนี้ยังมีไดเรกทอรีย่อยที่สำคัญอื่น ๆได้แก่ `include`, `local`, `share` ฯลฯ.

- `var`
  ไดเรกทอรีเก็บข้อมูลที่แปรผัน (variable) ได้เช่นข้อมูลที่เกี่ยวกับบันทึกของระบบ (system log) เป็นต้น. ล็อกของระบบอยู่ในไดเรกทอรีนี้เพราะขนาดของไฟล์ที่บันทึกนั้นแปรผันไปตามวัน, มากบ้าง, น้อยบ้าง. ไฟล์ที่มีลักษณ์เหมือนกับล็อกของระบบก็ควรจะอยู่ใต้ไดเรกทอรีนี้.

ไดเรกทอรี `/usr`

ไดเรกทอรี `/usr` เป็นไดเรกทอรีสำคัญรองมาจากไดเรกทอรีรูท. ใต้ไดเรกทอรีนี้เป็นที่ประกอบด้วยไดเรกทอรีย่อยอื่น ๆอีกมากมาย, ส่วนใหญ่เป็นข้อมูลหรือโปรแกรมที่ใช้ร่วมกันในระบบ. ไดเรกทอรีย่อยในไดเรกทอรีนี้มักจะเป็นข้อมูลอ่านได้อย่างเดียว, ไม่ใช่ที่ที่เก็บข้อมูลที่มีการเปลี่ยนแปลงบ่อย ๆ.

ไดเรกทอรีย่อยที่อยู่ใน `/usr` สำคัญ ๆได้แก่.

- `bin`
  เป็นไดเรกทอรีเก็บไฟล์โปรแกรม (binary) เกือบทั้งหมดที่ใช้ได้ในระบบ. จะแตกต่างจาก `/bin` ที่ไดเรกทอรี `/bin` เป็นที่เก็บไฟล์โปรแกรมหลัก (จำเป็นต่อระบบ), ส่วน `/usr/bin` เป็นที่เก็บไฟล์โปรแกรมที่อำนวยความสะดวกแก่ผู้ใช้แต่อาจจะไม่มีความจำเป็นต่อระบบก็ได้. ตัวอย่างไฟล์โปรแกรมที่เก็บไว้ในไดเรกทอรีนี้เช่น `bc`, `gcc`, `man`, `perl` เป็นต้น.

- `include`
  ไดเรกทอรีเก็บ *ไฟล์ header (header file)* ที่จำเป็นในการพัฒนาซอฟต์แวร์, คอมไพล์โปรแกรมต่าง ๆ.

header file ▶

- `lib`
  ไดเรกทอรีเก็บไลบรารีที่ใช้ในระบบ. ใต้ไดเรกทอรีนี้อาจจะมีไดเรกทอรีย่อยแยกเฉพาะสำหรับแต่ไลบรารีด้วย.

- `local`
  เป็นโครงสร้างระบบไฟล์ไดเรกทอรีแยกย่อยต่อไปอีก. ใช้สำหรับผู้ดูและระบบติดตั้งโปรแกรมหรือแพกเกจโปรแกรมที่อาจจะไปซ้ำซ้อนกับ `/usr` หรือใช้ติดตั้งแพกเกจโปรแกรมที่ไม่ต้องการไว้ใต้ `/usr`. ใต้ไดเรกทอรีนี้จะมีไดเรกทอรีย่อยเหมือน `/usr`.

✍ แพกเกจโปรแกรมหมายถึงโปรแกรมที่ไม่ใช่ไฟล์เดี่ยว ๆ. เป็นชุดโปรแกรมที่มีข้อมูล, คู่มือ, ไลบรารี มาด้วยกัน.

- `sbin`
  ไดเรกทอรีเก็บไฟล์โปรแกรมเกี่ยวกับการดูแลระบบ. ไฟล์โปรแกรมนี้อาจจะไม่มีความจำเป็นแต่สะดวกในการใช้งาน.

- `share`
  ไดเรกทอรีเก็บไฟล์ข้อมูลที่ไม่ขึ้นกับสถาปัตยกรรมคอมพิวเตอร์.

ก X Window
n 11 Release 6.

- `X11R6`
  ไดเรกทอรีเก็บไฟล์, โปรแกรม, ข้อมูลที่เกี่ยวกับ X วินโดว์.

- `src` ไดเรกทอรีที่เก็บรหัสต้นฉบับของโปรแกรมต่าง ๆเช่นรหัสต้นฉบับของเคอร์เนล.

ไดเรกทอรี `/usr/share`

ในไดเรกทอรี `/usr/share` จะมีข้อไฟล์ข้อมูลต่าง ๆที่น่าสนใจ. ไฟล์ที่อยู่ใต้ไดเรกทอรีนี้จะมักจะเป็นข้อมูลหรือเอกสารที่สามารถใช้ได้ด้วยกันไม่ขึ้นกับว่าต้องเป็นคอมพิวเตอร์สถาปัตยกรรมใด.

- `doc`
  ไดเรกทอรีเก็บเอกสารของแต่ละโปรแกรม. ไดเรกทอรีนี้จะมีไดเรกทอรีย่อยแบ่งเป็นชื่อโปรแกรมหรือชื่อแพกเกจต่าง ๆ. ในไดเรกทอรีย่อยเหล่านี้จะมีเอกสารที่เกี่ยวข้องกับโปรแกรมหรือแพกเกจนั้น ๆ.

- `man`
  ไดเรกทอรีเก็บไฟล์ข้อมูลของ on-line manual. โดยปรกติการใช้คำสั่ง `man` ดูคู่มือการใช้งานจะเปิดอ่านข้อมูลจากไฟล์ที่อยู่ใต้ไดเรกทอรีนี้.

- `info`
  เป็นไดเรกทอรีเก็บไฟล์ข้อมูลเอกสารการใช้งานที่อยู่รูปของ GNU info.

ไดเรกทอรี `/var`

ไดเรกทอรี `var` เป็นที่เก็บข้อมูลที่มีขนาดแปรผันไปเรื่อย ๆ. ไดเรกทอรีย่อยที่สำคัญ ๆได้แก่.

- `cache`
  ไฟล์ที่อยู่ใต้ไดเรกทอรีนี้จะเป็นข้อมูลที่ใช้บ่อย ๆและสร้างขึ้นโดยโปรแกรมที่ใช้ข้อมูลนั้น. โปรแกรมบางอย่างอาจจะสร้างข้อมูลที่ใช้บ่อย ๆไว้ในไดเรกทอรีนี้. เมื่อต้องการใช้ข้อมูลเหล่านี้จึงไม่ต้องสร้างข้อมูลใหม่ทำให้ทำงานเร็วกว่าสร้างข้อมูลทุกครั้ง.

- `lib`
  ไดเรกทอรีที่เก็บข้อมูลที่เกี่ยวกับสถานะของโปรแกรมที่ทำงานอยู่ในระบบ. ข้อมูลเหล่านี้จะไม่สูญหายหลังจากรีบูทระบบ.

- `log`
  ใต้ไดเรกทอรีนี้จะเป็นที่เก็บ*ล็อก (log)* ของโปรแกรมต่าง ๆ. โปรแกรมที่ว่านี้มักจะเป็นโปรแกรมเซิฟร์เวอร์หรือล็อกของระบบ. ล็อกเหล่านี้มีประโยชน์ใช้ดูแลความเป็นไปของระบบปฏิบัติการหรือเซิฟร์เวอร์ว่าทำงานปรกติดีหรือไม่. ถ้าเกิดมีข้อผิดพลาด (error) บางกรณีก็สามารถบอกได้จากล็อกว่ามีสาเหตุจากอะไร.

- `run`
  เป็นที่เก็บข้อมูลเกี่ยวกับโปรเซสได้แก่โปรเซส ID ของโปรแกรมต่าง ๆ. ตัวอย่างเช่นไฟล์ `syslog-ng.pid` เป็นไฟล์ที่เก็บโปรเซส ID ของ `syslog-ng`. ถ้าต้องการจะใช้คำสั่งที่ต้องการโปรเซส ID เช่น `kill` ก็ทำได้โดย `kill ‘cat /var/log/syslog-ng‘` คือใช้การแทนที่คำสั่งเรียกโปรเซส ID ของโปรแกรมที่ต้องการได้เลย. ไฟล์ที่เก็บโปรเซส ID แบบนี้ไม่ได้มีให้ทุกโปรแกรม. มักจะเป็นโปรแกรมเซิฟร์เวอร์หรือโปรแกรมที่เกี่ยวข้องกับระบบ.

- `spool`
  ไดเรกทอรีที่เก็บข้อมูลที่รอการประมวลผลและเมื่อประมวลผลเรียบร้อยแล้วก็จะลบทิ้ง. ข้อมูลพวกนี้เช่นข้อมูลที่รอประมวลผลพิมพ์ออกทางเครื่องพิมพ์เป็นต้น.

- `tmp`
  เป็นที่เก็บไฟล์ข้อมูลชั่วคราวของโปรแกรมต่าง ๆ. ไดเรกทอรีนี้ทำหน้าที่คล้ายกับไดเรอทอรี `/tmp` แต่ไดเรกทอรี `/tmp` จะมีความถี่ของการลบข้อมูลมากกว่าไดเรกทอรีนี้.

`syslog` เป็นโปรแกรมที่จัดการเกี่ยวกับล็อกของโปรแกรมต่าง ๆ.

3.5.4 ไดเรกทอรี `proc`

ไดเรกทอรี `proc` หรือเรียกอีกอย่างว่า*ระบบไฟล์ proc (proc filesystem)* [31] เป็นอินเทอร์เฟสสำหรับติดต่อกับเคอร์เนลในรูปแบบของไดเรกทอรี. ไดเรกทอรี `proc` ไม่ใช่ไดเรกทอรีจริง, แต่เป็นไดเรกทอรีเสมือนที่สร้างอยู่ในหน่วยความจำใช้ในการแสดงข้อมูลต่าง ๆของระบบหรือใช้ตั้งค่าตัวแปรของเคอร์เนล.

ตัวอย่างที่ 3.13: ไดเรกทอรี `proc`

```
$ ls -F /proc↵
1/      17700/  5908/  7526/  7697/  7743/  7927/      driver/      mounts@
10/     17706/  6/     7527/  7698/  7745/  8/         execdomains  mtrr
11/     17707/  6558/  7545/  7699/  7747/  8228/      fb           net/
12/     17710/  6560/  7547/  7701/  7749/  8235/      filesystems  partitions
13/     17711/  671/   7548/  7702/  7750/  8246/      fs/          pci
14/     18888/  7/     7613/  7703/  7751/  9/         ide/         scsi/
149/    2/      7045/  7631/  7722/  7756/  acpi/      interrupts   self@
15/     21388/  7064/  7633/  7723/  7757/  asound/    iomem        slabinfo
16/     21389/  7141/  7636/  7724/  7782/  buddyinfo  ioports      speakup/
16801/  21454/  7201/  7638/  7726/  7783/  bus/       irq/         stat
17670/  3/      7202/  7640/  7727/  7784/  cmdline    kallsyms     swaps
17671/  333/    7361/  7642/  7728/  7791/  config.gz  kcore        sys/
17673/  386/    7449/  7665/  7730/  7805/  cpuinfo    kmsg         sysvipc/
17677/  4/      7453/  7680/  7732/  7811/  crypto     loadavg      tty/
17682/  495/    7522/  7691/  7734/  7823/  devices    locks        uptime
17683/  5/      7523/  7693/  7736/  7840/  diskstats  meminfo      version
17690/  5230/   7524/  7695/  7738/  7924/  dma        misc         vmnet/
17695/  5867/   7525/  7696/  7740/  7925/  dri/       modules      vmstat
```

ในไดเรกทอรี `proc` จะมีไฟล์และไดเรกทอรีย่อยต่าง ๆ. การดูข้อมูลต่าง ๆสามารถใช้คำสั่งเช่น `cat` หรือ `less` เปิดไฟล์ดูเนื้อหาที่ต้องการได้. ข้อมูลที่เกี่ยวกับระบบที่

สามารถดูได้เช่น ข้อมูลเกี่ยวกับโปรเซสที่ทำงานในระบบ, ข้อมูลเฉพาะของเคอร์เนล, ข้อมูลเกี่ยวกับหน่วยความจำ, ข้อมูลเกี่ยวกับหน่วยประมวลกลาง เป็นต้น. ข้อมูลที่อยู่ในไฟล์ใต้ไดเรกทอรี `proc` ส่วนใหญ่จะไม่มีการปรับแต่งให้อ่านง่ายเพราะจุดประสงค์หลักคือการเสนอข้อมูลโดยตรงจากเคอร์เนลในรูปแบบของไดเรกทอรี. ดังนั้นถ้าต้องการดูรายละเอียดของระบบ, ให้ใช้โปรแกรมเฉพาะสำหรับดูข้อมูลที่เกี่ยวกับระบบเช่น ถ้าต้องการดูการใช้งานของหน่วยความจำตอนนี้ก็ใช้คำสั่ง `free` แทนที่จะดูข้อมูลโดยตรงจากไฟล์ `/proc/meminfo`, หรือถ้าต้องการดูรายการและรายละเอียดเบื้องต้นของโปรเซสต่าง ๆ ก็ใช้คำสั่ง `ps` แทนที่จะดูไฟล์ที่เกี่ยวกับโปรเซสในไดเรกทอรี `proc` เป็นต้น. อย่างไรก็ตามเนื่องจากระบบไฟล์ proc เป็นการแสดงข้อมูลจากเคอร์เนลโดยตรง, ส่วนโปรแกรมแสดงข้อมูลของระบบเช่น `ps` เป็นโปรแกรมที่อาจถูกแก้ไขได้ง่ายในกรณีที่เครื่องถูกบุกรุกและใช้งานโดยไม่พึงประสงค์. ดังนั้นการรู้จักกับระบบไฟล์ proc จะช่วยให้เราสำรวจข้อมูลได้แม่นยำกว่าโปรแกรมแสดงข้อมูลของระบบ, และในบางกรณีจะสามารถดูข้อมูลที่โปรแกรมเหล่านั้นไม่สามารถแสดงได้ด้วย.

`free` อ้างอิงหน้า 400

ไฟล์และไดเรกทอรีย่อยที่น่าสนใจในไดเรกทอรี `proc` ได้แก่.

- ไดเรกทอรีที่เป็นตัวเลข
  ตัวเลขที่เป็นชื่อไดเรกทอรีคือโปรเซส ID. ในไดเรกทอรีย่อยเหล่านี้จะเก็บข้อมูลต่าง ๆที่เกี่ยวกับโปรเซสนั้นเช่น คำสั่งบรรทัดที่ใช้ (`cmdline`), ไดเรกทอรีที่โปรเซสนั้นทำงานอยู่ (`cwd`), สภาพแวดล้อมที่ทำงานอยู่ (`environ`), file descriptor ที่โปรเซสใช้ (`fd`) เป็นต้น.

- `cmdline`
  บรรทัดคำสั่งของเคอร์เนล. ในไฟล์นี้สามารถดูตัวเลือก, ค่าตัวแปรต่าง ๆที่ส่งให้เคอร์เนลตอนเริ่มทำงานได้.

- `cpuinfo`
  ข้อมูลต่าง ๆที่เกี่ยวกับหน่วยประมวลผล.

- `devices`
  แสดง major device number และดีไวซ์ character หรือ block ที่มีในระบบ.

- `filesystems`
  รายการระบบไฟล์ที่ใช้ได้ในระบบ.

- ไดเรกทอรี `driver`
  ข้อมูลเกี่ยวกับไดร์เวอร์ต่าง ๆ. ปัจจุบันยังเพียงเฉพาะไดร์เวอร์บางตัวเท่านั้นที่ใช้ไดเรกทอรีนี้.

- ไดเรกทอรี `fs`
  ข้อมูลเกี่ยวกับระบบไฟล์ต่าง ๆเช่น nfs, reiserfs, xfs ฯลฯ.

- ไดเรกทอรี `ide`
  ข้อมูลต่าง ๆที่เกี่ยวกับดีไวซ์แบบ ide เช่นฮาร์ดดิสก์, ซีดีรอม ฯลฯ.

- `interrupts`
  ข้อมูลเกี่ยวกับการใช้ interrupt.

- `iomem`
  ข้อมูลตำแหน่งการใช้หน่วยความจำ.

- `ioports`
  ข้อมูลพอร์ต I/O ต่าง.

- `meminfo`
  ข้อมูลทั่วไปของการใช้งานหน่วยความจำ. คำสั่ง `free` จะแสดงผลในรูปแบบที่เข้าใจง่ายกว่า.

- `modules`
  รายชื่อโมดูลที่โหลดอยู่ในเคอร์เนล (kernel module) ที่ใช้อยู่. คำสั่ง `lsmod` จะแสดงรายชื่อโมดูลต่าง ๆให้ดูง่ายกว่า.

`lsmod` อ้างอิงหน้า 400

- `mounts`
  รายชื่อระบบไฟล์ที่ mount อยู่ในระบบ. คำสั่งที่เกี่ยวข้องคือ `mount`.

- ไดเรกทอรี `net`
  ข้อมูลเกี่ยวกับเน็ตเวิร์กต่าง ๆ.

- `paritions`
  ข้อมูลพื้นฐานของพาร์ทิชันต่าง ๆ. ข้อมูลที่ได้จากโปรแกรม `fdisk` จะละเอียดกว่า.

- `pci`
  รายละเอียดของอุปกรณ์ที่ต่อกับบัส PCI ในระบบ. จะใช้คำสั่ง `lspci` เพื่อดูรายละเอียดต่าง ๆก็ได้.

`lspci` อ้างอิงหน้า 400

- ไดเรกทอรี `scsi`
  ข้อมูลต่าง ๆที่เกี่ยวกับดีไวซ์แบบ scsi เช่นฮาร์ดดิสก์, ซีดีรอม ฯลฯ.

- `uptime`
  เวลาทั้งหมดตั้งแต่เครื่องเริ่มทำงาน. ให้ใช้คำสั่ง `uptime` แทนที่จะดูไฟล์นี้.

`uptime` อ้างอิงหน้า 404

- `version`
  ข้อมูลรุ่นของเคอร์เนลที่ใช้. ให้ใช้คำสั่ง `uname` จะแสดงผลได้เข้าใจง่ายกว่า.

`uname` อ้างอิงหน้า 425

การตั้งค่าต่าง ๆให้กับเคอร์เนลทำได้โดยการแก้ไขไฟล์ที่อยู่ใต้ไดเรกทอรี `/proc/sys`. ใต้ไดเรกทอรี `/proc/sys` จะมีไดเรกทอรีย่อยแบ่งตามหมวดหมู่. การตั้งค่าที่ต้องการทำได้โดยการใช้ `echo` ค่าที่ต้องการตั้งแล้วรีไดเรกลงในไฟล์. เมื่อตั้งค่าใหม่เรียบร้อย

การตั้งค่าต่าง ๆของเคอร์เนลอีกวิธีคือการใช้คำสั่ง `sysctl`.

แล้วจะมีผลทันที, ดังนั้นควรระมัดถ้าไม่มั่นใจในสิ่งที่กำลังทำอยู่ควรจะอ่านเอกสารที่เกี่ยวข้องกับเคอร์เนลก่อนลงมือ.

ตัวอย่างเช่นถ้าไม่ต้องการตอบสนองการ ping จากเครื่องคอมพิวเตอร์ที่อยู่ในเน็ตเวิร์กก็สามารถปรับแต่งค่าในเคอร์เนลโดยใส่ตัวเลข 1 ในไฟล์ `/procs/sys/net/ipv4/icmp_echo_ignore_all`.

ตัวอย่างที่ 3.14: การปรับแต่งระบบด้วยไฟล์ในไดเรกทอรี `/proc`.

```
# cat /proc/sys/net/ipv4/icmp_echo_ignore_all↵
0
# echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all↵
# cat /proc/sys/net/ipv4/icmp_echo_ignore_all↵
1
```

3.6 FIFO

จากบทที่แล้วเราได้รู้จักไปป์ (หน้า 44) ซึ่งเป็นวิธีการรับข้อมูลที่ส่งออกทาง stdout ของโปรเซสหนึ่งให้กับ stdin ของอีกโปรเซสหนึ่งเพื่อเป็นการส่งข้อมูลระหว่างโปรเซส. ข้อมูลที่ส่งผ่านไปป์มีลำดับ, กล่าวคือข้อมูลที่ส่งไปก่อนก็จะถึงก่อน. การถ่ายโอนข้อมูลโดยตามลำดับเช่นนี้เรียกว่า *FIFO (First In First Out)*. ในระบบยูนิกซ์, คำว่า FIFO จะหมายถึง *named pipe* ไม่ได้หมายถึงไปป์. named pipe แปลเป็นภาษาไทยว่า "ไปป์ที่มีชื่อ" คือไฟล์ที่ทำหน้าที่เหมือนไปป์.

ในที่นี้จะเรียกว่า "ไฟล์ไปป์"

การสร้างไฟล์ไปป์จะใช้คำสั่ง `mkfifo` หรือ `mknod`. ต่อไปนี้เป็นตัวอย่างการสร้างไฟล์ไปป์ด้วยคำสั่ง `mkfifo` และไฟล์นั้นมีชื่อเป็น `in_out`.


คำสั่ง `mknod` เป็นคำสั่งเก่าซึ่งนอกจากจะสร้างไฟล์ไปป์แล้วยังสามารถสร้างไฟล์ดีไวซ์ได้ด้วย.

mkfifo อ้างอิงหน้า 397

ตัวอย่างที่ 3.15: การสร้าง named pipe

```
$ mkfifo in_out↵
$ ls -lF in_out↵
prw-r--r--    1 poonlap  users           0 Jun 17 00:25 in_out|
```

คำสั่ง `ls -l` แสดงรายเอียดของไฟล์ในช่วงแรกเป็น `prw-r--r--`. ตัวอักษร `p` ในที่นี้บ่งบอกว่าไฟล์ดังกล่าวเป็นไฟล์ไปป์. ถ้าใช้ตัวเลือก `-F` ประกอบ, `ls` จะเติมเครื่องหมาย `|` ท้ายชื่อไฟล์เพื่อให้รู้ว่าเป็นไฟล์ไปป์.

ไฟล์ไปป์เป็นไฟล์สำหรับรับส่งข้อมูลระหว่างโปรเซส. การใช้ไฟล์ไปป์ก็เช่นเดียวกันกับไปป์คือต้องมีโปรแกรมที่ส่งข้อมูลเข้าไฟล์ไปป์และมีอีกโปรแกรมหนึ่งรับข้อมูลจากไฟล์ไปป์. ตัวอย่างเช่น

ตัวอย่างที่ 3.16: รับส่งข้อมูลระหว่างโปรเซสด้วยไฟล์ไปป์.

```
$ echo abc > in_out &↵
[1] 10024
$ cut -b2 in_out↵                                ← ลากัดเอาไบต์ตัวที่สองจากไฟล์ไปป์
b
[1]+  Done                    echo abc >in_out
```

ให้สังเกตว่าถ้าส่งข้อมูลลงไฟล์ที่ไม่ใช่ไปป์ด้วย echo เมื่อสั่งคำสั่งเสร็จแล้วจะสามารถสั่งคำสั่งต่อไปได้เลย. แต่จากตัวอย่างถ้าไม่สั่งคำสั่ง echo แบบ background (หน้า 63) จะไม่สามารถสั่งคำสั่งต่อไปได้เพราะข้อมูลที่ส่งไปทางไฟล์ไปป์ไม่มีโปรเซสมารับข้อมูลไป. การที่โปรเซส echo ค้างไปชั่วคราว (ถ้าสั่งคำสั่งแบบ foreground) แบบนี้เรียกว่าโปรเซสนั้นถูก *block* โดยเคอร์เนล. ถ้าไม่มีโปรเซสมารับข้อมูลจากไปป์ไปก็โปรเซสที่เป็นตัวป้อนข้อมูลก็จะค้างอยู่อย่างนั้น. การที่เราสั่งคำสั่งแบบ background ก็เพื่อที่จะสั่งคำสั่งต่อไปได้ แต่โปรเซส echo ก็ยังถูกบล็อกอยู่จนกว่าจะมีโปรเซสอื่นมารับข้อมูลจากไปป์ไป. เมื่อสั่งคำสั่ง cut ให้อ่านข้อมูลจากไฟล์ไปป์เสร็จเรียบร้อยแล้ว, คำสั่ง echo ก็จบการทำงานได้เป็นปรกติ.

มีน้อยครั้งที่จะใช้ไฟล์ไปป์แต่ไฟล์ไปป์จะช่วยแก้ปัญหาได้ในบางกรณีเช่น สมมติว่ามีโปรแกรมชื่อ genbigfile เป็นโปรแกรมประมวลผลและเก็บข้อมูลเป็นไฟล์ขนาดใหญ่มากชื่อ bigfile. เพื่อที่จะประหยัดพื้นที่ในการเก็บไฟล์นี้เราจึงต้องอัดบีบด้วยโปรแกรมเช่น gzip ให้ไฟล์มีขนาดเล็กลง. ถ้าโปรแกรม genbigfile สามารถส่งข้อมูลออกทาง stdout ได้, เราสามารถใช้ gzip บีบอัดข้อมูลจาก stdin ได้ทันทีโดยไม่ต้องรอให้เขียนไฟล์ลงในฮาร์ดดิสก์ก่อนโดยใช้ไปป์. แต่โปรแกรม genbigfile ที่สมมตินี้ไม่สามารถแสดงผลทาง stdout ได้, และจุดนี้เองที่ไฟล์ไปป์มีบทบาทสำคัญ. เราสามารถบีบอัดไฟล์ผลลัพธ์ bigfile ได้โดยไม่ต้องรอให้บันทึกไฟล์นี้ลงฮาร์ดดิสก์ก่อนได้ดังนี้.

สมมติว่าสั่งคำสั่ง genbigfile แล้วจะสร้างไฟล์ชื่อ bigfile ให้โดยปริยาย.

ตัวอย่างที่ 3.17: การใช้ไฟล์ไปป์

```
$ mkfifo bigfile↵                                  ← เตรียม fifo เป็นชื่อไฟล์ที่จะเกิดขึ้น
$ genbigfile &↵                          ← คำสั่งนี้ไม่ทำงานทันทีเพราะถูกบล็อกโดยเคอร์เนล
$ gzip - < bigfile > bigfile.gz↵                   ← ใช้ gzip บีบอัดลงข้อมูลผ่านไฟล์ไปป์
```

ก่อนที่เราสั่งคำสั่ง genbigfile ให้สร้างไฟล์ไปป์ที่มีชื่อเดียวกันกับไฟล์ที่จะเกิดขึ้นก่อน. หลังจากนั้นค่อยสั่งคำสั่ง genbigfile. เมื่อสั่งคำสั่ง genbigfile ไปแล้ว, โปรเซสนี้จะถูกเคอร์เนลบล็อกเพราะโปรเซสนี้ส่งข้อมูลออกทางไฟล์ไปป์และยังไม่มีโปรเซสปลายทางรับข้อมูล. โดยปรกติคำสั่ง gzip รับอาร์กิวเมนต์เป็นไฟล์ได้แต่ไฟล์ bigfile เป็นไฟล์พิเศษจึงต้องใช้การรีไดเรกข้อมูลจากไฟล์ bigfile เข้ามาทาง stdin และใช้ gzip กับตัวเลือก “-” ซึ่งหมายถึงรับข้อมูลเข้าจาก stdin. เมื่อ gzip เริ่มทำงาน, genbigfile ก็ออกจากการถูกบล็อกแล้วเริ่มทำงานส่งข้อมูลผ่านไปป์, และ gzip ก็รับข้อมูลนั้นมาบีบอัดเก็บเป็นไฟล์ bigfile.gz ต่อไป. วิธีการนี้ทำให้บีบอัดข้อมูลได้ทันทีโดยที่ไม่ต้องเขียนข้อมูลนั้นเป็นไฟล์ลงในฮาร์ดดิสก์ก่อน (สมมติว่าโปรแกรม genbigfile ส่งข้อมูลออกเป็นไฟล์ได้อย่างเดียว, ใช้ stdout ไม่ได้).

gzip อ้างอิงหน้า 388

3.7 UNIX Domain socket

socket ที่จะแนะนำต่อไปนี้ไม่ใช่ network socket แต่เป็น *UNIX domain socket*. UNIX domain socket นี้เป็นไฟล์ที่สร้างโดยโปรแกรม. ผู้ใช้ไม่สามารถสร้างไฟล์พิเศษนี้ด้วยคำสั่งเหมือนกับ FIFO. ไฟล์ socket นี้จะใช้แลกเปลี่ยนข้อมูลระหว่างโปรเซส (IPC,

socket ▶ วิธีการที่ใช้เขียนโปรแกรมติดต่อกันผ่านทางเน็ตเวิร์ก. socket นี้จะเกี่ยวข้องกับ TCP, IP และ port.

ในที่นี้จะเรียกว่าไฟล์ socket เพื่อความชัดเจน.

inter-process communications) เช่นเดียวกับ FIFO. นอกเหนือจากนี้แล้วไฟล์ socket ยังจะใช้โดยโปรแกรมที่ใช้งานผ่านทางเน็ตเวิร์กด้วย.

บ้างก็เรียกไฟล์ socket ว่า IPC socket.

ตัวอย่างของโปรแกรมที่ใช้ไฟล์ socket ได้แก่ X เซิฟร์เวอร์. โดยปรกติ X เซิฟร์เวอร์สามารถรับการติดต่อจาก client ได้ทางเน็ตเวิร์ก socket และไฟล์ socket. ถ้าเป็นการติดต่อกับเซิฟร์เวอร์ผ่านทางเน็ตเวิร์ก, ตัวเซิฟร์เวอร์ก็จะใช้เน็ตเวิร์ก socket, ถ้าเป็นการติดต่อกับเซิฟร์เวอร์จากเครื่องที่รันเซิฟร์เวอร์นั้นเช่นระบบเดกสท็อปส่วนบุคคลก็จะใช้ไฟล์ socket แทนที่จะใช้เน็ตเวิร์ก socket.

ไฟล์ socket เป็นไฟล์ที่สร้างโดยตัวโปรแกรมที่ใช้ไฟล์ socket นั้น. ผู้ใช้ไม่ต้องใส่ใจกับการสร้างไฟล์เหล่านี้. ส่วนใหญ่โปรแกรมที่ใช้ไฟล์ socket จะสร้างไฟล์ socket ที่ไดเรกทอรี `/tmp` เช่นไฟล์ `/tmp/.X11-unix/X0` เป็นไฟล์ socket ที่ใช้โดย X เซิฟร์เวอร์.

ตัวอย่างที่ 3.18: ไฟล์ socket ที่ใช้โดย X เซิฟร์เวอร์

```
$ ls -lF /tmp/.X11-unix/X0↵
srwxrwxrwx    1 root     root            0 Jun 23 20:57 /tmp/.X11-unix/X0=
```

คำสั่ง `ls -l` แสดงรายละเอียดไฟล์ในช่วงแรกเป็น `srwxrwxrwx` และตัวอักษร `s` บอกว่าไฟล์นี้คือไฟล์ socket. ตัวเลือก `-F` ช่วยเติมเครื่องหมาย `=` หลังชื่อไฟล์เพื่อทำให้รู้ว่าไฟล์นี้คือไฟล์ socket.

3.8 i-node

จากที่ได้แนะนำไปแล้วว่าไดเรกทอรีเป็นไฟล์พิเศษที่เก็บชื่อไฟล์และที่อยู่ของไฟล์. โดยปรกติแล้วไฟล์จะเก็บอยู่ในฮาร์ดดิสก์และที่อยู่ของไฟล์นี้เรียกว่า *i-node (index node)*. นอกจาก i-node จะเป็นตัวบอกที่อยู่ของไฟล์แล้วยังมีข้อมูลเกี่ยวกับไฟล์ต่าง ๆเช่น เจ้าของหรือกรุปที่เกี่ยวข้องกับไฟล์, สิทธิ์การใช้ไฟล์, และประเภทของไฟล์. ตอนที่ติดตั้งระบบปฏิบัติการลินุกซ์, i-node จะถูกสร้างช่วงที่สร้าง*ระบบไฟล์ (file system)* หลังจากที่แบ่ง*พาร์ทิชัน (partition)*.

file system ▶ *ระบบไฟล์.* วิธีการที่ระบบปฏิบัติใช้ในการเข้าถึงข้อมูลที่อยู่ในหน่วยความจำถาวร. ระบบไฟล์ที่ใช้ในลินุกซ์มีหลายประเภทได้เช่น ext2, ext3, xfs ฯลฯ.

partition ▶ *พาร์ทิชัน.* พื้นที่ในฮาร์ดดิสก์ที่แบ่งเป็นส่วน ๆ. เมื่อแบ่งพาร์ทิชันแล้วยังไม่สามารถใช้งานได้ทันทีต้องสร้างระบบไฟล์ (file system) ก่อน.

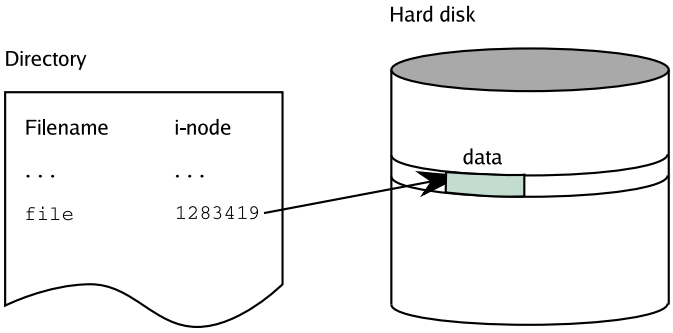
เวลาที่เราสั่งคำสั่งประมวลผลข้อมูลที่อยู่ในไฟล์, เราจะมักจะใช้ชื่อไฟล์เป็นอาร์กิวเมนต์ของคำสั่ง. **ชื่อไฟล์**นี้เป็นเพียงแค่**ชื่อ**, ส่วนพื้นที่ในหน่วยความจำถาวรที่เป็นที่เก็บข้อมูลนั้นคือ**ไฟล์**ตัวจริง. ระบบปฏิบัติการจะอ้างอิงถึงไฟล์ด้วย i-node. และใช้ชื่อไฟล์แสดงให้มนุษย์เข้าใจโดยที่ไม่ต้องรู้ i-node ของไฟล์นั้น.

กรณีที่ต้องการดูค่า i-node ของไฟล์ทำได้ด้วยคำสั่ง `ls -i`.

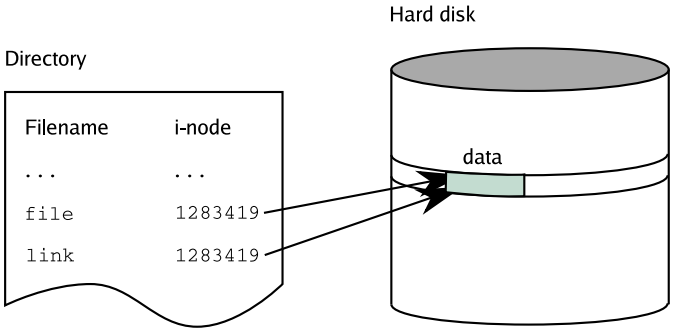
ตัวอย่างที่ 3.19: แสดงค่า i-node ของไฟล์

```
$ ls -i file↵
1283419 -rwxr--r--    1 poonlap  users          7 May 16 08:38 file
```

จากตัวอย่างค่า i-node ของไฟล์ `file` (ชื่อไฟล์) คือ 1283419. และจากรูปที่ 3.6 จะเห็นได้ว่าชื่อไฟล์หนึ่งชื่อจะจับคู่กับ i-node หนึ่งค่า. ในบางกรณีเราสามารถสร้างชื่อไฟล์ให้อ้างอิงถึงไฟล์ (i-node) ที่มีอยู่แล้วได้ดังรูปที่ 3.7



รูปที่ 3.6: ความสัมพันธ์ระหว่างไฟล์, ชื่อไฟล์และ i-node



รูปที่ 3.7: ฮาร์ดลิงค์ (hard link)

3.8.1 ฮาร์ดลิงค์

ในช่วงนี้ขอให้ผู้อ่านแยกแยะความแตกต่างระหว่าง “ไฟล์” กับ “ชื่อไฟล์” ให้ดี.

การสร้างชื่อไฟล์ใหม่อ้างอิงถึงไฟล์ (ไฟล์จริง) ที่มีอยู่แล้วเรียกว่าการสร้าง*ฮาร์ดลิงค์ (hard link)*. ซึ่งในความเป็นจริงแล้วชื่อไฟล์ที่มีตั้งแต่แรก (ในตัวอย่างคือ `file`) ก็เรียกว่าเป็นฮาร์ดลิงค์เช่นกัน.

`ln` อ้างอิงหน้า 396

ตัวอย่างต่อไปนี้จะเป็นการใช้คำสั่งสร้างฮาร์ดลิงค์ซึ่งได้แก่คำสั่ง `ln` สร้างลิงค์เหมือนกับที่แสดงในรูปที่ 3.7.

ตัวอย่างที่ 3.20: การสร้างฮาร์ดลิงค์.

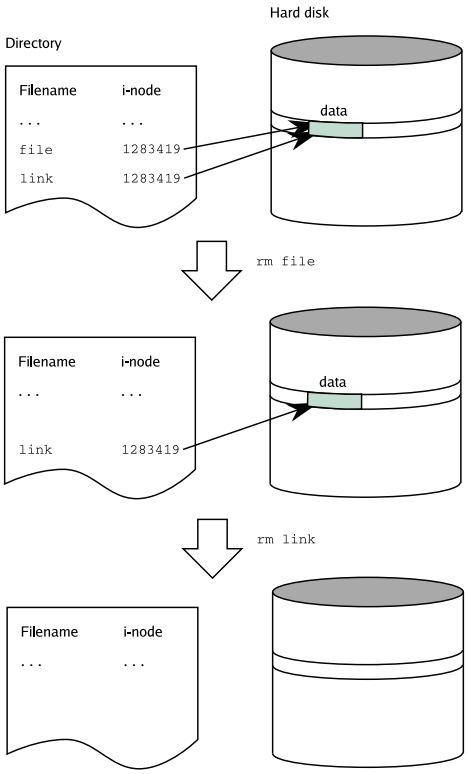
```
$ ln file link↵
$ ls -li file link↵
1283419 -rwxr--r--    2 poonlap  users           7 May 16 08:38 file
1283419 -rwxr--r--    2 poonlap  users           7 May 16 08:38 link
                      ↑ จำนวนฮาร์ดลิงค์ที่อ้างอิงกับ i-node 1283419
```

จะเห็นว่าชื่อไฟล์ (ฮาร์ดลิงค์) ทั้งสองอ้างอิงไฟล์ไฟล์เดียวกัน. ให้สังเกตว่าค่า i-node ของไฟล์ทั้งสองมีค่าเหมือนกัน, นั้นคือเป็นไฟล์ไฟล์เดียวกันแต่มีการอ้างอิงด้วยชื่อไฟล์สองชื่อ. จากตัวอย่าง `ls -li` ยังแสดงจำนวนฮาร์ดลิงค์ที่จับคู่กับ i-node ด้วย.

ถ้าเราลบไฟล์ด้วยคำสั่ง `rm` เช่น “`rm file`”, ไฟล์ `file` จะหายไป (รูปที่ 3.8). แต่ในความเป็นจริงแล้ว `rm` เป็นการลบชื่อไฟล์ไม่ได้ลบไฟล์จริง. ไฟล์จริงที่อ้างอิงด้วย

i-node นั้นจะถูกลบหรือหายไปก็ต่อเมื่อไม่มีฮาร์ดลิงค์อ้างอิงถึง. ดังนั้นถ้าไม่มีฮาร์ดลิงค์ `link` อยู่, เมื่อสั่งคำสั่ง "`rm file`" ทั้งชื่อไฟล์และไฟล์จริงก็จะหายไปเพราะไฟล์นั้นไม่มีฮาร์ดลิงค์อื่น ๆมาอ้างอิงอีกต่อไป.

ไฟล์ข้อมูลนั้นถ้าไม่มีฮาร์ดลิงค์มาอ้างอิงก็ไม่หายไปไหน. แต่ถ้ามีการสร้างไฟล์ใหม่และเผอิญระบบปฏิบัติการใช้ i-node นั้นเป็นที่เก็บไฟล์ใหม่, ข้อมูลที่เคยอยู่ในไฟล์เก่าก็จะหายไปจริง ๆ และกู้กลับมาไม่ได้.



รูปที่ 3.8: การลบไฟล์ด้วย `rm`

คำสั่งที่อธิบายประกอบรูป 3.8 ได้แก่ตัวอย่างต่อไปนี้. จะเห็นว่าค่า i-node ไม่เปลี่ยนแปลง, หมายความว่าไฟล์ยังไม่ได้ถูกลบไปเพราะยังมีฮาร์ดลิงค์ (ชื่อไฟล์ link) อ้างอิงถึงไฟล์จริงอยู่.

ตัวอย่างที่ 3.21: การลบไฟล์ด้วย `rm`

```
$ rm file↵
$ ls -li link↵
1283419 -rwxr--r--    1 poonlap  users          13 Jun 26 02:35 link
                      ↑ จำนวนฮาร์ดลิงค์ลดลงหนึ่งตัว
```

ไดเรกทอรีเป็นไฟล์พิเศษประเภทหนึ่ง, ดังนั้นไดเรกทอรีก็สามารถมีฮาร์ดลิงค์ได้เช่นกัน. แต่ฮาร์ดลิงค์ของไดเรกทอรีนี้เคอร์เนลจะเป็นตัวจัดการสร้างตอนนี้สร้างไดเรกทอรี. สำหรับผู้ใช้ทั่วไปหรือแม้กระทั่งก็ไม่สามารถสร้างฮาร์ดลิงค์ด้วยคำสั่ง `ln` เหมือนกับฮาร์ดลิงค์ของไฟล์ธรรมดา.

สมมติว่าเราอยู่ในไดเรกทอรี `~/tmp` และสร้างไดเรกทอรีใหม่ชื่อ `subdir` ด้วยคำสั่ง `mkdir`.

ตัวอย่างที่ 3.22: สร้างไดเรกทอรี

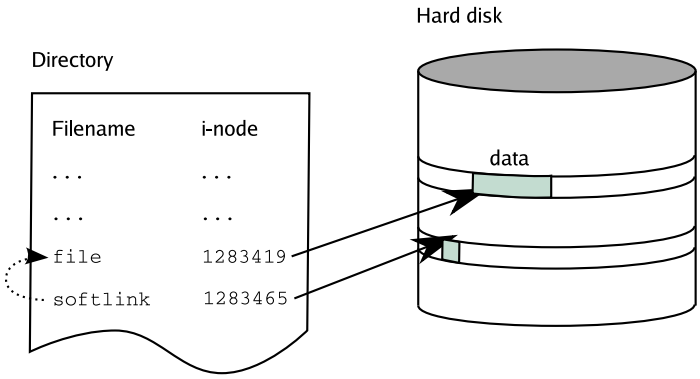
```
$ mkdir subdir↵
$ ls -li↵
total 4
1283467 drwxr-xr-x    2 poonlap  users        4096 Jun 28 00:51 subdir/
```

จะเห็นได้ว่า i-node ของไดเรกทอรีที่สร้างใหม่จะลิงค์อ้างอิงอยู่สองตัว. ตัวหนึ่งคือชื่อไดเรกทอรี (`subdir`) และอีกตัวหนึ่งคือลิงค์พิเศษที่เคอร์เนลสร้างให้โดยอัตโนมัติ. ลิงค์พิเศษที่เคอร์เนลสร้างให้นี้อยู่ใต้ไดเรกทอรีที่สร้างและมีชื่อเป็น . (จุด) และไดเรกทอรี . นี้ใช้สำหรับอ้างอิงไดเรกทอรีที่อยู่ในปัจจุบัน.

นอกจากไดเรกทอรี . แล้ว, เคอร์เนลยังสร้างลิงค์พิเศษอีกตัวคือ .. ใต้ไดเรกทอรีที่สร้างใหม่เป็นลิงค์ไปหาไดเรกทอรีแม่. ดังนั้นเมื่อมีการสร้างไดเรกทอรีใหม่ในไดเรกทอรีใด ๆ, จำนวนลิงค์ที่อ้างอิงไดเรกทอรีที่สร้างนั้นจะเพิ่มขึ้นเสมอ. ในตัวอย่างข้างต้น, ไดเรกทอรี `~/tmp` จะมีจำนวนลิงค์ที่อ้างอิงเพิ่มขึ้น.

3.8.2 ซอฟต์ลิงค์

ฮาร์ดลิงค์มีข้อจำกัดบางประการได้แก่, ไม่สามารถสร้างฮาร์ดลิงค์ไปหาไฟล์ที่อยู่คนละพาร์ทิชันได้ และไม่สามารถสร้างฮาร์ดลิงค์ไปหาไดเรกทอรี. การแก้ไขข้อจำกัดนี้สามารถทำได้โดยการสร้าง*ซอฟต์ลิงค์ (soft link)*. ซอฟต์ลิงค์มีชื่อเรียกอีกอย่างว่า *symbolic link* คือไฟล์ที่มีตัวตนจริงในหน่วยความจำถาวรโดยที่ข้อมูลของไฟล์นี้จะไปเชื่อมต่อกับไฟล์ปลายทางที่กำหนดไว้ (รูปที่ 3.9).



รูปที่ 3.9: ซอฟต์ลิงค์ (soft link)

การสร้างซอฟต์ลิงค์จะใช้คำสั่ง `ln` เหมือนกับการสร้างฮาร์ดลิงค์แต่จะใช้ตัวเลือก `-s` ประกอบ.

ตัวอย่างที่ 3.23: การสร้างซอฟต์ลิงค์

```
$ ln -s file softlink↵
$ ls -li file softlink↵
1283419 -rwxr--r--    1 poonlap  users          13 Jun 26 02:35 file
1283465 lrwxrwxrwx    1 poonlap  users           4 Jun 27 22:17 softlink -> file
```

รายละเอียดของคำสั่ง ls ช่วงแรกได้แก่ lrwxrwxrwx, และตัวอักษร l นี้บ่งบอกว่าไฟล์นี้คือซอฟต์ลิงค์. ให้สังเกตว่าค่า i-node เป็นคนละค่ากับไฟล์ที่ลิงค์ไปหา. และรายละเอียดของตัวเลือก -l จะแสดงไฟล์ที่ลิงค์ไปหาด้วย. ถ้าใช้ตัวเลือก -F, คำสั่ง ls จะเติมตัวอักษร @ หลังชื่อไฟล์เพื่อบ่งบอกว่าไฟล์นั้นเป็นลิงค์.

3.9 รายละเอียดของไฟล์

การแสดงรายละเอียดของไฟล์จะใช้คำสั่ง ls กับตัวเลือก -l.

ตัวอย่างที่ 3.24: คำสั่ง ls -l แสดงรายละเอียดของไฟล์.

```
$ ls -l↵
total 4
drwxr-xr-x    2 poonlap  users        4096 Jul 20 00:21 bar
-rw-r--r--    1 poonlap  users           0 Jul 20 00:21 foo
```

เมื่อใช้ตัวเลือก -l ดูรายละเอียดของไดเรกทอรีตามตัวอย่าง, บรรทัดแรก (total 4) จะแสดงจำนวนพื้นที่ที่ใช้ไปโดยไดเรกทอรีนั้นในหน่วยบล็อก (block). จากนั้นจะแสดงรายละเอียดของไฟล์หรือไดเรกทอรีบรรทัดต่อบรรทัด.

โดยทั่วไป 1 บล็อกจะหมายถึง 1024 ไบต์.

ตัวอักษรที่เรียงกัน 10 เช่น drwxr-xr-x ในตัวอย่างจะบอกประเภทของไฟล์โดยใช้อักษรตัวแรก, ส่วนอักษร 9 ตัวที่เหลือจะแสดงรายละเอียดสิทธิ์การใช้ไฟล์หรือไดเรกทอรี่นั้น. ตัวเลขที่อยู่ถัดจากตัวอักษร 10 ตัวคือจำนวนฮาร์ดลิงค์ที่อ้างอิงไฟล์. ไดเรกทอรีจะมีจำนวนฮาร์ดลิงค์อย่างน้อย 2 ตัว. ฮาร์ดลิงค์ตัวแรกคือชื่อไดเรกทอรี (bar) และฮาร์ดลิงค์อีกตัวคือชื่อไดเรกทอรี . ที่อ้างอิงไดเรกทอรีตัวเอง (bar/.).

ตารางที่ 3.3: ตัวอักษรที่ใช้แสดงประเภทของไฟล์.

ตัวอักษร	คำอธิบาย
d	ไดเรกทอรี
b	ไฟล์ดีไวซ์ (block device)
c	ไฟล์ดีไวซ์ (character device)
l	ซอฟต์ลิงค์
p	ไฟล์ไปป์
s	ไฟล์ socket
-	ไฟล์ธรรมดา

รายละเอียดที่แสดงต่อจำนวนฮาร์ดลิงค์คือชื่อเจ้าของไฟล์ซึ่งได้แก่ชื่อล็อกอิน, และชื่อกลุ่ม (group) ของไฟล์. โดยปรกติผู้ใช้ในระบบมีอยู่ในกลุ่มผู้ใช้ที่กำหนดไว้กลุ่มใดกลุ่มหนึ่งเช่นผู้ใช้ทั่วไปจะอยู่ในกลุ่ม users. เวลาที่สร้างไฟล์หรือไดเรกทอรีขึ้นมาก็จะถือว่าเจ้าของไฟล์คือผู้ที่สร้างไฟล์นั้น, และกลุ่มของไฟล์คือกลุ่มที่ผู้สร้างไฟล์นั้นอยู่.

ข้อมูลที่ถัดจากเจ้าของไฟล์และกลุ่มได้แก่ ขนาดของไฟล์, timestamp และชื่อไฟล์. ขนาดของไฟล์จะแสดงเป็นจำนวนไบต์. หรือใช้ตัวเลือกที่แสดงในตารางที่ 3.4 แสดงขนาดของไฟล์ในหน่วยที่ต้องการ. timestamp คือเวลาที่ไฟล์นั้นถูกแก้ไขเป็นครั้งสุดท้าย.

ตารางที่ 3.4: ตัวเลือกของ `ls` ที่เกี่ยวกับการแสดงขนาดของไฟล์.

ตัวเลือก	ความหมาย
`-h`	แสดงหน่วยที่มนุษย์อ่านแล้วเข้าใจเช่น M (megabytes) หรือ G (gigabytes) เป็นต้น.
`-k`	แสดงหน่วยเป็นกิโลไบต์.
`-s`	แสดงหน่วยเป็นบล็อก (1024 ไบต์)
`-S`	เรียงลำดับไฟล์ที่แสดงตามพื้นที่ไฟล์.

3.9.1 คำสั่ง `stat`

stat อ้างอิงหน้า 399

คำสั่งที่แสดงรายละเอียดของไฟล์ได้ละเอียดกว่าคำสั่ง `ls` คือ `stat`. คำสั่ง `stat` จะแสดงสถานะของไฟล์หรือระบบไฟล์ที่ใช้บันทึกไฟล์นั้น.

ตัวอย่างที่ 3.25: แสดงสถานะของไฟล์ด้วย `stat`.

```
$ stat foo↵
  File: ‘foo’
  Size: 2               Blocks: 8          IO Block: 4096   regular file
Device: 342h/834d       Inode: 575494      Links: 1
Access: (0600/-rw-------)  Uid: ( 1000/ poonlap)   Gid: (  100/   users)
Access: 2004-07-20 23:03:01.000000000 +0900
Modify: 2004-07-20 01:29:37.000000000 +0900
Change: 2004-07-20 23:02:14.000000000 +0900
```

ผลลัพธ์ของคำสั่งมีคำอธิบายสั้น ๆและมีความหมายในตัวเอง. ในที่นี้ขออธิบายเกี่ยวกับ timestamp ของไฟล์ซึ่งจากคำสั่ง `stat` จะแสดง timestamp สามอย่างคือ

1. Time of last access (atime) ได้แก่เวลาล่าสุดที่มีการเปิดไฟล์. การใช้ไฟล์ในที่นี้รวมถึงการอ่าน, สร้าง, แก้ไขไฟล์.

2. Time of last modification (mtime) ได้แก่เวลาล่าสุดที่มีการแก้ไขเนื้อหาไฟล์.

3. Time of last status change (ctime) ได้แก่เวลาล่าสุดที่มีการเปลี่ยนสภาพของไฟล์ เช่นการแก้สิทธิ์การใช้ไฟล์.

จะเห็นว่าเคอร์เนลจะบันทึกเวลา atime ทุกครั้งเมื่อเปิดไฟล์ใด ๆก็ตาม. สำหรับระบบที่ไม่สนใจข้อมูลของ atime อาจจะปรับแต่งระบบให้ไม่บันทึกค่า atime ก็ได้โดยตอนที่ mount ให้ใช้ตัวเลือก `-o noatime`. การไม่บันทึก atime อาจจะช่วยเพิ่มประสิทธิภาพการทำงานของระบบได้ด้วย.

3.10 การเปลี่ยนเจ้าของและกลุ่มของไฟล์

คำสั่ง chown เป็นคำสั่งสำหรับเปลี่ยนเจ้าของไฟล์ซึ่งผู้ใช้ที่จะเปลี่ยนเจ้าของไฟล์นั้นคือ root เท่านั้น. คำสั่ง chown นอกจากจะเปลี่ยนเจ้าของไฟล์ได้แล้วยังเปลี่ยนกลุ่มผู้ใช้ที่เกี่ยวข้องได้พร้อม ๆกันตามตัวอย่างต่อไปนี้.

chown อ้างอิงหน้า 405

ตัวอย่างที่ 3.26: การเปลี่ยนเจ้าของไฟล์.

```
# ls -l foo↵
-rw-r--r--    1 root     root           0 Jul 24 13:28 foo
# chown poonlap:users foo↵
# ls -f foo↵
-rw-r--r--    1 poonlap  users          0 Jul 24 13:28 foo
```

สำหรับผู้ใช้ทั่วไปแล้วไม่มีสิทธิ์ที่จะเปลี่ยนเจ้าของไฟล์, แต่สามารถเปลี่ยนกลุ่มผู้ใช้ที่เกี่ยวข้องกับไฟล์นั้นได้ถ้าผู้ใช้นั้นอยู่ในกลุ่มที่ต้องการเปลี่ยน. เช่นถ้าผู้ใช้อยู่ในกลุ่ม wheel ก็สามารถเปลี่ยนกลุ่มผู้ใช้ที่เกี่ยวข้องกับไฟล์นั้นไปเป็น wheel ได้ด้วยคำสั่ง chgrp.

chgrp อ้างอิงหน้า 394

ตัวอย่างที่ 3.27: การเปลี่ยนกลุ่มของไฟล์.

```
$ ls -l foo↵
-rw-r--r--    1 poonlap  users          0 Jul 24 13:28 foo
$ chgrp wheel foo↵
$ ls -l foo↵
-rw-r--r--    1 poonlap  wheel          0 Jul 24 13:28 foo
```

3.11 สิทธิการใช้ไฟล์

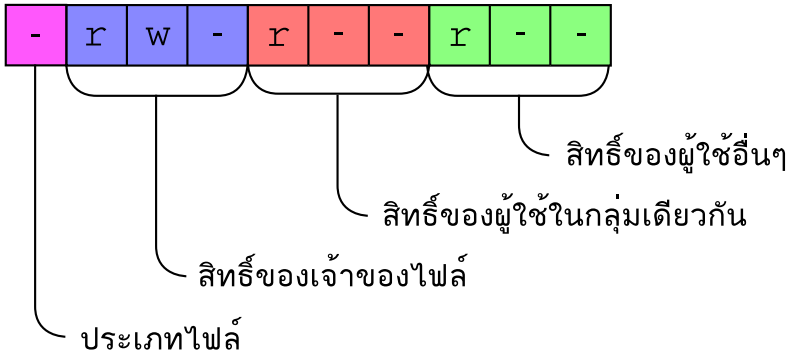
เนื่องจากระบบปฏิบัติการลินุกซ์และยูนิกซ์สามารถมีผู้ใช้ในระบบได้หลายคน, จึงมีแนวคิดเกี่ยวกับการกำหนดสิทธิ์ (permission) อนุญาติให้ผู้ใช้ในระบบใช้ไฟล์ที่ต้องการได้. ไฟล์หรือไดเรกทอรีในลินุกซ์แต่ละไฟล์จะมีเจ้าของไฟล์, และกลุ่มของไฟล์. เราสามารถอนุญาติประเภทการใช้งานไฟล์แบบต่าง ๆเช่นการอ่าน, สร้าง, แก้ไข ฯลฯ โดยแยกประเภทผู้ใช้ไฟล์ออกเป็นเจ้าของไฟล์ (owner), ผู้ที่อยู่ในกลุ่มเดียวกัน (group) และผู้ใช้อื่น ๆ (other).

คำสั่ง ls กับตัวเลือก -l จะแสดงสิทธิ์การใช้ไฟล์ของไฟล์นั้น ๆ. จากตัวอย่างที่ 3.24 ส่วนที่เกี่ยวข้องกับสิทธิ์การใช้ไฟล์คือตัวอักษร rw-r--r-- ซึ่งเป็นตัวอักษรที่อยู่ถัดจากตัวอักษรที่แสดงประเภทของไฟล์. อักษรสามตัวแรก (rw-) เป็นสิทธิ์ที่เจ้าของไฟล์ได้รับอนุญาต. อักษรสามตัวถัดไปเป็นสิทธิ์ที่ผู้ใช้ที่อยู่ในกลุ่มเดียวกันได้รับอนุญาต. และอักษรสามตัวสุดท้ายเป็นสิทธิ์ที่ผู้ใช้นอกเหนือจากที่กล่าวมาแล้วได้รับอนุญาต.

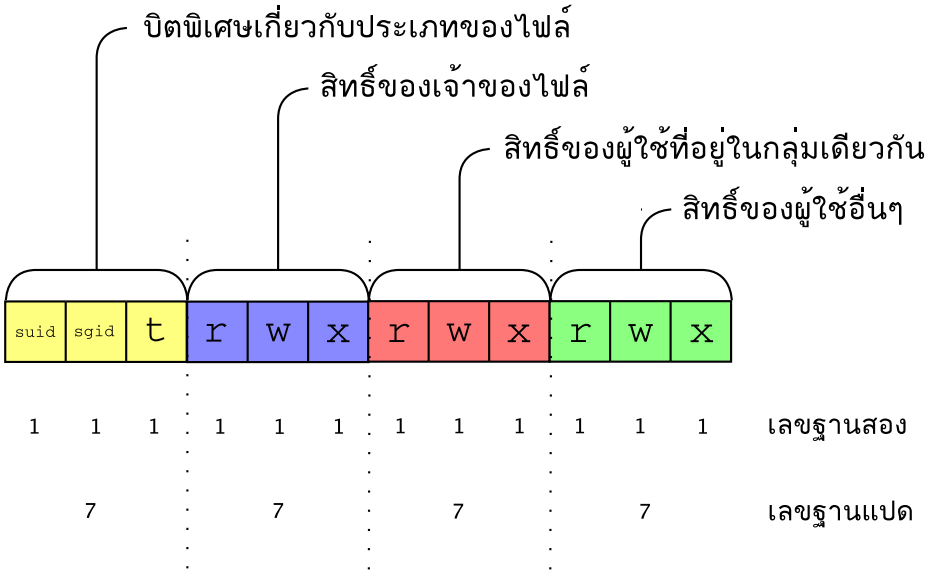
ในระบบปฏิบัติการยูนิกซ์และลินุกซ์สิทธิ์การใช้ไฟล์กำหนดโดย*โหมด (mode)* ซึ่งเป็นค่าตัวเลขที่แสดงได้ด้วยบิต 9 ตัว (รูปที่ 3.11) ที่เกี่ยวกับสิทธิ์การใช้ไฟล์และบิตพิเศษ 3 ตัวที่เกี่ยวกับคุณสมบัติของไฟล์นั้น.

ช่วงที่ 3.12 จะอธิบายเกี่ยวกับบิตพิเศษ.

บิตที่เกี่ยวกับสิทธิ์การใช้ไฟล์จะอยู่ในตำแหน่งล่าง 9 หลัก, และใช้สัญลักษณ์ r, w



รูปที่ 3.10: ผลลัพธ์ของคำสั่ง `ls -l` ที่เกี่ยวกับสิทธิ์การใช้ไฟล์และประเภทของไฟล์.



รูปที่ 3.11: ความสัมพันธ์ระหว่างโหมดและตำแหน่งบิต.

และ `x` แทนบิตในแต่ละตำแหน่ง. เช่นถ้าบิตในหลักแรกเป็น 1 ตัวอักษรที่ใช้ในตำแหน่งนั้นจะแทนค่าบิตด้วย `x`, ถ้าไม่มีการตั้งค่าบิตนี้ (ค่าเป็น 0) ก็จะแสดงด้วยตัวอักษร `-`.

3.11.1 สิทธิ์การกระทำ

สิทธิ์ที่สามารถกระทำได้กับไฟล์แบ่งออกเป็น 3 ชนิดใหญ่ ๆได้แก่ read permission (`r`), write permission (`w`) และ execute permission (`x`). การกระทำเหล่านี้มีความหมายแตกต่างกันขึ้นอยู่กับว่าตัวถูกกระทำเป็นไฟล์หรือเป็นไดเรกทอรี.

การกระทำกับไฟล์

- `r` : อนุญาตให้เปิดอ่านข้อมูลในไฟล์ได้.
- `w` : อนุญาตให้แก้ไขข้อมูลในไฟล์ได้.

- `x` : อนุญาตให้ไฟล์นั้นกระทำการได้. ไฟล์ที่กระทำการได้หมายถึงไฟล์โปรแกรม. ถ้ามีไฟล์โปรแกรมแต่ไม่อนุญาติสิทธิ์กระทำการกับไฟล์โปรแกรมนั้นก็ไม่สามารถทำงานได้.

การกระทำกับไดเรกทอรี

- `r` : อนุญาตให้แสดงรายการไฟล์ต่าง ๆที่อยู่ในไดเรกทอรี. หมายถึงอนุญาตให้เปิดไดเรกทอรีแล้วอ่านข้อมูลที่บันทึกอยู่ในไดเรกทอรี. ข้อมูลที่บันทึกอยู่ในไดเรกทอรีคือรายการไฟล์หรือไดเรกทอรีย่อยต่าง ๆ.

- `w` : อนุญาตให้สร้างหรือลบไฟล์ในไดเรกทอรี. หมายถึงอนุญาตให้แก้ไขรายการไฟล์หรือไดเรกทอรีย่อยที่บันทึกอยู่ในไดเรกทอรี.

- `x` : อนุญาตให้เข้าไปในไดเรกทอรี. หมายถึงอนุญาตให้ใช้คำสั่ง `cd` เปลี่ยนไดเรกทอรีปัจจุบันไปเป็นไดเรกทอรีนั้นได้. เพราะฉะนั้นไดเรกทอรีต้องตั้งค่าบิตนี้ไว้เสมออย่างน้อยในตำแหน่งเจ้าของไฟล์. มิฉะนั้นจะไม่สามารถเปลี่ยนไดเรกทอรีเป็นไดเรกทอรีนั้น.

3.11.2 การแก้สิทธิ์การใช้ไฟล์

เจ้าของไฟล์สามารถแก้สิทธิ์การใช้ไฟล์ได้ด้วยคำสั่ง `chmod`. ผู้ใช้ต้องระบุโหมดซึ่งก็คือรายละเอียดของสิทธิ์ที่ต้องการตั้งเป็นอาร์กิวเมนต์ให้กับคำสั่ง. วิธีการระบุโหมดมี 2 แบบคือการระบุเป็นตัวอักษรหรือระบุเป็นบิตด้วยเลขฐานแปด.

chmod อ้างอิงหน้า 394

การระบุเป็นตัวอักษรเรียกว่า *symbolic mode* โดยมีรูปแบบดังนี้.

```
chmod [ugoa][+-=][rwx] file
```

`[ugoa]` เป็นช่วงที่ระบุว่าใครได้รับอนุญาตให้ทำอะไรกับไฟล์ได้. เครื่องหมาย `+` หมายถึงเพิ่มสิทธิ์. เครื่องหมาย `-` หมายถึงการตัดสิทธิ์. และเครื่องหมาย `=` เป็นการให้สิทธิ์ที่ต้องการ. และสิทธิ์การใช้ไฟล์ได้อธิบายไปแล้วในตารางที่ ??.

ตารางที่ 3.5: ตัวแปรโหมดที่เกี่ยวข้องกับผู้ใช้

ตัวอักษร	ความหมาย
`u`	เจ้าของไฟล์
`g`	ผู้ใช้ที่อยู่ในกลุ่มเดียวกัน
`o`	ผู้ใช้อื่น ๆ
`a`	ผู้ใช้ทั้งหมด. มีค่าเท่ากับ `ugo`

ตัวอย่างเช่นการอนุญาตให้ผู้ใช้ที่อยู่ในกลุ่มเดียวกันดูรายการไฟล์ที่อยู่ในโฮมไดเรกทอรีของตัวเอง, ตั้งค่าได้ดังนี้.

ตัวอย่างที่ 3.28: อนุญาตให้ผู้ใช้ที่อยู่ในกลุ่มเดียวกันดูรายการไฟล์ที่อยู่ในโฮมไดเรกทอรี.

```
$ chmod g+rx ~↵
```

วิธีการระบุโหมดของสิทธิ์การใช้ไฟล์อีกวิธีหนึ่งคือการระบุโหมดด้วยบิตที่ต้องการตั้ง. ให้คิดว่า `rwxrwxrwx` เป็นการเรียงกันของตัวเลขฐานสอง (บิต) 9 ตัวคือ `111111111`. สิทธิ์ที่อนุญาตจะให้บิตที่สอดคล้องเป็น 1 และสิทธิ์ที่ไม่อนุญาตจะให้บิตที่สอดคล้องเป็น 0. เช่น `rw-rw----` จะเขียนในรูปของเลขฐานสองเป็น `110110000`. เวลาใช้กับคำสั่ง `chmod` ให้แปลงบิตเหล่านี้เป็นเลขฐานแปดก่อน.

ตัวอย่างเช่นถ้าต้องการอนุญาตให้เจ้าของไฟล์และคนที่อยู่ในกลุ่มสามารถแก้ไขและอ่านไฟล์ชื่อ `file` ได้ให้ตั้งค่าโหมดเป็น 660.ในตารางที่ 3.6 แสดงโหมดต่าง ๆที่ใช้บ่อยและความหมาย.

ตัวอย่างที่ 3.29: การใช้คำสั่ง `chmod` โดยใช้เลขฐานแปดเป็นโหมด.

```
$ chmod 660 file↵
$ ls -l file↵
-rw-rw----    1 poonlap  users          13 Jun 26 02:35 file
```

นอกจากการตั้งสิทธิ์การใช้ไฟล์ด้วยคำสั่ง `chmod` แล้ว, ถ้าเคอร์เนลที่ใช้อยู่สนับสนุน *Extended attribute* ของระบบไฟล์, *Access Control List (ACL)* ก็จะสามารถตั้งสิทธิ์การใช้ไฟล์ได้ละเอียดขึ้นเช่น ระบุสิทธิ์เฉพาะของผู้ใช้หรือกลุ่ม, ป้องกันไม่ให้ลบไฟล์ เป็นต้น.

3.11.3 สิทธิ์การใช้ไฟล์โดยปริยาย

เราสามารถตั้งค่าสิทธิ์การใช้ไฟล์โดยปริยายได้จากค่าของ umask. umask คือค่าบิต (เลขฐานแปด) ที่ใช้ตั้งสิทธิ์ที่ไม่ต้องการอนุญาต. ตัวอย่างเช่น umask ที่มีค่าเป็น 022 จะหมายถึงถ้ามีการสร้างไฟล์ใหม่, จะไม่อนุญาตให้ผู้ใช้ในกลุ่มเดียวกันและผู้ใช้อื่น ๆเขียนไฟล์. คำสั่งที่ใช้ตั้งค่าหรือดูค่า umask ได้แก่คำสั่ง `umask`. โดยปรกติจะตั้งค่า umask ไว้ในไฟล์ตั้งค่าเริ่มต้นของเชลล์.

umask อ้างอิงหน้า 425

ตัวอย่างที่ 3.30: ตั้งและดูค่า umask.

```
$ umask↵
0022
$ touch foo↵
$ ls -l foo↵
-rw-r--r--    1 poonlap  users           0 Jul 22 00:48 foo
$ umask 066↵
$ touch bar↵
-rw-------    1 poonlap  users           0 Jul 22 00:53 bar
```

เวลาสร้างไฟล์ใหม่, ลินุกซ์จะตั้งสิทธิ์การใช้ไฟล์เป็น 666 และลบออกด้วยค่าของ umask. ดังนั้นถ้า umask มีค่าเป็น 022, สิทธิ์การใช้ไฟล์ของไฟล์ที่สร้างใหม่จะเป็น 666 - 022 ซึ่งได้ผลลัพธ์เป็น 644 (`-rw-r--r--`). การสร้างไดเรกทอรีใหม่ก็มีหลักการคล้าย

ตารางที่ 3.6: โหมดที่ใช้บ่อย.

สิทธิ์การใช้ไฟล์	โหมด	ความหมาย
-rw-------	600	เจ้าของไฟล์เป็นผู้ใช้คนเดียวที่รับอนุญาติอ่านและเขียนไฟล์นั้น.
-rw-r--r--	644	เจ้าของไฟล์สามารถอ่านและเขียนไฟล์ได้, ผู้ใช้คนอื่น ๆอ่านได้อย่างเดียว. ตั้งค่าการใช้ไฟล์แบบนี้เมื่อต้องการให้คนอื่นอ่านไฟล์นี้.
-rw-rw-rw-	666	ใครก็ได้อ่านเขียนไฟล์นี้ได้. การตั้งค่าการใช้ไฟล์แบบนี้ไม่คำนึงถึงความปลอดภัยและไม่เหมาะสมเพราะใครก็ได้สามารถแก้ไขเนื้อหาได้.
-rwx------	700	เจ้าของไฟล์เท่านั้นที่สามารถอ่านเขียนและรันไฟล์นี้. ไฟล์นี้อาจจะเป็นไฟล์โปรแกรมที่ไม่ต้องการให้คนอื่นรันได้นอกจากเจ้าของไฟล์.
-rwxr-xr-x	755	เจ้าของไฟล์มีสิทธิ์เขียนอ่านและรัน. สำหรับผู้ใช้ที่อยู่ในกลุ่มเดียวกันและคนอื่น ๆจะอ่านและรันได้, แต่แก้ไขไฟล์ไม่ได้.
-rwxrwxrwx	777	ใครก็ได้เขียน, อ่าน, และรันไฟล์นั้นได้. การตั้งค่าสิทธิ์แบบนี้ไม่ปลอดภัยเพราะใครก็ได้แก้ไขไฟล์นี้ได้.
-rwx--x--x	711	เจ้าของไฟล์มีสิทธิ์หมด. คนอื่นที่เหลืออนุญาตให้รันไฟล์นั้นได้อย่างเดียว. ในกรณีนี้ผู้ใช้ที่ไม่ใช่เจ้าของไฟล์จะไม่สามารถก็อปปี้ไฟล์ได้.
drwx------	700	ไดเรกทอรีนี้เจ้าของมีสิทธิ์การทำทุกอย่าง. ผู้ใช้อื่น ๆไม่มีสิทธิ์ทำอะไรทั้งสิ้น.
drwxr-xr-x	755	เจ้าของไดเรกทอรีมีสิทธิ์ทุกอย่าง. คนอื่น ๆเข้ามาในไดเรกทอรีนี้ได้และแสดงรายการไฟล์ได้.
drwx--x--x	711	เจ้าของไดเรกทอรีมีสิทธิ์ทุกอย่าง. คนอื่น ๆไม่มีสิทธิ์ดูรายการไฟล์ที่อยู่ในไดเรกทอรี. แต่ถ้ารู้ชื่อไฟล์ที่อยู่ในไดเรกทอรีนั้นก็สามารถอ่านไฟล์นั้นได้.

กัน, ลินุกซ์จะตั้งสิทธิ์การใช้ไฟล์เป็น 777 และลบออกด้วยค่าของ umask. ถ้าค่า umask เป็น 022 ก็ได้ไดเรกทอรีที่มีสิทธิ์การใช้ไฟล์เป็น 755 (-rwxr-xr-x).

3.12 บิต suid, sgid และ sticky

ในค่าของโหมดที่ใช้ตั้งสิทธิ์การใช้ไฟล์มีบิตพิเศษ 3 ตัวที่เกี่ยวกับประเภทของไฟล์อยู่ด้วยซึ่งในช่วงที่ผ่านมาไม่ได้ระบุค่าบิตพิเศษเหล่านี้. บิตพิเศษ 3 ตัวได้แก่.

1. *บิต suid* เป็นบิตที่เกี่ยวกับไฟล์โปรแกรม. สำหรับไฟล์โปรแกรมที่มีการตั้งค่าบิต

suid ไว้, เวลาที่ผู้ใช้ใด ๆรันไฟล์ (โปรแกรม) นั้นจะมีผลเหมือนกับเจ้าของไฟล์นั้นเป็นผู้กระทำการรันโปรแกรม. การที่ UID ของผู้ใช้ที่สั่งคำสั่งนั้นเปลี่ยนไปเป็น UID ของเจ้าของไฟล์ขณะที่โปรแกรมนั้นทำงานอยู่เรียกว่า `effective UID` คือ UID ที่มีผลจริงเมื่อใช้งาน. ไฟล์ที่มีค่าบิต suid อยู่เวลาดูรายละเอียดไฟล์ด้วยคำสั่ง `ls`, จะมีตัวอักษร `s` แทนอยู่ในตำแหน่ง `x` ของเจ้าของไฟล์.

2. *บิต sgid* เป็นบิตที่เกี่ยวกับไฟล์โปรแกรมเช่นกัน. เวลาที่ผู้ใช้ใด ๆรันไฟล์โปรแกรมนั้นจะมีผลให้ผู้ใช้นั้นเหมือนอยู่ในกลุ่มเดียวกับที่ไฟล์ในอยู่. การที่ GID ของผู้ใช้ที่สั่งคำสั่งนั้นเปลี่ยนไปเป็น GID ของกลุ่มไฟล์ขณะที่โปรแกรมนั้นทำงานอยู่เรียกว่า `effective GID`. ไฟล์ที่มีค่าบิต sgid อยู่เวลาดูรายละเอียดไฟล์ด้วยคำสั่ง `ls`, จะมีตัวอักษร `s` แทนอยู่ในตำแหน่ง `x` ของกลุ่มไฟล์.

3. *บิต sticky* ใช้กับไดเรกทอรี, คือผู้ใช้ที่ลบไฟล์ใต้ไดเรกทอรีนั้นต้องเป็นผู้สร้างไฟล์นั้นเท่านั้น. ไดเรกทอรีที่ใช้ sticky bit นี้เช่นไดเรกทอรี `/tmp` เป็นไดเรกทอรีที่ใครก็ได้สามารถสร้างไฟล์ได้แต่ไม่สามารถลบไฟล์ที่ไม่ใช่ของตัวเอง. คำสั่ง `ls` ที่แสดงรายละเอียดไฟล์จะแสดงเครื่องหมาย `t` ที่ตำแหน่ง `x` ของผู้ใช้อื่น ๆ.

ตัวอย่างที่ 3.31: ไฟล์ที่มีค่าบิต suid, sgid และ stick

```
$ ls -l /usr/bin/passwd↵                                  ← ไฟล์ที่มีค่าบิต suid
-rwsr-xr-x    1 root     root        26512 Apr 13 17:33 /usr/bin/passwd
$ ls -l /usr/bin/man↵                                     ← ไฟล์ที่มีค่าบิต sgid
-r-xr-sr-x    1 root     man         40400 Apr 13 16:46 /usr/bin/man
$ ls -l / | grep tmp↵                                ← ไดเรกทอรีที่มีค่าบิต sticky
drwxrwxrwt   28 root     root         4096 Jul 24 16:25 tmp
```

ถ้ามีไฟล์โปรแกรมที่ตั้งค่าบิต suid ไว้และเจ้าของโปรแกรมนั้นคือ root, เวลาที่ผู้ใช้ทั่วไปรันโปรแกรมนั้นก็จะเหมือนกับ root เป็นผู้รันโปรแกรมนั้น. เนื่องจาก root สามารถลบ, อ่านไฟล์ใด ๆก็ได้ในระบบ, โปรแกรมที่รันนั้นก็จะได้รับสิทธิ์ที่ root สามารถกระทำได้เช่นกัน. ตัวอย่างโปรแกรมที่ตั้งค่าบิต suid ไว้เช่นโปรแกรม `passwd`.

passwd อ้างอิงหน้า 407

ตัวอย่างที่ 3.32: ค่าบิต suid ของคำสั่ง `passwd`

```
$ ls -l /usr/bin/passwd↵
-rwsr-xr-x    1 root     root        26512 Apr 13 17:33 /usr/bin/passwd
$ ls -l /etc/shadow↵
-rw-------    1 root     root          615 Jul  4 21:05 /etc/shadow
```

โปรแกรม `passwd` เป็นโปรแกรมที่ต้องแก้ไขไฟล์ `/etc/shadow` ซึ่งเป็นไฟล์ที่เก็บรหัสผ่านที่ผ่านการเข้ารหัสเรียบร้อยแล้ว. ผู้ที่มีสิทธิ์ในการอ่านและแก้ไขไฟล์นี้คือ root เท่านั้น, ดังนั้นไฟล์โปรแกรม `passwd` จึงมีการตั้งค่าบิต suid ไว้เพื่อให้ใครก็ได้เมื่อสั่งคำสั่ง `passwd` แล้วอ่านและแก้ไขไฟล์ `/etc/shadow` ได้.

โปรแกรมที่มีเจ้าของเป็น root และตั้งค่าบิต suid ไว้ในบางครั้งอาจเป็นตัวก่อให้เกิดอันตรายต่อระบบถ้าหากจู่โจมโดยวิธี buffer overflow. เพราะฉะนั้นจึงควรระวังหากไม่จำเป็นไม่ควรจะใช้หรือเก็บไฟล์โปรแกรมที่ตั้งค่าบิต suid ไว้.

buffer overflow ▶ เป็นอาการที่โปรแกรมไม่สามารถรับข้อมูลเข้าทำให้ล้นพื้นที่ที่รองรับ (buffer). ในกรณีที่โปรแกรมนั้นออกแบบมาไม่ได้, อาจจะทำให้ระบบเกิดความเสียหายหรือเป็นลู่ทางให้ผู้ที่รันโปรแกรมที่มีค่าบิต suid สั่งคำสั่งหรือกระทำการที่ root กระทำได้.

3.12.1 การตั้งค่าบิตพิเศษ

การตั้งค่าบิต suid ให้กับไฟล์ทำได้โดยใช้คำสั่ง chmod โดยมีรูปแบบต่อไปนี้.

```
chmod [ug]+[st] file
```

การระบุโหมดแบบ symbolic, u+s จะหมายถึงการตั้งค่าบิต suid, g+s หมายถึงการตั้งค่าบิต sgid, และ o+t จะหมายถึงการตั้งค่าบิต sticky. การระบุโหมดเป็น +s จะหมายถึงตั้งค่าบิตทั้ง suid และ sgid ทั้งสองตัว.

การระบุโหมดแบบใช้ค่าบิตจะเพิ่มบิต 3 ตัวนำหน้าบิตของสิทธิ์การใช้ไฟล์. เช่นถ้าสิทธิ์การใช้ไฟล์ที่ต้องการตั้งให้ไฟล์เป็น 755 และต้องการให้ไฟล์นี้มีค่าบิต suid ด้วยจะเพิ่มบิตอีกสามตัวข้างหน้าเป็น 4755. กล่าวคือค่า 4000 คือบิต suid, 2000 คือบิต sgid, และ 1000 คือบิต sticky ตามลำดับ.

ดูรูปที่ 3.11 ประกอบเรื่องโหมด.

3.13 การจัดการไฟล์และไดเรกทอรีเบื้องต้น

การจัดการกับไฟล์และไดเรกทอรีเป็นงานที่เกิดขึ้นบ่อยเช่นการลบไฟล์, สร้างไดเรกทอรี, ย้ายไฟล์ไปที่ต่าง ๆ ฯลฯ. งานเหล่านี้ผู้ใช้อาจจะใช้ file browser แบบ GUI เช่น nautilus หรือ konqueror ก็ได้. แต่สำหรับระบบที่ไม่สามารถใช้ X วินโดว์หรือเช่นระบบที่เป็นเซิฟร์เวอร์, การเรียนรู้และใช้คำสั่งสำหรับจัดการไฟล์, ประมวลผลข้อมูลในไฟล์มีความสำคัญอย่างยิ่ง.

คำสั่งที่จัดการไฟล์และไดเรกทอรีส่วนใหญ่จะรับชื่อไฟล์หรือไดเรกอทอรีเป็นอาร์กิวเมนต์และสามารถรับชื่อไฟล์หลาย ๆไฟล์ได้ในคราวเดียวกัน.

3.13.1 การสร้างไฟล์

การสร้างไฟล์ที่ไม่มีข้อมูล (ไฟล์เปล่า) อาจทำได้โดยการรีไดเรกหรือใช้คำสั่ง touch.

ตัวอย่างที่ 3.33: การสร้างไฟล์เปล่าด้วยการรีไดเรกและคำสั่ง touch

```
$ > file1↵
$ touch file2↵
$ ls -l file[12]↵
-rw-r--r--    1 poonlap  users           0 Jul  8 22:45 file1
-rw-r--r--    1 poonlap  users           0 Jul  8 22:46 file2
```

การสร้างไฟล์เท็กซ์ที่มีเนื้อหาสั้น ๆ, ใช้คำสั่ง cat กับการใช้ here document จะเป็นวิธีที่สะดวกที่สุดแต่จะไม่สามารถแก้ไขบรรทัดที่พิมพ์ไปแล้ว.

ตัวอย่างที่ 3.34: การสร้างไฟล์เท็กซ์แบบสั้น ๆด้วย cat

```
$ cat <<EOF > hello.c↵
> #include <stdio.h>
```

```
> main() printf("Hello World!\n");
EOF
```

การสร้างไฟล์เท็กซ์ยาว ๆหรือแก้ไขไฟล์ที่มีอยู่แล้วควรจะใช้บรรณาธิกรณ์สำหรับงานทั่วไปเช่น vi, emacs, gedit, nano ฯลฯ.

บรรณาธิกรณ์สำหรับงานทั่วไปหมายถึงบรรณาธิกรณ์ที่ใช้เขียน, แก้ไขเท็กซ์ไฟล์, ไม่ใช่ word processor เช่น OpenOffice.

สำหรับการสร้างไฟล์ที่มีขนาดไม่เป็นศูนย์ปลอม ๆ, ให้ใช้คำสั่ง dd ตามที่แสดงในตัวอย่างที่ 3.10 (หน้า 100).

3.13.2 การสร้างไดเรกทอรี

mkdir อ้างอิงหน้า 397

การสร้างไดเรกทอรีจะใช้คำสั่ง mkdir. ในกรณีที่ต้องการสร้างไดเรกทอรีซ้อนกันหลายชั้นแต่ยังไม่มีไดเรกทอรีแม่, ให้ใช้ตัวเลือก -p ประกอบเพื่อให้สร้างไดเรกทอรีแม่ที่จำเป็นโดยอัตโนมัติ. ตัวอย่างเช่น

ตัวอย่างที่ 3.35: การสร้างไดเรกทอรีซ้อนกันหลายชั้น.

```
$ mkdir tmp/sub/subsub↵
mkdir: cannot create directory ‘tmp/sub/subsub’: No such file or directory
$ mkdir -p tmp/sub/subsub↵
$ ls -l tmp/sub↵
total 4
drwxr-xr-x    2 poonlap  users        4096 Jul  8 23:13 subsub/
```

3.13.3 การลบไฟล์

rm อ้างอิงหน้า 399

การลบไฟล์จะใช้คำสั่ง rm ซึ่งสามารถลบไฟล์ได้หลายไฟล์พร้อม ๆกัน. ถ้าไฟล์นั้นเป็นไดเรกทอรีและต้องการลบไดเรกทอรีและไฟล์ทั้งหมดที่อยู่ใต้ไดเรกทอรีนั้นให้ใช้ตัวเลือก -r. การลบไฟล์เป็นอันตรายอย่างยิ่งถ้า root เป็นผู้ลบไฟล์ที่จำเป็นต่อระบบ. ดังนั้นจึงควรใช้ตัวเลือก -i เพื่อให้มีการถามย้ำหรือตั้งเป็น alias ไว้.

3.13.4 การลบไดเรกทอรี

rmdir อ้างอิงหน้า 399

ในลินุกซ์จะมีคำสั่งสำหรับลบไดเรกทอรีเฉพาะคือ rmdir แต่จะใช้คำสั่งนี้ได้ก็ต่อเมื่อไฟล์ที่อยู่ใต้ไดเรกทอรีที่ต้องการลบนั้นถูกลบหมดแล้ว. ดังนั้นถ้าต้องการลบไดเรกทอรีที่มีไฟล์อยู่ในนั้นอยู่แล้วใช้คำสั่ง rm -r จะสะดวกกว่าใช้คำสั่ง rmdir.

3.13.5 การย้ายไฟล์, เปลี่ยนชื่อ

mv อ้างอิงหน้า 398

คำสั่ง mv เป็นคำสั่งสำหรับเปลี่ยนชื่อหรือย้ายไฟล์. ตัวอย่างใช้งานแบบง่ายคือมีชื่อไฟล์ที่ต้องการเปลี่ยนชื่อเป็นอาร์กิวเมนต์ตัวที่หนึ่ง, และชื่อไฟล์ใหม่เป็นอาร์กิวเมนต์ตัวที่สอง.

ตัวอย่างที่ 3.36: การเปลี่ยนชื่อหรือย้ายไฟล์.

```
$ ls↵
file
```

```
$ mv -v file newname↵
‘file’ -> ‘newname’
$ ls↵
newname
$ mv -v newname /tmp/newname↵                    ← หรือ mv newname /tmp
‘newname’ -> ‘/tmp/newname’
removed ‘newname’
$ ls -0↵
total 0
$ ls /tmp/newname↵
/tmp/newname
```

จากตัวอย่างจะเห็นว่าการเปลี่ยนชื่อไฟล์ก็คือการย้ายไฟล์แบบหนึ่ง. ส่วนที่เป็นชื่อไฟล์ที่ต้องการเปลี่ยนไปหานั้นจะเป็นชื่อไฟล์แบบ full path, ไดเรกทอรี, หรือ ชื่อไฟล์ก็ได้. ถ้าเป็น full path, ก็จะเป็นการย้ายไฟล์ (ถ้า full path นั้นไม่ใช่ไดเรกทอรีปัจจุบัน). ถ้าเป็นไดเรกทอรี, จะเป็นการย้ายไฟล์ไปไดเรกทอรีที่ต้องการ. ถ้าเป็นชื่อไฟล์เดี่ยว ๆก็จะหมายถึงการเปลี่ยนชื่อไฟล์. เนื่องจากไดเรกทอรีก็คือไฟล์แบบหนึ่งดังนั้นคำสั่ง mv ก็ใช้กับไดเรกทอรีได้ด้วย.

นอกจากจะย้ายหรือเปลี่ยนชื่อไฟล์ชื่อต่อชื่อแล้ว, คำสั่ง mv ยังสามารถย้ายไฟล์ได้มากว่าหนึ่งไฟล์. ในกรณีนี้อาร์กิวเมนต์ตัวสุดท้ายของคำสั่งต้องเป็นชื่อไดเรกทอรีเท่านั้น (เป็นชื่อไฟล์ไม่ได้).

จากการที่เรารู้จักความสัมพันธ์ระหว่างชื่อไฟล์กับไฟล์จริงแล้วก็จะพอจะเดาได้ว่าคำสั่ง mv จะไม่ได้แตะต้องข้อมูลเลย, เป็นการเปลี่ยนชื่อที่ไฟล์หรือไดเรกทอรีที่บันทึกไว้ในไดเรกทอรีหนึ่งไปเขียนใหม่หรือเปลี่ยนชื่อในไดเรกทอรีที่ต้องการย้ายไป. ดังนั้นจึงไม่ใช้เวลามากนักถึงแม้จะเป็นการย้ายไฟล์ที่มีขนาดใหญ่.

การย้ายไฟล์ที่อยู่ต่างพาร์ทิชันหรือระบบไฟล์จะเป็นการเคลื่อนย้ายข้อมูลจริง ๆ(ไฟล์จริง).

คำสั่งที่ใช้เปลี่ยนชื่อไฟล์อีกคำสั่งคือ rename. คำสั่ง rename เหมาะสำหรับการเปลี่ยนชื่อไฟล์หลายไฟล์ด้วยคำสั่งเดียวโดยที่ไฟล์เหล่านี้มีรูปแบบที่ตามตัว.

rename อ้างอิงหน้า 398

ตัวอย่างที่ 3.37: การใช้คำสั่ง rename.

```
$ ls↵
bar1.htm  bar2.htm  foo1.htm  foo2.htm
$ rename .htm .html *↵
$ ls↵
bar1.html  bar2.html  foo1.html  foo2.html
```

จากตัวอย่าง, คำสั่ง rename จะเปลี่ยนชื่อไฟล์ที่ต้องการจาก .htm เป็น .html.

* เป็นไวล์คาร์ดแทนไฟล์ที่อยู่ในไดเรกทอรีปัจจุบันทั้งหมด.

3.13.6 การสำเนาไฟล์

การทำสำเนาไฟล์หรือเรียกง่าย ๆว่าก็อปปี้ทำได้โดยใช้คำสั่ง cp คำสั่ง cp ต้องการอาร์กิวเมนต์เป็นชื่อไฟล์อย่างน้อยสองไฟล์. ไฟล์หลังจากการก็อปปี้ต้องมีชื่อไม่เหมือนกับไฟล์ที่ต้องการก็อปปี้. หรือจะมีชื่อเหมือนกันก็ได้แต่ต้องไม่อยู่ในไดเรกทอรีเดียวกัน. ถ้าต้องการก็อปปี้ไฟล์หลายไฟล์, อาร์กิวเมนต์ตัวสุดท้ายต้องเป็นชื่อไดเรกทอรี. ผลของคำสั่ง cp จะได้ไฟล์ที่มีชื่อเหมือนกันอยู่ในไดเรกทอรีนั้น.

cp อ้างอิงหน้า 395

ตัวอย่างที่ 3.38: การก็อปปี้ไฟล์ด้วยคำสั่ง cp.

```
$ ls↵
foo
$ cp foo bar↵
$ ls↵
bar foo
$ cp -v bar foo /tmp↵
‘bar’ -> ‘/tmp/bar’
‘foo’ -> ‘/tmp/foo’
```

การก็อปปี้ไดเรกทอรีและไฟล์ทั้งหมดที่อยู่ใต้ไดเรกทอรีนั้นให้ใช้คำสั่ง cp กับตัวเลือก -r.

ตัวอย่างที่ 3.39: การก็อปปี้ไดเรกทอรี.

```
$ ls -F↵
dir1/
$ cp -rv dir1 dir2↵
‘dir1’ -> ‘dir2’
‘dir1/foo’ -> ‘dir2/foo’
‘dir1/bar’ -> ‘dir2/bar’
```

ความหมายของการก็อปปี้คือการอ่านข้อมูลจากไฟล์หนึ่งแล้วเขียนลงอีกไฟล์หนึ่ง. ดังนั้นคำสั่ง cat ก็ใช้แทนคำสั่ง cp ได้ด้วยเช่น

ตัวอย่างที่ 3.40: การก็อปปี้ไฟล์ด้วย cat

```
$ cat < foo > bar↵                    ← หรือ cat foo > bar
```

เพราะว่าในโลกของยูนิกซ์ไม่มีการแยกแยะไฟล์ไบนารีกับไฟล์เท็กซ์, ดังนั้นจึงใช้คำสั่ง cat ก็อปปี้ไฟล์ที่ไม่ใช่ไฟล์เท็กซ์ก็ได้.

คำสั่งที่ใช้ก็อปปี้ไฟล์ได้อีกคำสั่งคือ dd. คำสั่ง dd มันจะก็อปปี้*ดีไวซ์ดิบ (raw device)* เช่นการก็อปปี้ /dev/fd0 เก็บไว้ในไฟล์เดียว. ไฟล์ที่ได้มาจากการก็อปปี้เก็บดีไวซ์ดิบแบบนี้บ้างก็เรียกว่า*ไฟล์อิมเมจ (image file)*.

raw device ▶ หมายถึงไฟล์ดีไวซ์. ไฟล์ที่ยังไม่ได้ mount เข้าโครงสร้างไฟล์ไดเรกทอรี. ตัวอย่าง raw device ได้แก่ /dev/hda, /dev/cdrom เป็นต้น.

ตัวอย่างที่ 3.41: การใช้ dd ก็อปปี้ดีไวซ์ดิบเก็บในไฟล์เดียว.

```
$ dd if=/dev/fd0 of=floppy.img↵
```

คำสั่ง cp หรือ cat สามารถใช้ก็อปปี้ดีไวซ์ดิบได้เช่นเดียวกับคำสั่ง dd เพราะคำสั่งเหล่านี้ไฟล์ดีไวซ์ก็เหมือนกับไฟล์ที่อยู่ในระบบไฟล์ทั่วไปคือเป็นไฟล์. เพียงแต่ว่าไฟล์ดีไวซ์ไม่ได้ mount ปะติดกับโครงสร้างไฟล์ไดเรกทอรี.

ตัวอย่างที่ 3.42: การใช้ cp ก็อปปี้ดีไวซ์ดิบเก็บในไฟล์เดียว.

```
$ cp /dev/fd0 floppy.img↵
```

dd อ้างอิงหน้า 385

ความแตกต่างระหว่าง dd กับ cp คือคำสั่ง dd สามารถระบุขนาดของบล็อก (block size)

ที่ต้องการอ่านหรือเขียนได้. การปรับค่าขนาดของบล็อกให้เหมาะสมกับดีไวซ์จะทำให้การเขียนอ่านเร็วขึ้น.

3.13.7 การหาไฟล์

การหาไฟล์ด้วยบรรทัดคำสั่งจะแบ่งได้เป็นสองวิธีคือ

1. การหาไฟล์โดยอาศัยฐานข้อมูลที่เตรียมไว้ล่วงหน้า. ฐานข้อมูลนี้เป็นข้อมูลที่เก็บที่อยู่ของไฟล์และชื่อไฟล์, และมีการปรับปรุงข้อมูลให้ทันสมัยเป็นระยะที่กำหนดไว้โดยอัตโนมัติ. คำสั่งหาไฟล์ที่ทำงานในลักษณะนี้ได้แก่ `slocate` หรือ `locate`.
2. การหาไฟล์โดยการไล่หาไปเรื่อย ๆจากไดเรกทอรีที่กำหนดเป็นจุดเริ่มต้นหา. วิธีนี้จะเป็นหาไฟล์จริง ๆโดยไม่ใช้ฐานข้อมูลที่เตรียมไว้, ดังนั้นไม่จำเป็นต้องมีการสร้างฐานข้อมูลล่วงหน้าก่อนการหาไฟล์. แต่มีข้อเสียที่จะใช้เวลานานในการหาไฟล์ในบางกรณี. คำสั่งหาไฟล์ที่ทำงานในลักษณะนี้ได้แก่คำสั่ง `find`.

ถ้าจะเรียกการหาไฟล์ให้ถูกต้องควรจะใช้คำว่าการหาชื่อไฟล์. ในที่นี้ขอใช้คำว่าการหาไฟล์เพื่อความสะดวก.

คำสั่งที่ใช้สร้างฐานข้อมูลสำหรับคำสั่ง `locate` คือคำสั่ง `updatedb`.

คำสั่ง `locate`, `slocate`

`locate` เป็นโปรแกรมคำสั่งที่หาไฟล์ที่โดยอาศัยฐานข้อมูลของไฟล์ในระบบที่เตรียมไว้ก่อนแล้ว. ส่วนคำสั่ง `slocate` เป็นโปรแกรม `locate` ที่ได้รับการปรับปรุงด้านความปลอดภัย (security enhanced) ให้ดีขึ้นและทำหน้าที่เหมือน `locate`. ดังนั้นในลินุกซ์ดิสทริบิวชันทั่ว ๆไปจึงใช้ `slocate` และมี `locate` เป็นซอฟต์ลิงค์ชี้ไปที่ `slocate` อีกที.

การไฟล์ด้วยคำสั่ง `locate` ทำได้โดยระบุชื่อไฟล์หรือส่วนของชื่อไฟล์เป็นอาร์กิวเมนต์ของคำสั่ง. สมมติว่าเราต้องการหาดูว่ามีไฟล์โปรแกรมอะไรบ้างที่เกี่ยวกับการวาดกราฟก็อาจจะใช้ `locate` หาไฟล์ที่มีคำว่า “plot”.

ต่อไปนี้ถ้ามีการอ้างถึง `locate` จะหมายถึง `slocate` ด้วยไปในตัว.

locate อ้างอิงหน้า 396

ตัวอย่างที่ 3.43: การหาไฟล์ด้วย `locate`

```
$ locate plot↵
/var/cache/edb/dep/app-emacs/gnuplot-mode-0.6.0
/var/cache/edb/dep/app-sci/kmatplot-0.4-r1
/var/cache/edb/dep/app-sci/kmatplot-0.4-r2
...
```

จากตัวอย่างจะเห็นว่าไฟล์ที่มีคำว่า “plot” อยู่ด้วยมีมากเกินไปและไฟล์บางไฟล์ไม่ใช่ข้อมูลที่ต้องการ. เราสามารถใช้คำสั่ง `grep` กรองข้อมูลที่ต้องการได้เช่น เอาผลลัพธ์ที่มีคำว่า “bin” มาแสดงอย่างเดียว. สาเหตุที่เอาผลลัพธ์ที่มีคำว่า “bin” มาแสดงเพราะไฟล์โปรแกรมมักจะอยู่ในไดเรกทอรี `/usr/bin` หรือ `/usr/local/bin`.

ตัวอย่างที่ 3.44: การใช้ `locate` และ `grep` หาไฟล์ที่ต้องการ.

```
$ locate plot | grep bin↵
/usr/bin/pbmtoplot
/usr/bin/plot
/usr/bin/tek2plot
```

```
/usr/bin/plotfont
/usr/bin/pic2plot
/usr/bin/gnuplot
/usr/share/doc/gnuplot-4.0/demo/binary.dem
/usr/share/doc/gnuplot-4.0/demo/binary1
/usr/share/doc/gnuplot-4.0/demo/binary2
/usr/share/doc/gnuplot-4.0/demo/binary3
/usr/kde/3.2/bin/kmplot
```

การใช้ `locate` ร่วมกับ `grep` ก็จะแสดงไฟล์โปรแกรมที่มีคำว่า “plot” ให้. ส่วนโปรแกรมเหล่านี้มีวิธีใช้อย่างไรต้องไปดูคู่มือเพิ่มเติม, หรือลองใช้ด้วยตัวเอง.

นอกจากการหาไฟล์แบบง่าย ๆตามที่แสดงไปแล้ว, คำสั่ง `locate` ยังมีตัวเลือก `-i` (insensitive case) ไว้หาไฟล์โดยไม่แยกแยะตัวอักษรว่าจะเป็นตัวใหญ่หรือตัวเล็ก. ตัวเลือก `-r` ให้ผู้ใช้ระบุคำที่ต้องการหาในรูปแบบของ regular expression ได้ด้วย. ตัวอย่างเช่นจะหาไฟล์ที่ลงท้ายด้วย “xml” ที่มีอยู่ในระบบสามารถใช้ `locate` กับ regular expression ได้ตามตัวอย่าง.

รายละเอียดเรื่อง regular expression ดูที่หน้า 164.

regular expression ▶ กลุ่มอักขระที่ใช้แสดงเป็นตัวแทนของคำโดยที่กลุ่มอักขระที่ใช้แสดง regular expression นี้มีไวย์กรณ์ที่แน่นอน. regular expression มีประโยชน์ในการเขียนนิยามของคำที่ต้องการค้นหา. regular expression มักจะใช้กับโปรแกรม `grep`, `sed`, `perl` เป็นต้น. ตัวอย่างเช่น “`A.*$`” หมายถึงคำที่ขึ้นต้นด้วย A จะมีอักขระใด ๆหรือไม่มี (ใช้ . แทนอักขระใด ๆ) มากกว่าหนึ่งตัวจนสุดบรรทัด. regular expression นี้แทนตัว A, A3, Asdf ได้เป็นต้น.

ตัวอย่างที่ 3.45: การใช้ `locate` กับ regular expression.

```
$ locate -r 'xml$'↵
/etc/xml
/etc/bonobo-activation/bonobo-activation-config.xml
/etc/gconf/gconf.xml.defaults/%gconf.xml
/etc/gconf/gconf.xml.defaults/system/%gconf.xml
...
```

คำสั่ง find

คำสั่ง `find` เป็นคำสั่งที่หาไฟล์ใต้ไดเรกทอรี่ที่กำหนดโดยระบุเงื่อนไขที่ต้องการ. การใช้คำสั่ง `find` จะซับซ้อนกว่าการใช้คำสั่ง `locate` เพราะสามารถตั้งเงื่อนไขได้หลายอย่าง, ไม่จำเป็นต้องเป็นชื่อไฟล์เช่น หาไฟล์ตามเงื่อนไขสิทธิ์การใช้ไฟล์, เวลาที่มีการใช้ไฟล์นั้นครั้งสุดท้าย ฯลฯ. นอกจากนั้นยังสามารถกระทำการกับไฟล์ที่ตรงตามเงื่อนไขตามที่ต้องการด้วย.

find อ้างอิงหน้า 396

การหาไฟล์ตามเงื่อนไขแบบง่าย ๆได้แก่การหาไฟล์ตามชื่อที่ต้องการ. ในตัวอย่างต่อไปนี้เป็นหารไฟล์ที่ลงท้ายด้วย “`.o`” (ตัวเลือก `-name`) ที่อยู่ใต้ไดเรกทอรีปัจจุบันและให้แสดงชื่อไฟล์ (relative path) ทางหน้าจอ (ตัวเลือก `-print`).

ตัวอย่างที่ 3.46: การใช้ `find` หาไฟล์ตามชื่อที่ต้องการ.

```
$ find . -name "*.o" -print↵
./CVS/software/cttex/map.o
./CVS/software/xiterm+thai/src/command.o
./CVS/software/xiterm+thai/src/debug.o
...
```

สมมติว่าเราต้องการจะลบไฟล์เหล่านี้ด้วยก็ใช้คำสั่ง `-exec` ให้คำสั่ง `find` ดังนี้.

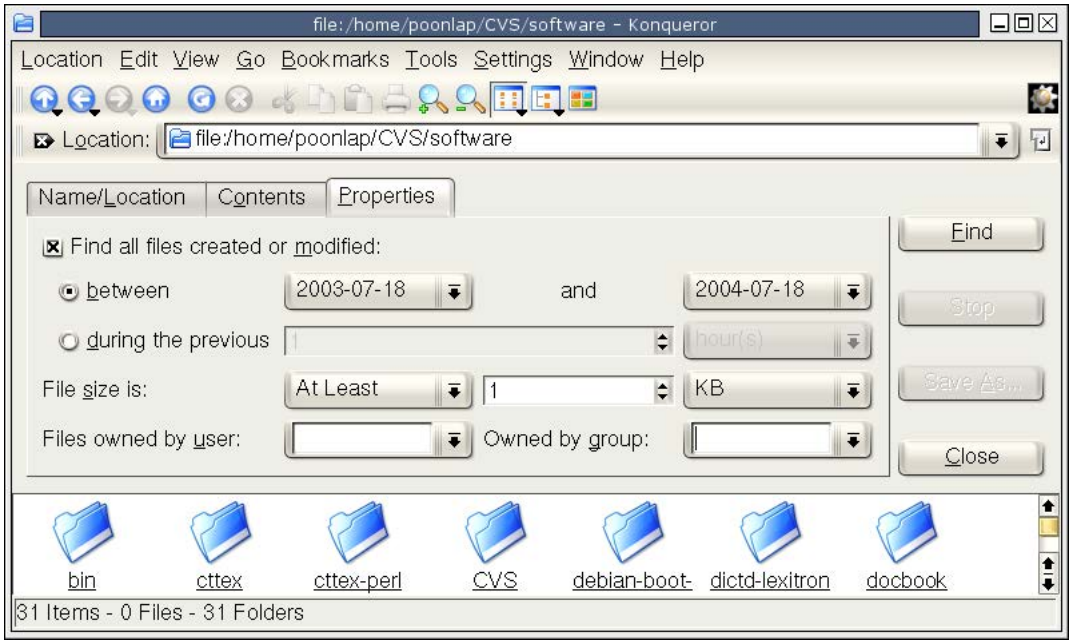
ตัวอย่างที่ 3.47: ใช้คำสั่ง `find` ลบไฟล์ที่ต้องการ.

```
$ find . -name "*.o" -exec rm -v '{}' \;↵
removed `./CVS/software/cttex/map.o'
removed `./CVS/software/xiterm+thai/src/command.o'
removed `./CVS/software/xiterm+thai/src/debug.o'
...
```

อาร์กิวเมนต์ที่อยู่หลัง `-exec` คือคำสั่งที่ต้องการทำกับไฟล์ที่ค้นหาเจอ. `'{}'` เป็นตัวแทนชื่อไฟล์ที่หาเจอและสาเหตุที่ต้องใช้เครื่องหมาย quote ครอบเพราะเครื่องหมายปีกกามีความหมายพิเศษต่อเชลล์. ตัวอักษร "`\;`" เป็นตัวบอกให้คำสั่ง `find` รู้ว่าจบคำสั่งที่ต้องการใช้เมื่อหาไฟล์นั้นเจอ. สาเหตุที่ต้องใส่เครื่องหมาย backslash หน้า ; เพราะ ; มีความหมายพิเศษสำหรับเชลล์.

จะใช้เครื่องหมาย quote ก็ได้เช่น `find . -name "*.o" -exec rm -v '{}' ';'`.

สำหรับวิธีการหาไฟล์จากเงื่อนไขอื่น ๆเช่น สิทธิ์การใช้ไฟล์, เวลาที่แก้ไขไฟล์ ฯลฯ สามารถอ่านได้จาก `man find`. เนื่องจากคำสั่ง `find` เป็นคำสั่งที่ซับซ้อนและใช้ยาก, สำหรับผู้ที่ใช้เดกสทอปอาจจะใช้ไฟล์บราวเซอร์ `konqueror` ในการหาไฟล์ที่ต้องการ. เมนูการหาไฟล์ของ `konqueror` มีตัวเลือกต่าง ๆเหมือนกับที่คำสั่ง `find` ทำได้.



รูปที่ 3.12: เมนูหาไฟล์ใน konqueror.

3.13.8 ดูข้อมูลในไฟล์

การกระทำกับไฟล์อย่างหนึ่งที่เกิดขึ้นบ่อยได้แก่การดูข้อมูลในไฟล์. เราอาจจะใช้บรรณาธิกรณ์ที่ถนัดเช่น `vi`, `emacs`, `gedit` ฯลฯ เปิดไฟล์ดูข้อมูลข้างในก็ได้. แต่การเรียกใช้โปรแกรมเหล่านี้เป็นการใช้ทรัพยากรณ์ (resource) ของคอมพิวเตอร์โดยใช่เหตุ. ถ้าต้องการดูเนื้อหาในไฟล์ควรใช้คำสั่งที่ใช้ทรัพยากรณ์น้อยเช่น `cat`, `less`. และ

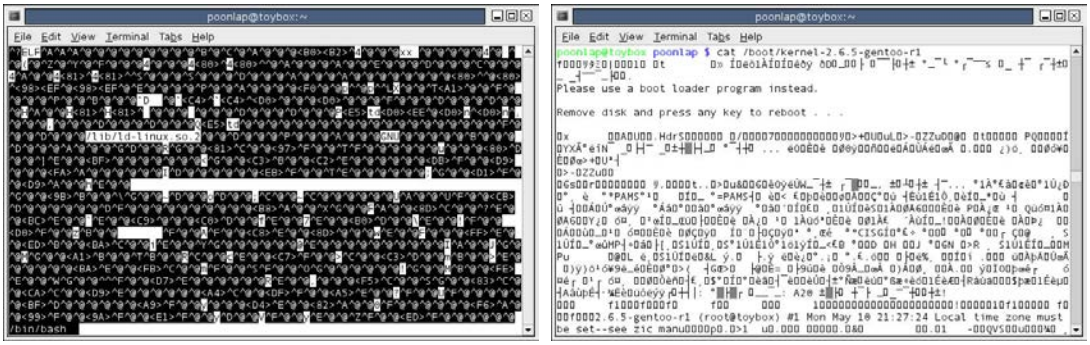
โปรแกรมเหล่านี้สามารใช้ได้ในเทอร์มินอล, ทำงานเฉพาะทางจึงทำให้การทำงานเร็วกว่าการใช้บรรณาธิกรณ์ทั่วไป.

โดยทั่วไปเราจะใช้คำสั่ง `less` เพื่อดูข้อมูลไฟล์เท็กซ์. การดูเนื้อหาในไฟล์อย่างเดียวไม่เพียงพอ, บ่อยครั้งที่เราต้องการหาคำที่อยู่ในไฟล์นั้นด้วย. ในกรณีนี้ keybinding "/" ซึ่งจะเป็นการหาคำไปข้างหน้า. ใช้คีย์ "?" เพื่อหาคำย้อมหลัง. ถ้าต้องการแก้ไขไฟล์นั้นก็ใช้คีย์ `v` เพื่อเรียกบรรณาธิกรณ์ที่ตั้งค่าไว้ในตัวแปรสภาพแวดล้อมมาเป็นโปรแกรมแก้ไขไฟล์นั้น.

key binding ของ `less` ดูได้ที่หน้า 68.

การดูไฟล์ไบนารี

เราสามารถดูไฟล์อะไรก็ได้ด้วยคำสั่งเช่น `less`, `cat`, `head`ฯลฯ. ถ้าในไฟล์นั้นเป็นไฟล์ไบนารีก็อาจจะแสดงอักขระที่อ่านไม่ออกทางเทอร์มินอลในรูปที่ 3.13.



รูปที่ 3.13: การดูไฟล์ไบนารีทางเทอร์มินอล

ในกรณีที่ใช้โปรแกรมที่ไม่มีการกรองอักขระควบคุมที่แสดงทางเทอร์มินอลเช่น `cat`, `head` ฯลฯ จะทำให้เทอร์มินอลแสดงผลหรือทำงานผิดปรกติหลังจากดูไฟล์ไบนารี. ในกรณีให้พิมพ์คำสั่ง `reset` ถึงแม้จะอ่านไม่ออกก็ตามแล้วกดคีย์ Enter ตาม. คำสั่ง `reset` จะทำให้เทอร์มินอลกลับอยู่ในสภาพปรกติ.

`reset`
รีเซ็ตให้เทอร์มินอลอยู่ในสภาพเริ่มต้นปรกติ.

โอกาสที่จะดูเนื้อหาของไฟล์ไบนารีคงมีไม่บ่อยนักแต่ถ้ามีความจำเป็น, คำสั่ง `od` ช่วยเราได้.

`od` อ้างอิงหน้า 414

ลินุกซ์ยังมีคำสั่งที่คล้ายกับคำสั่ง `od` อื่น ๆอีกเช่น `hexdump`

ตัวอย่างที่ 3.48: ใช้ `od` ดูไฟล์ไบนารี.

```
$ od /bin/bash | head -n 3↵
0000000 042577 043114 000401 000001 000000 000000 000000 000000
0000020 000002 000003 000001 000000 131260 004005 000064 000000
0000040 074170 000011 000000 000000 000064 000040 000010 000050
```

ในแต่ละบรรทัดจะเป็นข้อมูลในไฟล์ที่แสดงในรูปของค่าฐานแปด. ถ้าต้องการให้แสดงข้อมูลเป็นฐานสิบหกให้ใช้ตัวเลือก `-h` (hexadecimal). อย่างไรก็ตามการข้อมูลที่แสดงนั้นไม่สื่อความหมายเท่าที่ควรเพราะเป็นตัวค่าของข้อมูลล้วน ๆ. ถ้าใช้ตัวเลือก `-c` (character), คำสั่ง `od` จะพยายามแสดงข้อมูลที่เป็นอักขระ ASCII ทางหน้าจอ.

ตัวอย่างที่ 3.49: ใช้ od กับตัวเลือก -c ดูไฟล์ไบนารี.

```
$ od -c /bin/bash | head -n 3↵
0000000 177   E   L   F 001 001 001  \0  \0  \0  \0  \0  \0  \0  \0
0000020 002  \0 003  \0 001  \0  \0  \0 260 262 005  \b   4  \0  \0  \0
0000040   x   x  \t  \0  \0  \0  \0  \0   4  \0      \0  \b  \0   (  \0
```

จะเห็นข้อมูลบางส่วนในไบนารีไฟล์มีข้อมูลเท็กซ์ผสมอยู่ด้วย.

คำสั่ง less สามารถเปิดดูไฟล์ไบนารีได้เช่นเดียวกับการดูไฟล์เท็กซ์. จากรูปที่ 3.13 เป็นการใช้ less เปิดดูไฟล์ /bin/bash ซึ่งถ้าเปรียบเทียบกับตัวอย่างที่ 3.49 จะเห็นว่าคำสั่ง less จะแสดงส่วนที่เป็นอักขระ ASCII ให้. ถ้ามีการตั้งตัวแปรสภาพแวดล้อม LESSOPEN ไว้, less จะใช้โปรแกรมที่ระบุไว้ในตัวแปรนั้นเป็นโปรแกรมประมวลผลไฟล์ที่ต้องการดูข้อมูลก่อนที่จะใช้ less เปิดอ่านไฟล์นั้น. ตัวอย่างเช่นถ้าไฟล์ที่ต้องการเปิดอ่านด้วย less เป็นไฟล์ที่บีบอัดด้วย gzip (.gz), โปรแกรมที่ระบุไว้ในตัวแปรสภาพแวดล้อมก็จะใช้คำสั่ง gzip ขยายไฟล์นั้นก่อนแล้วส่งข้อมูลไปให้ less. ถ้าไม่ต้องการใช้โปรแกรมที่ระบุไว้ในตัวแปรสภาพแวดล้อม LESSOPEN ก็ใช้ใช้ตัวเลือก -L.

less อ้างอิงหน้า 410

คำสั่งที่ใช้ดูข้อมูลเท็กซ์ที่อยู่ในไฟล์ไบนารีที่มีประโยชน์อีกคำสั่งคือ strings. คำสั่ง strings จะสกัดส่วนที่เป็นข้อมูลเท็กซ์ออกมาจากไฟล์ไบนารี. ถ้าไม่มีการระบุตัวเลือก, คำสั่ง strings จะถือว่าข้อมูลเท็กซ์ที่ต้องการคืออักขระ ASCII ที่เรียงติดกันอย่างน้อย 4 ตัว.

strings อ้างอิงหน้า 415

ตัวอย่างที่ 3.50: การสกัดส่วนที่เป็นข้อมูลเท็กซ์จากไฟล์ไบนารี.

```
$ strings /bin/bash | head -n 3↵
/lib/ld-linux.so.2
libdl.so.2
dlerror
```

สาเหตุที่คำสั่ง strings ไม่แสดงคำว่า “ELF” ทั้งๆที่คำสั่ง od เห็นว่ามีคำนี้เพราะว่าคำสั่ง strings จะสกัดข้อมูลเท็กซ์จากไฟล์ไบนารีที่เป็นไฟล์โปรแกรม, และจะสกัดข้อมูลเท็กซ์ที่อยู่ในช่วงของข้อมูลโปรแกรม (data sectoin) เท่านั้น. ถ้าต้องการจะให้คำสั่ง strings สกัดข้อมูลเท็กซ์จากทุกส่วนในไฟล์โปรแกรม, ให้ใช้ตัวเลือก -a (all) ประกอบ.

ตัวอย่างที่ 3.51: การใช้ strings กับตัวเลือก -a.

```
$ strings -a -3 /bin/bash | head -n 3↵
ELF
xx
‘D
```

3.14 สรุปท้ายบท

- พาร์ทิชันคือการแบ่งฮาร์ดดิสก์เชิงตรรกะยังไม่สามารถใช้งานได้จนกว่าจะสร้างระบบไฟล์ในพาร์ทิชันนั้น.

- ระบบไฟล์เชิงตรรกะในลินุกซ์จะมีโครงสร้างเป็นแผนภาพต้นไม้โดยมีไดเรกทอรีรูทเป็นจุดเริ่มต้นและขยายเป็นไดเรกทอรีย่อยต่อไป.
- การ mount คือการนำพาร์ทิชันมาปะติดกับไดเรกทอรีในระบบไฟล์.
- ไฟล์คือกลุ่มข้อมูลหน่วยเป็นไบต์ที่เรียงเป็นลำดับต่อกันไปเป็นสาย.
- ทุกอย่างในระบบปฏิบัติการยูนิกซ์และลินุกซ์สามารถแสดงได้ด้วยไฟล์.
- ไดเรกทอรีคือไฟล์พิเศษที่มีข้อมูลเกี่ยวกับชื่อไฟล์และที่อยู่ในหน่วยความจำถาวรของไฟล์นั้น.
- คำว่า “ไฟล์” มีความได้สองอย่างคือไฟล์จริง (ข้อมูล) หรือชื่อไฟล์ (ชื่ออ้างอิงข้อมูล).

บทที่ 4
โปรแกรมคำสั่งพื้นฐาน

"Write programs that do one thing and do it well.
Write programs to work together.
Write programs to handle text streams,
because that is a universal interface."
Doug McIlroy

ในลินุกซ์จะเป็นเพียงแค่ระบบปฏิบัติที่ไร้ค่าถ้าไม่โปรแกรมต่าง ๆใช้ร่วมกัน. โปรแกรมต่าง ๆที่ว่านี้ได้แก่โปรแกรมเสรีแบบบรรทัดคำสั่ง (command line) และโปรแกรมแบบ GUI.

ตั้งแต่เริ่มต้นของระบบปฏิบัติการยูนิกซ์ซึ่งเป็นเวลานานกว่า 30 ปีแล้วโปรแกรมคำสั่งต่าง ๆที่ใช้ในตอนนั้นก็ยังใช้กันอยู่ในตอนนี้. โปรแกรมคำสั่งชื่อเดียวกันในสมัยในอาจจะมีส่วนที่เหมือนกันและแตกต่างกันกับโปรแกรมคำสั่งสมัยนี้ที่มีความสามารถเพิ่มเติมต่าง ๆมากขึ้น, ทำงานได้ดีขึ้น. แต่สิ่งที่ยังคงไว้อยู่คือโปรแกรมส่วนใหญ่ใช้ประมวลผลข้อมูลเท็กซ์, แสดงผลให้ผู้ใช้เป็นเท็กซ์ และใช้อินเทอร์เฟสแบบบรรทัดคำสั่ง. แน่นอนว่าการแสดงผลและรับข้อมูลเป็นเท็กซ์มีขีดจำกัด, และบางกรณีไม่สะดวกต่อการใช้จึงมีการสร้างอินเทอร์เฟสแบบหน้าต่างที่เรียกว่า X วินโดว์เสริมขึ้นจนเป็นอินเทอร์เฟสกึ่งมาตรฐานที่ใช้กันในลินุกซ์.

ในบทนี้จะแนะนำโปรแกรมต่าง ๆที่มีในระบบลินุกซ์ซึ่งส่วนมากจะเป็นโปรแกรมบรรทัดคำสั่ง, และโปรแกรม GUI เป็นกรณีไป.

จากที่ได้แนะนำไปแล้วในบทที่ 1 ว่า Free Software Foundation หรือ GNU เป็นองค์กรณ์ที่มีบทบาทสำคัญอย่างยิ่งต่อลินุกซ์เพราะเป็นองค์กรณ์ที่สร้างหรือเป็นผู้สนับสนุนโครงการโปรแกรมพื้นฐานหลักที่ใช้กันในระบบปฏิบัติการยูนิกซ์ให้มีใช้ในลินุกซ์อย่างเสรี. แพ็กเกจที่สำคัญและเป็นพื้นฐานที่ลินุกซ์ทุกค่ายใช้ได้แก่ coreutils ซึ่งในแพ็กเกจนี้ยังแบ่งย่อยออกเป็น fileutils, shellutils และ textutils.

4.1 แพ็กเกจ fileutils

แพ็กเกจ fileutils เป็นแพ็กเกจที่รวมโปรแกรมบรรทัดคำสั่งที่เกี่ยวกับการจัดการไฟล์ ซึ่งในบทที่ผ่านมาได้แนะนำคำสั่งที่ใช้บ่อยและคำสั่งที่จำเป็นไปแล้วพอสมควร. ในตารางที่ 4.1

ตารางที่ 4.1: โปรแกรมคำสั่งในแพ็กเกจ fileutils.

คำสั่ง	คำอธิบาย	อ้างอิง
`chgrp`	เปลี่ยนกลุ่มของไฟล์.	หน้า 394
`chown`	เปลี่ยนเจ้าของไฟล์.	หน้า 405
`chmod`	เปลี่ยนสิทธิ์การใช้ไฟล์.	หน้า 394
`cp`	ก็อปปี้ไฟล์.	หน้า 395
`dd`	ก็อปปี้และแปลงข้อมูล.	หน้า 385
`df`	แสดงพื้นที่การใช้งานฮาร์ดดิสก์.	หน้า 386
`dir`	แสดงรายการไฟล์.	
`dircolors`	ตั้งค่าเริ่มต้นเพื่อให้คำสั่ง `ls` แสดงสีต่าง ๆได้.	
`du`	แสดงพื้นที่การใช้งานฮาร์ดดิสก์.	หน้า 395
`install`	ก็อปปี้ไฟล์และตั้งค่าสิทธิ์การใช้ไฟล์.	หน้า 389
`ln`	สร้างลิงค์.	หน้า 396
`ls`	แสดงรายการไฟล์.	หน้า 397
`mkdir`	สร้างไดเรกทอรี.	หน้า 397
`mkfifo`	สร้างไฟล์ fifo.	หน้า 397
`mknod`	สร้างไฟล์พิเศษ.	
`mv`	ย้ายไฟล์.	หน้า 398
`rm`	ลบไฟล์.	หน้า 399
`rmdir`	ลบไดเรกทอรีว่าง.	หน้า 399
`shred`	ทำลายข้อมูลในไฟล์.	หน้า 399
`sync`	เขียนข้อมูลที่อยู่ใน buffer ลงดิสก์.	หน้า 401
`touch`	เปลี่ยน timestamp ของไฟล์.	หน้า 400
`vdir`	แสดงรายการไฟล์.	

4.1.1 สำรวจพื้นที่ฮาร์ดดิสก์

ถ้าอ่านจากคำอธิบายจากตารางที่ 4.1 จะเห็นว่า `du` และ `df` ทำงานคล้าย ๆกัน. คำสั่ง `df` จะแสดงพื้นที่ฮาร์ดดิสก์แบ่งตามดีไวซ์ต่าง ๆ, ส่วนคำสั่ง `du` จะแสดงพื้นที่ใช้งานของไดเรกทอรี่ที่ต้องการ.

ตัวอย่างที่ 4.1: สำรวจพื้นที่ฮาร์ดดิสก์ด้วย `df` และ `du`.

```
$ df -h↵
Filesystem            Size  Used Avail Use% Mounted on
/dev/hdb1              19G   13G  4.6G  74% /
/dev/hdb2              18G   13G  4.7G  73% /home
none                  442M     0  442M   0% /dev/shm
/dev/cdrom            3.2G  3.2G     0 100% /mnt/cdrom
$ du -sh $HOME↵
13G     /home/poonlap
```

4.1.2 ก็อปปี้ไฟล์

คำสั่งสำหรับก็อปปี้ไฟล์ที่ใช้ทั่วไปคือ cp. ส่วนคำสั่ง install ก็ทำงานเหมือนกับคำสั่ง cp และวิธีใช้งานเบื้องต้นก็เหมือนกันคือก็อปปี้ไฟล์จากที่หนึ่งไปอีกที่หนึ่ง. คำสั่ง install มักใช้ในการติดตั้งโปรแกรมในระบบ.

การติดตั้งโปรแกรมแบบง่ายที่สุดในระบบยูนิกซ์และลินุกซ์คือการก็อปปี้ไฟล์ไปไว้ในไดเรกทอรีเดียวกันกับไดเรกทอรีที่กำหนดในตัวแปรสภาพแวดล้อม PATH เช่น /usr/bin. ดิสทริบิวชันแต่ละค่ายอาจจะมีโปรแกรมสำหรับติดตั้งโปรแกรมที่มาเป็นแพ็กเกจต่าง ๆกัน เช่น rpm, dpkg, emerge ฯลฯ. โปรแกรมติดตั้งแพ็กเกจเหล่านี้ก็ใช้การก็อปปี้ไฟล์โปรแกรมและไฟล์ที่เกี่ยวข้องอื่น ๆไปไว้ที่ต้องการเท่านั้นเอง, และมีการสร้างฐานข้อมูลสำหรับควบคุมและติดตามการติดตั้งแพ็กเกจต่าง ๆ. การก็อปปี้ไฟล์จะใช้คำสั่ง cp ก็ได้แต่ถ้ามีการตั้งค่าต่าง ๆเช่นสิทธิ์การใช้ไฟล์, เจ้าของไฟล์ด้วยก็ต้องมาสั่งคำสั่ง chmod, chown ภายหลัง. ดังนั้นจึงมีคำสั่ง install เพื่อตั้งค่าต่าง ๆเหล่านี้พร้อมกับการก็อปปี้.

สำหรับโปรแกรมที่เป็นไฟล์ไบนารี, ไฟล์นั้นอาจจะมีข้อมูลที่ไม่จำเป็นในการใช้งาน เช่นข้อมูลที่เกี่ยวกับการ debugในไฟล์. ข้อมูลเหล่านี้สร้างขึ้นตอนที่คอมไพล์โปรแกรมนั้น ๆ. คำสั่ง install ยังช่วยเอาข้อมูลที่ไม่จำเป็นเหล่านี้ออกด้วยเวลาติดตั้งด้วยตัวเลือก -s. ถ้าไม่ใช้ตัวเลือกนี้, ก็อาจจะใช้คำสั่ง strip เอาข้อมูลที่ไม่จำเป็นออกไปต่างหาก.

debug ▶
การกำจัดข้อมผิดพลาดของโปรแกรม. ข้อผิดพลาดนี้มักเป็นข้อผิดพลาดที่ไม่คาดคิดจนกระทั่งใช้งานจริงที่เรียกว่า runtime error.

install อ้างอิงหน้า 389

การเอาของมูลที่ไม่จำเป็นสำหรับการรันโปรแกรมนี้เรียกว่า strip.

สมมติว่าเรามีไฟล์โปรแกรมไบนารีที่คอมไพล์เองและต้องการก็อปปี้เป็นชื่อ myprogram ไว้ที่ $HOME/bin เพื่อใช้งาน. เราสามารถใช้คำสั่ง install แทนคำสั่ง cp ได้ตามตัวอย่างต่อไปนี้.

ตัวอย่างที่ 4.2: การใช้คำสั่ง `install`.

```
$ ls -lF a.out↵
-rwxr-xr-x    1 poonlap  users       14596 Sep  3 00:37 a.out*
$ install -vs -m 750 a.out ~/bin/myprogram↵
'a.out' -> '/home/poonlap/bin/myprogram'
$ ls -lF a.out ~/bin/myprogram↵
-rwxr-x---    1 poonlap  users        3116 Sep  3 00:41 /home/poonlap/bin/myprogram*
-rwxr-xr-x    1 poonlap  users       14596 Sep  3 00:38 a.out*
```

ตัวเลือก -v เป็นตัวเลือกที่ใช้ได้กับคำสั่ง cp คือแสดงสถานะการทำงาน. ตัวเลือก -m เป็นการระบุโหมดของไฟล์ที่ต้องการติดตั้ง. ให้สังเกตขนาดของไฟล์และโหมดของไฟล์

ก่อนและหลังการก็อปปี้.

4.1.3 เปลี่ยนเจ้าของไฟล์และกลุ่ม

การเปลี่ยนเจ้าของไฟล์จะใช้คำสั่ง `chown`. ส่วนการเปลี่ยนกลุ่มของไฟล์จะใช้คำสั่ง `chgrp`. ถ้าต้องการเปลี่ยนทั้งเจ้าของไฟล์และกลุ่มไฟล์, สามารถใช้คำสั่ง `chown` เปลี่ยนค่าที่ต้องการในรวดเดียวโดยระบุเจ้าของไฟล์และชื่อกลุ่มโดยใช้เครื่องหมาย colon คั่นเช่นถ้าต้องการให้ไดเรกทอรี `/var/www` เป็นที่เก็บไฟล์สำหรับเว็บเซิฟร์เวอร์ให้มีเจ้าของเป็น apache และอยู่ในกลุ่ม users ให้ทำดังนี้.

ตัวอย่างที่ 4.3: การเปลี่ยนเจ้าของไฟล์และกลุ่มในคราวเดียวกัน.

```
# chown -R apache:users /var/www↵
```

4.1.4 สร้างไฟล์พิเศษ

เราได้รู้จักกับคำสั่งที่สร้างไฟล์พิเศษจากบทที่ผ่านมาเช่นถ้าต้องการสร้างไฟล์ fifo จะใช้คำสั่ง `mkfifo`. ถ้าต้องการสร้างลิงค์จะใช้คำสั่ง `ln`. คำสั่ง `mknod` เป็นอีกคำสั่งหนึ่งที่ใช้สร้างไฟล์พิเศษได้แก่ ไฟล์ fifo, ไฟล์ดีไวซ์แบบ character และแบบ block. ผู้ใช้ทั่วไปสามารถสร้างไฟล์ fifo ได้ด้วยคำสั่ง `mkfifo` หรือ `mknod` ก็ได้. ส่วนการสร้างไฟล์ดีไวซ์ทำได้โดย root เท่านั้น.

mknod อ้างอิงหน้า 398

4.1.5 ทำลายข้อมูล

ในระบบปฏิบัติการตระกูลยูนิกซ์, การลบไฟล์ด้วยคำสั่ง `rm` นั้นเป็นเพียงแค่การลบฮาร์ดลิงค์ออกจากไดเรกทอรีเท่านั้น. เนื้อหาในไฟล์นั้นยังคงในที่ใดที่หนึ่งในฮาร์ดดิสก์จนกว่าพื้นที่นั้นจะถูกใช้เก็บข้อมูลใหม่. และในตอนนั้นข้อมูลตรงนั้นก็จะถูกแทนที่ด้วยข้อมูลใหม่. คำสั่ง `shred` เป็นคำสั่งทำลายข้อมูลในไฟล์โดยที่เขียนข้อมูลปลอม ๆซ้ำกันหลายครั้งลงในไฟล์ที่ต้องการ. ถ้าใช้ตัวเลือก `-u` ประกอบด้วยจะทำการลบไฟล์นั้นหลังจากที่ทำลายข้อมูลเรียบร้อยแล้ว.

shred อ้างอิงหน้า 399

`shred` มาจากคำว่า shredder คือเครื่องทำลายเอกสาร.

4.1.6 sync

การอ่านหรือเขียนข้อมูลลงดิกส์จะช้ากว่าการอ่านหรือเขียนข้อมูลลงหน่วยความจำ. ดังนั้นระบบปฏิบัติการสมัยใหม่จะใช้หน่วยความจำบางส่วนเก็บข้อมูลชั่วคราวเวลาเขียนข้อมูลลงไฟล์หรืออ่านข้อมูลลงไฟล์. เราเรียกหน่วยความจำที่ใช้ในลักษณะนี้ว่า *buffer cache*. การทำงานของ buffer cache จะเห็นชัดมากในกรณีที่ใช้ฟล็อปปี้. บางครั้งเราเขียนข้อมูลลงในฟล็อปปี้, ระบบจะไม่บันทึกข้อมูลที่เขียนลงในแผ่นฟล็อปปี้ทันทีแต่จะเก็บไว้ใน buffer cache. เวลาสั่งคำสั่ง `umount` แล้วจึงจะย้ายข้อมูลที่อยู่ในหน่วยความจำที่ต้องบันทึกลงในแผ่นฟล็อปปี้. การใช้คำสั่ง `sync` เป็นการบังคับเขียนข้อมูลที่อยู่ในหน่วยความจำลงในดิสก์. โดยปรกติผู้ใช้ไม่จำเป็นต้องใช้คำสั่งนี้เพราะลินุกซ์จะจัดการให้โดยอัตโนมัติอยู่แล้ว.

4.2 แพ็กเกจ shellutils

จากบทที่ผ่านมาเราทราบแล้วว่าคำสั่งที่ใช้ในเชลล์แบ่งเป็นคำสั่งประกอบภายใน (builtin command) และโปรแกรมคำสั่งที่มีอยู่ในระบบ. แพ็กเกจ shellutils เป็นแพ็กเกจที่รวมโปรแกรมคำสั่งที่ช่วยเสริมการทำงานของเชลล์ให้ใช้งานสะดวกยิ่งขึ้น, และคำสั่งหลายตัวมักใช้ประกอบในการเขียนเชลล์สคริปต์.

โปรแกรมคำสั่งหลายตัวในแพ็กเกจนี้ถูกรวมอยู่ในคำสั่งประกอบภายในเชลล์ bash ด้วยเช่น `test`, `echo` ฯลฯ. หมายความว่าถ้าเราสั่งคำสั่ง `echo`, เชลล์จะเป็นตัวดำเนินงานเอง, ไม่ได้ใช้ไฟล์โปรแกรม `/bin/echo`. เดิมทีคำสั่งเหล่านี้ไม่ได้เป็นคำสั่งประกอบภายในเชลล์แต่มีการปรับปรุงเชลล์ bash ให้เอาคำสั่งเหล่านี้เป็นคำสั่งประกอบภายในด้วย. ในแพ็กเกจ shellutils ยังคงต้องเก็บโปรแกรมคำสั่งเหล่านี้ไว้เพื่อให้เข้ากันได้กับเชลล์อื่นและเชลล์ bash รุ่นเก่า ๆ.

ตารางที่ 4.2: โปรแกรมคำสั่งในแพ็กเกจ shellutils.

คำสั่ง	คำอธิบาย	อ้างอิง
`[`	ตรวจสอบชนิดของไฟล์และเปรียบเทียบค่า.	หน้า 416
`basename`	ลบส่วนที่อยู่หน้าชื่อไฟล์ออก และให้ผลลัพธ์เฉพาะชื่อไฟล์อย่างเดียว.	หน้า 419
`chroot`	เปลี่ยนไดเรกทอรีรูท.	
`date`	แสดงหรือตั้งเวลา, วันที่.	หน้า 411
`dirname`	ตัดส่วนที่เป็นชื่อไฟล์ออกและแสดงเฉพาะส่วนที่อยู่หน้าชื่อไฟล์.	หน้า 420
`echo`	แสดงบรรทัดข้อความ.	หน้า 420
`env`	แสดงหรือแก้ไขตัวแปรสภาพแวดล้อม.	หน้า 410
`expr`	ตีความนิพจน์ (expression).	หน้า 421
`factor`	หาตัวประกอบของจำนวนเต็ม.	หน้า 421
`false`	คืนค่าสถานะการจบการทำงานไม่สมบูรณ์ (false).	หน้า 421
`groups`	แสดงกลุ่มที่ผู้ใช้อยู่.	หน้า 413
`hostid`	แสดงค่าเฉพาะของโฮส, เครื่องคอมพิวเตอร์.	
`hostname`	แสดงชื่อโฮส.	หน้า 405
`id`	แสดง uid และ gid.	หน้า 413
`logname`	แสดงชื่อล็อกอิน.	
`nice`	แก้ไขลำดับความสำคัญของการทำงานโปรเซส (scheduling).	หน้า ??
`nohup`	อนุญาติให้โปรแกรมที่ระบุทำงานต่อถึงแม้ผู้ใช้จะล็อกเอาท์ไปแล้วก็ตาม.	หน้า 423
`pathchk`	ตรวจสอบว่าชื่อไฟล์สามารถใช้ในระบบอื่น ๆ (พอร์ต) ได้หรือไม่.	

pinky	โปรแกรม finger.	
printenv	แสดงตัวแปรสภาพแวดล้อมและค่าของตัวแปร.	หน้า 414
printf	จัดรูปข้อมูลแล้วแสดงผลข้อมูลนั้น.	
pwd	แสดงไดเรกทอรีที่ทำงานอยู่.	หน้า 398
seq	แสดงลำดับจำนวนที่ระบุ.	หน้า 424
sleep	หยุดการทำงานชั่วคราวตามระยะเวลาที่กำหนด.	
stty	แสดงและตั้งค่าของเทอร์มินอล.	หน้า 415
su	เปลี่ยนผู้ใช้เป็นผู้ใช้คนอื่น.	หน้า 407
tee	ส่งข้อมูลออกไปบันทึกในไฟล์ที่ต้องการ.	
test	ทดสอบนิพจน์.	หน้า 416
true	คืนค่าสถานะการจบการทำงานสมบูรณ์ (true).	หน้า 424
tty	แสดงชื่อเทอร์มินอล.	หน้า 417
uname	แสดงข้อมูลของระบบปฏิบัติการ.	หน้า 425
users	แสดงชื่อผู้ใช้ที่ล็อกอินอยู่ในระบบ.	หน้า 417
who	แสดงชื่อผู้ใช้ที่ล็อกอินอยู่ในระบบ.	หน้า 419
whoami	แสดง id ที่ให้ผลจริง.	หน้า 419
yes	แสดงสายอักขระที่ต้องการไปเรื่อย ๆ.	

4.2.1 ตรวจสอบไฟล์และค่าต่าง ๆ

test อ้างอิงหน้า 416

คำสั่งที่ใช้ตรวจสอบไฟล์และค่าต่าง ๆได้แก่คำสั่ง `test` และคำสั่ง `[`. คำสั่งทั้งสองตัวเหมือนกันเพียงแต่มีไวยกรณ์ต่างกันตรงที่คำสั่ง `[` ต้องการเครื่องหมาย `]` ปิดท้ายเสมอ. โดยปรกติคำสั่ง `test` มักจะใช้ร่วมกับไวยกรณ์ `if` ของเชลล์เพื่อตรวจสอบไฟล์หรือค่าต่าง ๆ.

ตัวอย่างต่อไปนี้เป็นตัวอย่างการตรวจสอบไฟล์ว่ามีจริงหรือไม่ด้วยตัวเลือก `-e`.

ตัวอย่างที่ 4.4: ตรวจสอบว่ามีไฟล์หรือไม่.

```
$ test -e /etc/passwd↵
$ [ -e /etc/passwd ]↵
```

expression ▶
นิพจน์. ประโยคที่แสดงความหมายอย่างใดอย่างหนึ่ง.

เราจะเรียก "`-e /etc/passwd`" ว่าเป็น*นิพจน์ (expression)* และคำสั่ง `test` จะตรวจสอบว่านิพจน์นั้นจริงหรือเท็จ. คำสั่ง `test` จะไม่แสดงผลออกทางหน้าจอไม่ว่าจะจริงหรือเท็จ, แต่จะใช้ตัวแปรสถานะจบการทำงาน "`$?`" บอกว่าจริงหรือเท็จ. ถ้าการตรวจสอบเป็นจริงจะมีค่าเป็น 0, ถ้าเป็นเท็จจะมีค่าเป็นค่าอื่น ๆที่ไม่ใช่ 0.

ตัวอย่างต่อไปนี้เป็นการตรวจสอบใช้คำสั่ง `test` ร่วมกับ `if`.

ตัวอย่างที่ 4.5: การใช้คำสั่ง `test` ร่วมกับ `if`.

```
$ if↵
> [ -f /etc/shadow -a `whoami` == root ]↵
> then↵
> echo You are root and file /etc/shadow existed.↵
```

```
> else↵
> echo You are not root or file /etc/shadow does not exist.↵
> fi↵
You are not root or file /etc/shadow does not exist.
```

คำสั่ง `if` จะตรวจสอบตัวแปรสถานะจบการทำงานถ้าเป็นจริงก็จะกระทำส่วนที่ต่อจาก `then` ถ้าเป็นเท็จก็จะกระทำส่วนที่ต่อจาก `else`. จากตัวอย่างมีการเชื่อมนิพจน์ด้วย `-a` ซึ่งนิพจน์ทั้งสองที่ถูกเชื่อมต้องเป็นจริงแล้วนิพจน์โดยรวมจะเป็นจริง.

คำสั่ง `test` สามารถตรวจสอบประเภทไฟล์, สิทธิ์การใช้ไฟล์เบื้องต้น, เปรียบเทียบค่าสายอักขระ, และเปรียบเทียบจำนวนเต็ม. รายละเอียดเหล่านี้ดูได้จากสรุปการใช้คำสั่งหน้า 416.

4.2.2 สกัดส่วนของชื่อไฟล์

เวลาใช้เชลล์จะมีบางกรณีที่เราต้องการสกัดส่วนของชื่อไฟล์มาเพื่อกระทำการอย่างใดอย่างหนึ่ง. คำสั่งที่ช่วยแยกชื่อไฟล์หรือไดเรกทอรีจริง ๆ ออกจากส่วนเป็นพาท (path) ของชื่อไฟล์คือคำสั่ง `basename` และ `dirname`. คำสั่งสองคำสั่งนี้ไม่ได้ทำอะไรมากเพียงแต่สกัดส่วนที่ต้องการให้เท่านั้น. มีประโยชน์เมื่อใช้เขียนเชลล์สคริปต์.

basename อ้างอิงหน้า 419
dirname อ้างอิงหน้า 420

ตัวอย่างที่ 4.6: การสกัดชื่อไฟล์หรือไดเรกทอรีจากชื่อไฟล์แบบ full path.

```
$ basename /usr/local/bin/test.pl↵
test.pl
$ basename /usr/local/bin/test.pl .pl↵      ← .pl คือส่วนที่ต้องการตัดออกจากชื่อไฟล์.
test
$ dirname /usr/local/bin/test.pl↵
/usr/local/bin
$ basename /home/↵
home                                        ← ผลลัพธ์จะตัดเครื่องหมาย \ ให้ด้วย.
$ dirname /home/↵
/
```

จากตัวอย่างจะเห็นว่าคำสั่ง `basename` สามารถสกัดเอาส่วนขยายชื่อไฟล์ออกได้ด้วยถ้าระบุตามหลังชื่อไฟล์. คำสั่งที่เกี่ยวกับชื่อไฟล์อีกตัวคือ `pathchk` ใช้สำหรับตรวจดูว่าชื่อไฟล์นั้นมีรูปแบบถูกต้องหรือไม่เช่นจำนวนอักษรที่ใช้ในชื่อไฟล์เกินขีดจำกัดหรือไม่เป็นต้น. คำสั่ง `pathchk` เป็นคำสั่งที่ไม่ค่อยใช้.

4.2.3 ถูกผิด

คำสั่ง `true` และ `false` ไม่ได้ทำอะไรเป็นพิเศษเพียงแต่ตั้งค่าสถานะจบการทำงานเป็น 0 (`true`) หรือ 1 `false` ตามต้องการ. มักจะใช้ในการเขียนเชลล์สคริปต์ใช้ในการตั้งค่าตัวแปรสถานะจบการทำงาน. ถ้าการทำงานจบบริบูรณ์ถูกต้องก็สั่งคำสั่ง `true` เป็นคำสั่งสุดท้าย. ถ้ามีข้อผิดพลาดก็สั่งคำสั่ง `false`.

true อ้างอิงหน้า 424
false อ้างอิงหน้า 421

4.2.4 คำสั่งเกี่ยวกับข้อมูลของระบบ

ในระบบปฏิบัติการลินุกซ์จะมีคำสั่งที่แสดงข้อมูลเบื้องต้นเกี่ยวกับระบบและผู้ใช้ในระบบหลายตัว. คำสั่งที่บอกว่าผู้ใช้ตอนนี้คือใครได้แก่ `id`, `whoami` และ `logname`. คำสั่ง `whoami` จะเหมือนกับการสั่งคำสั่ง "`id -un`". คำสั่ง `id` จะให้ข้อมูลเกี่ยวกับกลุ่มด้วย, ส่วนคำสั่ง `logname` จะแสดงเฉพาะชื่อล็อกอินเท่านั้น. ถ้าต้องการดูแค่กลุ่มที่ตัวเองอยู่ก็ให้ใช้คำสั่ง `groups`.

groups อ้างอิงหน้า 413

คำสั่ง `hostid` เป็นคำสั่งที่แสดงหมายเลขเฉพาะประจำเครื่อง. หมายเลขนี้จะไม่ซ้ำกับเครื่องอื่น ๆและสามารถใช้ในการควบคุมการติดตั้งซอฟต์แวร์ที่ต้องการยึดติดกับเครื่อง. ซอฟต์แวร์บางตัวสามารถตรวจสอบ `hostid` เพื่อดูว่าเครื่องนั้นมีสิทธิ์รันซอฟต์แวร์นั้นได้หรือไม่. แต่ปัจจุบันไม่ค่อยใช้วิธีการตรวจสอบ `hostid` แล้วเพราะค่าเหล่าสามารถปลอมกันได้. คำสั่งที่เกี่ยวกับคอมพิวเตอร์อีกตัวคือ `hostname` ใช้แสดงชื่อเครื่องหรือตั้งชื่อเครื่องคอมพิวเตอร์. ชื่อโฮสดังกล่าวจะใช้กับโปรแกรมที่เกี่ยวกับเน็ตเวิร์กต่าง ๆเพื่อระบุตัวเครื่อง.

hostname อ้างอิงหน้า 405

คำสั่งที่ใช้แสดงข้อมูลเบื้องต้นเกี่ยวกับเคอร์เนลได้แก่ `uname`. โดยปรกติจะใช้กับตัวเลือก `-a` เพื่อแสดงข้อมูลทั้งหมดเช่น รุ่นของเคอร์เนล, สถาปัตยกรรมของเครื่อง, ข้อมูลเบื้องต้นของหน่วยประมวลผล ฯลฯ. ข้อมูลเหล่านี้มีความสำคัญเวลาใช้โปรแกรมหรือไดร์เวอร์ที่ขึ้นอยู่กับรุ่นของเคอร์เนล, เป็นวิธีที่ตรวจสอบว่าลินุกซ์ที่เราใช้อยู่นั้นใช้เคอร์เนลรุ่นไหนอยู่.

uname อ้างอิงหน้า 425

4.2.5 สำรวจผู้ใช้ที่ล็อกอิน

คำสั่งที่ใช้ตรวจสอบดูว่าขณะนี้มีใครบ้างที่ล็อกอินอยู่ในระบบได้แก่คำสั่ง `users` และ `who`. คำสั่ง `users` จะอาศัยข้อมูลจากไฟล์ `/var/run/utmp` โดยปริยายหรืออาศัยข้อมูลจากไฟล์ `/var/log/wtmp`. ไฟล์ `utmp` เป็นไฟล์ไบนารีที่เก็บข้อมูลผู้ใช้ที่กำลังอยู่ในระบบ. ส่วนไฟล์ `wtmp` เป็นไฟล์เก็บข้อมูลล็อกอินและล็อกเอาท์ของผู้ใช้, เวลาที่ระบบเริ่มทำงาน (startup) และจบการทำงาน (shutdown) ด้วย. คำสั่ง `users` จะแสดงแค่ชื่อล็อกอินของผู้ที่ล็อกอินอยู่ในระบบไม่มีข้อมูลอื่นเพิ่มเติม. เมื่อเทียบกับคำสั่ง `who` จะให้ข้อมูลมากกว่าคำสั่ง `users`.

who อ้างอิงหน้า 419

ตัวอย่างที่ 4.7: สำรวจผู้ใช้ที่ล็อกอินอยู่ในระบบด้วย `who`.

```
$ who -Hu↵
NAME     LINE         TIME         IDLE          PID COMMENT
poonlap  :0           Sep 13 19:53   ?          8395
poonlap  pts/2        Sep 13 19:54   .         10444 (:0.0)
poonlap  pts/3        Sep 13 21:05 00:24       10444 (:0.0)
poonlap  pts/4        Sep 13 21:05   .         10444 (:0.0)
root     pts/5        Sep 13 21:29   .         26565 (10.0.0.196)
```

เราสามารถรู้ได้ว่าผู้ใช้ล็อกอินเข้ามาทางไหนได้จากคอลัมน์ LINE และ COMMENT. คอลัมน์ LINE จะแสดงเทอร์มินอลที่ผู้ใช้ใช้อยู่. `pts` หมายถึง pseudo-terminal คือเทอร์มินอลเสมือนที่ใช้อยู่ในเทอร์มินอลเอมิวเลเตอร์. `:0` แสดงว่าผู้ใช้ล็อกอินผ่านทาง X

อ่าน `man pts` เกี่ยวกับรายละเอียดของ `pts`.

Display Manager. ในคอลัมน์ COMMENT จะแสดง IP address ถ้าผู้ใช้ล็อกอินผ่านทางเน็ตเวิร์ก. ในคอลัมน์ TIME จะแสดงวันและเวลาที่ผู้ใช้ล็อกอิน. IDLE จะแสดงเวลาที่ว่างเฉย. เวลา IDLE นี้บอกให้เราทราบเวลาที่ผู้ใช้นั้นล็อกอินอยู่แต่ไม่ได้กระทำการใด ๆ ในระบบ. PID จะแสดงหมายเลขโปรเซสของโปรแกรมที่ใช้หลังล็อกอิน.

คำสั่งที่คล้ายกับ who คือ w แต่จะแสดงรายละเอียดเน้นการใช้งานของหน่วยประมวลผลและอ่านเข้าใจง่ายกว่าคำสั่ง who.

w อ้างอิงหน้า 417

ตัวอย่างที่ 4.8: ใช้คำสั่ง w ดูผู้ใช้ที่ล็อกอินและโปรแกรมที่ใช้.

```
$ w↵
 21:53:17 up  2:02,  5 users,  load average: 0.15, 0.28, 0.32
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
poonlap  :0        19:53   ?xdm?  15:28   0.60s gnome-session
poonlap  pts/2     19:54    0.00s  2:11   0.00s w
poonlap  pts/3     21:05    2:35   0.00s  0.00s -bash
poonlap  pts/4     21:05   23:04  32.25s  0.67s xmms
root     pts/5     21:29   23:53   0.00s  0.00s -bash
```

ในบรรทัดแรกของคำสั่ง w จะแสดงเวลาปัจจุบัน, ระยะเวลาที่ระบบทำงานตั้งแต่เปิดเครื่อง, จำนวนผู้ใช้ที่ล็อกอินในระะบ, และ*โหลดโดยเฉลี่ย (load average)* ในช่วงเวลา 1, 5 และ 15 นาทีที่ผ่านมา. บรรทัดถัดจากนั้นจะแสดงชื่อผู้ใช้ที่ล็อกอินและรายละเอียดต่าง ๆคล้ายกับคำสั่ง who. JCPU คือระยะเวลาหน่วยประมวลใช้ไปกับโปรแกรมที่กระทำอยู่ในเทอร์มินอลนั้นไม่ว่าจะเป็นโปรเซส foreground หรือ background. PCPU คือระยะเวลาหน่วยประมวลใช้ไปกับโปรแกรมที่กระทำอยู่ขณะนั้นในเทอร์มินอล, ได้แก่โปรแกรมที่แสดงในคอลัมน์ WHAT.

load average ▶
โหลดโดยเฉลี่ย. ค่าเฉลี่ยของจำนวนโปรเซสที่ active อยู่ในระบบ. ค่านี้จะแสดงให้ทราบว่าระบบยุ่งหรือไม่. ค่านี้ไม่จำเป็นต้องสัมพันธ์กับเปอร์เซ็นต์การใช้งานของหน่วยประมวลผล.

บรรทัดแรกของคำสั่ง w เป็นข้อมูลสถิติการทำงานระบบโดยทั่วไปซึ่งสามารถแสดงได้ด้วยคำสั่งต่างหากคือ uptime ซึ่งจะให้ผลเหมือนกัน.

uptime อ้างอิงหน้า 404

คำสั่งที่เกี่ยวข้องแต่ไม่ได้อยู่ในแพ็กเกจ shellutils อีกคำสั่งคือ last.

last อ้างอิงหน้า 413

ตัวอย่างที่ 4.9: ดูสถิติการล็อกอินด้วย last.

```
$ last -n 5↵
poonlap  pts/3        yahoobb219037040 Mon Sep 13 22:28   still logged in
xxxxx    pts/3        yahoobb219215060 Mon Sep 13 22:02 - 22:04  (00:01)
xxxxx    pts/1        yahoobb219215060 Mon Sep 13 21:59   still logged in
poonlap  pts/1        yahoobb219037040 Sun Sep 12 15:10 - 15:17  (00:07)
xxxxx    pts/1        yahoobb219215060 Sun Sep 12 12:15 - 12:21  (00:06)

wtmp begins Wed Sep  1 13:46:59 2004
```

คำสั่ง last จะแสดงเทอร์มินอลที่ใช้, IP address หรือชื่อโฮสที่ใช้ล็อกอินเข้ามา (ถ้ามี), และข้อมูลเวลาล็อกอินและล็อกเอาท์ของผู้ใช้แต่ละคน. เนื่องจากคำสั่ง last ใช้ข้อมูลจากไฟล์ /var/log/wtmp ดังนั้นจะมีข้อมูลเวลาเกี่ยวกับการรีบูต, เปิดเครื่อง, และปิดเครื่องด้วยถ้าใช้ร่วมกับตัวเลือก -x.

คำสั่งนี้ยังสามารถดูผู้ใช้ที่ใช้เทอร์มินอลที่ระบุได้ด้วย. ตัวอย่างเช่นเรามีเซิฟร์เวอร์อยู่ในห้องที่แยกต่างหากคนโดยทั่วไปไม่สามารถใช้คอนโซลได้ง่าย ๆ. ถ้าต้องการดูชื่อผู้ใช้

ชื่อเทอร์มินอลของคอนโซลในบางระบบอาจจะไม่ใช่ vc/1 เช่นใน Debian จะเป็น tty1.

บางระบบไม่มีไฟล์ btmp อาจจะใช้ touch สร้างไฟล์นี้ให้ทำงานได้.

"b" ย่อมาจาก bad.

และเวลาของคนที่ใช้คอนโซลก็สามารถใช้คำสั่ง "last vc/1" ระบุเทอร์มินอลเสมือน (คอนโซล) อันแรก (Ctrl+Alt+F1) ดูคนที่ใช้คอนโซลและเวลาได้.

ในกรณีที่ผู้ใช้ล็อกอินผิดพลาดเช่นพิมพ์ชื่อหรือรหัสผ่านผิด. ถ้าในระบบมีไฟล์ /var/log/btmp, ก็จะบันทึกความผิดพลาดที่เกี่ยวกับการล็อกอินไว้ด้วยในไฟล์นี้. ไฟล์ btmp เป็นไฟล์ประเภทเดียวกับ utmp และ wtmp คือเก็บข้อมูลเวลาเกี่ยวกับการล็อกอินล็อกเอาท์. ในระบบจะมีคำสั่ง lastb สำหรับดูว่าใครล็อกอินพลาดบ้าง.

ตัวอย่างที่ 4.10: ดูบันทึกล็อกอินที่ผิดพลาดด้วย lastb.

```
$ lastb vc/1↵
UNKNOWN  vc/1                      Mon Sep 13 22:50 - 22:50  (00:00)
nobody   vc/1                      Mon Sep 13 22:50 - 22:50  (00:00)
root     vc/1                      Mon Sep 13 22:50 - 22:50  (00:00)
poonlap  vc/1                      Mon Sep 13 22:49 - 22:49  (00:00)

btmp begins Mon Sep 13 22:49:35 2004
```

จากตัวอย่างคำสั่ง lastb, เราพอจะคาดเดาได้ว่ามีคนพยายามล็อกอินจากคอนโซลแต่ล็อกอินไม่ได้. ไม่ว่าจะเป็นรหัสผ่านไม่ถูกต้อง, หรือใช้ชื่อล็อกอินที่ระบบไม่รู้จัก (UNKNOWN).

4.2.6 เทอร์มินอล

เทอร์มินอลเป็นอินเทอร์เฟสที่ใช้ติดต่อกับเชลล์, รับข้อมูลนำเข้ามาตรฐาน stdin จากคีย์บอร์ดและส่งข้อมูลมาตรฐาน stdout ออกทางหน้าจอ. ในสมัยก่อนเทอร์มินอลเป็น serial device ที่ต่อกับเครื่องคอมพิวเตอร์. ในปัจจุบันเทอร์มินอลที่ใช้กันทั่วไปจะเป็นเทอร์มินอลเอมิวเลเตอร์คือโปรแกรมที่จำลองการทำงานของเทอร์มินอลเช่น xterm เป็นต้น.

ตัวแปรสภาพแวดล้อม TERM เป็นตัวแปรที่กำหนดประเภทของเทอร์มินอลที่ใช้. ถ้าลองแสดงค่าของตัวแปรสภาพแวดล้อม TERM ด้วยคำสั่ง echo $TERM ในเทอร์มินอลเอมิวเลเตอร์ xterm ก็จะได้ผลเป็น xterm. ในลินุกซ์จะมีฐานข้อมูลของเทอร์มินอลเรียกว่า terminfo ซึ่งจะเก็บข้อมูลความสามารถต่าง ๆของเทอร์มินอลเช่น จำนวนคอลัมน์ในแต่ละบรรทัด, จำนวนสีที่ใช้ได้ในเทอร์มินอล เป็นต้น. ฐานข้อมูลเหล่านี้ช่วยให้โปรแกรมที่ใช้เทอร์มินอลแสดงผลได้ถูกต้อง. โดยปรกติแล้วค่าของตัวแปรสภาพแวดล้อม TERM จะถูกตั้งค่าโดยอัตโนมัติ. ถ้าการแสดงผลของโปรแกรมที่ใช้เทอร์มินอลผิดปรกติก็อาจจะเป็นเพราะค่าตัวแปรสภาพแวดล้อม TERM ตั้งไว้ไม่ตรงกับประเภทของเทอร์มินอลที่ใช้.

ดูรายละเอียดได้จาก terminfo(5)

โปรแกรมที่แสดงผลและมีความสัมพันธ์กับเทอร์มินอลมักใช้ไลบรารี ncurse ประกอบการทำงานเช่น โปรแกรม vim เป็นบรรณาธิกรณ์ที่ใช้ได้ในเทอร์มินอล.

tty อ้างอิงหน้า 417

เทอร์มินอลสามารถแสดงได้ด้วยไฟล์เช่นเดียวกับอุปกรณ์ต่าง ๆในระบบปฏิบัติการและคำสั่งที่ใช้ตรวจสอบว่าเทอร์มินอลที่ใช้อยู่คือไฟล์อะไรได้แก่คำสั่ง tty.

ตัวอย่างที่ 4.11: ตรวจสอบไฟล์ดีไวซ์ของเทอร์มินอลที่ใช้.

```
$ tty↵
/dev/pts/5
$ ls -l /dev/pts/5↵
crw--w----    1 poonlap  tty      136,   5 Sep 18 20:19 /dev/pts/5
```

ในตัวอย่างเป็นการแสดงไฟล์เทอร์มินอลของ gnome-terminal. เวลาเรียกใช้เทอร์มินอลเอมิวเลเตอร์, ระบบจะสร้างไฟล์สำหรับเทอร์มินอลนั้นในไดเรอทอรี /dev/pts ให้โดยอัตโนมัติ.

เนื่องจากเทอร์มินอลสามารถได้ด้วยไฟล์ในระบบ. ดังนั้นเราสามารถเขียนหรืออ่านข้อมูลด้วยไฟล์นั้นได้ด้วย. สมมติว่าเราเปิดเทอร์มินอลไว้สองอัน. อันหนึ่งคือ pts/0 และอีกอันหนึ่งคือ pts/5. เราสามารถส่งข้อมูลจากเทอร์มินอล pts/0 ไปหาเทอร์มินอล pts/5 ได้ด้วยการ echo แล้วรีไดเรกไปที่ไฟล์ /dev/pts/5 ตามตัวอย่าง.

ตัวอย่างที่ 4.12: การรีไดเรกข้อมูลไปหาเทอร์มินอลอื่น ๆ.

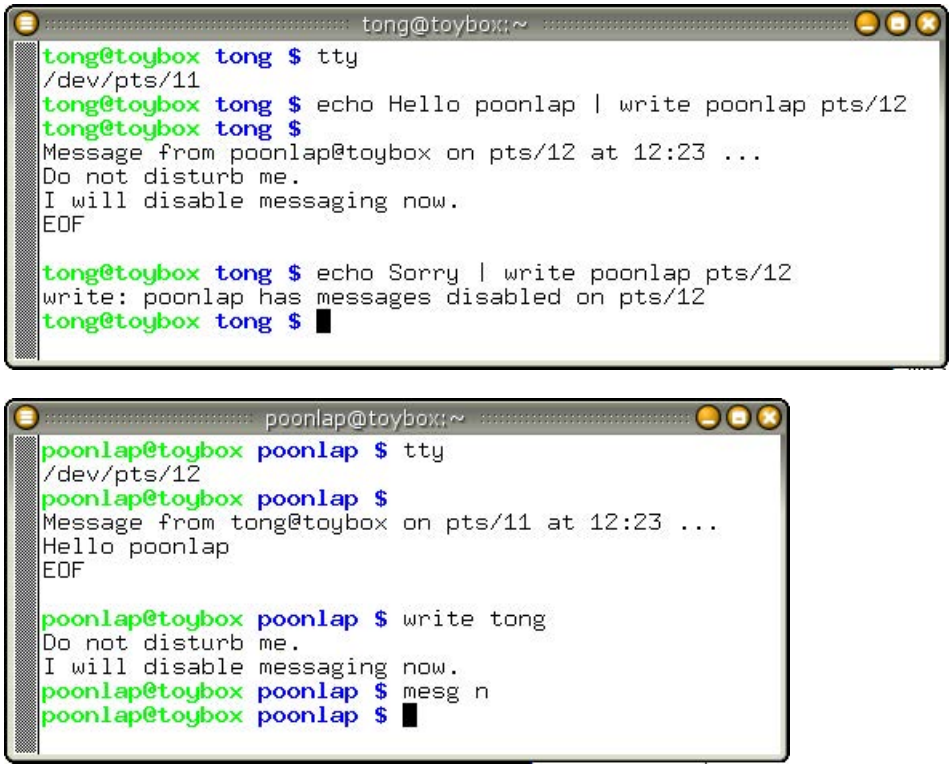
```
$ echo 'Hello world' > /dev/pts/5↵
```

คำว่า "Hello world" จะปรากฏที่หน้าจอของเทอร์มินอล pts/5. จากกรณีสมมติในตัวอย่างเจ้าของเทอร์มินอลทั้งสองเป็นผู้ใช้เดียวกันจึงทำแบบนี้ได้. ถ้าเจ้าของเทอร์มินอลเป็นคนละคนกันอาจจะใช้คำสั่ง chmod แก้สิทธิ์การใช้ไฟล์เทอร์มินอลก่อน.

ในการใช้งานจริงแล้ว, ลินุกซ์จะมีคำสั่ง write เพื่อเขียนส่งข้อความไปหาผู้ใช้ที่ล็อกอินอยู่. ถ้าผู้ใช้นั้นใช้เทอร์มินอลหลายตัวพร้อม ๆก็สามารถระบุเทอร์มินอลที่ต้องการส่งข้อความได้ด้วย. ในกรณีที่ไม่ต้องการให้คนอื่นรบกวนเขียนข้อความเข้ามาในเทอร์มินอลก็ให้ใช้คำสั่ง mesg อนุญาตหรือไม่อนุญาตการส่งข้อมูลผ่านเทอร์มินอล.

write อ้างอิงหน้า 425

mesg อ้างอิงหน้า 422



รูปที่ 4.1: ใช้ write ส่งข้อความระหว่างผู้ใช้ในระบบ.

ในบางกรณีอาจจะมีความจำเป็นต้องส่งข้อความผ่านเทอร์มินอลหาผู้ใช้ทุกคนในระบบที่ใช้เทอร์มินอลอยู่, ให้ใช้คำสั่ง wall ส่งข้อมูลให้ผู้ใช้ที่ใช้เทอร์มินอลทุกตัวในระบบ.

wall อ้างอิงหน้า 408

ปัจจุบัน *Instant Messenger* เช่น MSN messenger, Yahoo messenger เป็นเครื่องมือที่นิยมใช้สื่อสารส่งข้อความผ่านทางเน็ตเวิร์ก. ในระบบปฏิบัติการแบบยูนิกซ์ช่วงแรกก็มีโปรแกรมแบบนี้เช่นเดียวกันใช้คุยกันผ่านทางเน็ตเวิร์กได้แก่ `talk` หรือ `phone`. โปรแกรม `talk` จะใช้เทอร์มินอลในการสื่อสารง่ายต่อการใช้. ในปัจจุบันไม่นิยมใช้กันเท่าไหร่นักและในลินุกซ์ก็มีโปรแกรม Instant Messenger ที่เข้ากันได้กับ MSN messenger หรือ Yahoo messenger ด้วย.

ปรับแต่งเทอร์มินอล

ถ้าผู้อ่านเคยใช้เทอร์มินอลหลาย ๆแบบหลาย ๆที่อาจจะสังเกตุเห็นว่าพฤติกรรมการแสดงผลหรือคีย์บอร์ดที่ใช้บางทีจะไม่เหมือนกัน. สาเหตุหนึ่งคือประเภทของเทอร์มินอลไม่เหมือนกันซึ่งสามารถแก้ได้ด้วยการตั้งค่าตัวแปรสภาพแวดล้อม `TERM` ให้ถูกต้อง. อีกสาเหตุหนึ่งคือการตั้งค่าเฉพาะของเทอร์มินอลไม่เหมือนกัน. เราสามารถดูค่าของเทอร์มินอลต่าง ๆที่ตั้งไว้ด้วยคำสั่ง `stty`.

stty อ้างอิงหน้า 415

ตัวอย่างที่ 4.13: ดูค่าของเทอร์มินอล ๆที่ตั้งไว้.

```
$ stty -a↵
speed 38400 baud; rows 24; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = M-^?; eol2 = M-^?;
start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V;
flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts
-ignbrk brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff
-iuclc ixany imaxbel
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt
echoctl echoke
```

จากผลลัพธ์ของคำสั่งจะเห็นว่ามีการกำหนดคีย์ต่าง ๆให้เข้ากับสัญญาณที่ต้องการเช่น ใช้ `^C` (กดคีย์ Ctrl ค้างไว้แล้วกดคีย์ C ตาม) จะหมายถึงการส่งสัญญาณ Interrupt ให้โปรแกรมเลิกทำงานเป็นต้น. อีกตัวอย่างที่จะแนะนำในที่นี้คือความสามารถ echo. เวลาเรากดคีย์ต่าง ๆจะเห็นว่าอักษรที่พิมพ์จะปรากฏบนหน้าจอเทอร์มินอล. นี่เป็นความสามารถของเทอร์มินอลที่เรียกว่า echo.

ตัวอย่างที่ 4.14: หยุดการแสดงผลของสิ่งที่พิมพ์จากแป้นพิมพ์.

```
$ stty -echo↵
$ I can not see ← พิมพ์ echo I can not see แต่จะไม่แสดงสิ่งที่พิมพ์ไป. จะแสดงแค่ผลลัพธ์.
$ $                                    ← พิมพ์ stty echo เพื่อกลับสู่สภาพปรกติ.
$ echo Now I can see again↵
Now I can see again
```

การประยุกต์ใช้การตั้งค่าเทอร์มินอลในลักษณะนี้เช่นใช้ในเวลาต้องการให้ผู้ใช้พิมพ์ข้อความที่เป็นความลับ, รหัสผ่าน. ในกรณีนี้สามารถตั้งค่าให้เทอร์มินอลไม่ echo รหัสผ่านที่พิมพ์ทางหน้าจอ.

คำสั่ง stty ยังสามารถใช้แก้ไขเทอร์มินอลที่มีการแสดงผลไม่ถูกต้องหลังจากที่เปิดดูไฟล์ไบนารี. ในกรณีนี้จะใช้คำสั่ง “stty sane” เพื่อปรับสภาพของเทอร์มินอลให้เป็นปรกติ. หรือจะใช้คำสั่ง reset ที่เคยแนะนำไปแล้วก็ได้.

พฤติกรรมของเทอร์มินอลที่สามารถกำหนดได้ด้วยคำสั่ง stty มีหลายอย่างให้อ่านรายละเอียดจาก stty(1).

4.2.7 ควบคุมคำสั่ง

ลินุกซ์เป็นระบบปฏิบัติการที่สามารถใช้โปรแกรมหลายโปรแกรมพร้อม ๆกัน. ลินุกซ์เคอร์เนลจะมีหน้าที่แบ่งเวลาให้โปรเซสต่าง ๆใช้หน่วยประมวลผลทำงาน. เราสามารถใช้คำสั่ง nice เปลี่ยนความสำคัญของโปรเซส (scheduling priority) บางตัวที่ไม่สำคัญให้หน่วยประมวลไม่ต้องเสียเวลากับโปรเซสนั้นมากกนัก. ในทางกลับกัน, สามารถกำหนดให้หน่วยประมวลผลใช้เวลากับโปรเซสที่สำคัญมากขึ้นกว่าปรกติได้. คำสั่ง nice มักจะใช้โปรแกรมที่ใช้เวลาทำงานนาน ๆหรือโปรแกรมที่ทำงานแบบ background. คำสั่งที่ทำงานใช้เวลาไม่มากแล้วจบทันทีไม่จำเป็นต้นใช้ nice.

คำสั่ง nice ใช้ตอนที่จะเรียกใช้โปรแกรมที่ต้องการ.

nice อ้างอิงหน้า 423

ตัวอย่างที่ 4.15: ตั้งความสำคัญของโปรเซสเวลารัน.

```
$ nice -n 19 ./a.out↵
```

สมมติว่าโปรแกรม a.out เป็นโปรแกรมที่เขียนขึ้นเอง.

ตัวเลือก -n เป็นการระบุความสำคัญ (priority) ของคำสั่งโดยความสำคัญจะเป็นตัวเลขตั้งแต่ -20 จนถึง 19. ค่าตัวเลขยิ่งน้อยยิ่งมีความสำคัญสูง. ถ้าไม่มีการระบุความสำคัญจะถือว่ามีค่าเป็น 10 โดยปริยาย. ผู้ใช้ทั่วไปไม่สามารถใช้ค่าความสำคัญมากได้แก่เลขที่มีค่าน้อยกว่า 0, ต้องเป็น root เท่านั้น.

คำสั่ง nice จะตั้งค่าความสำคัญของโปรเซสได้ตอนที่จะสั่งคำสั่งเท่านั้น. ถ้าต้องการเปลี่ยนความสำคัญของโปรแกรมที่สั่งแล้ว (โปรเซส) จะใช้คำสั่ง renice ซึ่ง root เท่านั้นที่จะใช้คำสั่งนี้ได้.

renice อ้างอิงหน้า 423

ตัวอย่างที่ 4.16: เปลี่ยนความสำคัญของโปรเซส.

```
# renice -10 -p 5362↵
5362: old priority 0, new priority -10
```

จากตัวอย่างเป็นการเปลี่ยนค่าความสำคัญของโปรเซสหมายเลข 5362 ให้เป็น -10 (ความสำคัญมากขึ้น).

ในบทที่ผ่านมาเราได้ทำความรู้จักกับการสั่งคำสั่งแบบ background ไปแล้ว. สมมติว่าเราล็อกอินผ่านทางเน็ตเวิร์กเพื่อสั่งคำสั่งอะไรบางอย่าง. คำสั่งนั้นกินเวลานานเกินกว่าที่จะรอได้, ในกรณีเราสามารถสั่งคำสั่งแบบ background และล็อกเอาท์ออกจากระบบนั้นโดยไม่ต้องรอให้โปรแกรมที่รันจบแล้วค่อยล็อกเอาท์. คำสั่งที่สั่งจะทำงานต่อไปถึงแม้ว่าเราจะล็อกเอาท์ไปแล้วก็ตามเพราะเราสั่งคำสั่งแบบ background. ถ้าคำสั่งนั้นยกเลิกการทำงานถึงแม้ว่าจะสั่งคำสั่งแบบ background แล้วล็อกเอาท์, ให้ใช้คำสั่ง nohup ช่วย.

nohup อ้างอิงหน้า 423

การใช้คำสั่ง nohup ทำได้ดังนี้.

ตัวอย่างที่ 4.17: ให้คำสั่งทำงานต่อไปหลังจากล็อกเอาท์.

```
$ nohup ./a.out &↵
[1] 11599                                          ← หมายเลขโปรเซส
$ logout↵                          ← ล็อกเอาท์ได้โดยที่ a.out ทำงานต่อไปเรื่อยๆจนจบ.
```

ผู้ใช้ต้องสั่งคำสั่งเป็นแบบ background เอง. หลังจากนั้นสามารถล็อกเอาท์ได้โดยที่คำสั่งนั้นยังคงทำงานต่อไป. ถ้าคำสั่งนั้นมีการส่งข้อมูลออกทาง stdout จะเก็บไว้ในไฟล์ nohup.out. ในทางเทคนิคแล้วคำสั่ง nohup จะทำให้โปรแกรมที่สั่งเพิกเฉยกับสัญญาณ SIGHUP ทำให้ผู้ใช้สามารถล็อกเอาท์ได้. คำสั่ง nohup จะไม่เปลี่ยนความสำคัญของโปรเซสให้, ถ้าต้องการให้คำสั่งนั้นกระทำการโดยมีความสำคัญต่ำให้ใช้คำสั่ง nice เข้าช่วย.

4.2.8 นอนพัก

sleep เป็นคำสั่งที่หยุด, ไม่ทำอะไรตามระยะเวลาที่กำหนด. เช่นถ้าต้องการหยุดรอ 2 นาทีแล้วค่อยสั่งคำสั่งทำได้ดังนี้.

ตัวอย่างที่ 4.18: ใช้คำสั่ง sleep.

```
$ sleep 2m; echo 'Wake up!'↵
Wake up!                          ← เวลาผ่านไป 2 นาทีแล้ว echo จึงทำงาน.
```

ระยะเวลาถ้าเป็นตัวเลขโดยไม่มีหน่วยจะหมายถึงวินาที, m หมายถึงนาที, h หมายถึงชั่วโมง, และ d หมายถึงวัน.

4.2.9 คณิตศาสตร์

expr อ้างอิงหน้า 421

คำสั่ง expr สามารถใช้คำนวณคณิตศาสตร์ง่าย ๆเช่นบวก, ลบ, คูณ, หารได้ตามตัวอย่างต่อไปนี้.

ตัวอย่างที่ 4.19: ใช้ expr คำนวณคณิตศาสตร์แบบง่าย ๆ.

```
$ expr 1 + 2 + 3↵
6
$ expr 1 - 2↵
-1
$ expr 2 \* 4↵                          ← ต้องใช้เครื่องหมาย \ นำหน้า *
8
$ expr 5 / 3↵
1                                                  ← ปัดเศษทิ้ง
$ expr 5 % 3↵
2                                              ← เศษของการหารด้วย 3
```

การคำนวณนี้สามารถทำได้เฉพาะเลขจำนวนเต็มเท่านั้นและระหว่างตัวปฏิบัติการต้องมีช่องไฟคั่น. คำสั่งนี้ใช้ไม่สะดวกเท่าไหร่นักและสามารถใช้ความสามารถการกระจายนิพจน์

คณิตศาสตร์ของเชลล์ bash จะสะดวกกว่า. ตัวอย่างต่อไปนี้จะให้ผลเหมือนตัวอย่างที่แสดงไปแล้วแต่จะใช้ความสามารถของเชลล์ bash แทน.

ตัวอย่างที่ 4.20: ใช้ความสามารถของเชลล์คำนวณคณิตศาสตร์แบบง่าย ๆ.

```
$ echo $((1+2+3))↵
6
$ echo $((1-2))↵
-1
$ echo $((2*4))↵                          ← ไม่ต้องมี เครื่องหมาย \ นำหน้า *
8
$ echo $((5/3))↵
1                                                  ← ปัด เศษทิ้ง
$ echo $((5%3))↵
2                                          ← เศษของการหารด้วย 3
```

ตารางที่ 4.3: ตัวปฏิบัติการคณิตศาสตร์ต่าง ๆที่ใช้ในเชลล์.

ตัวปฏิบัติการ	ความหมาย
var++ *var*--	เพิ่มหรือลดค่าของตัวแปรหลังใช้งาน.
++*var* --*var*	เพิ่มหรือลดค่าของตัวแปรก่อนใช้งาน.
! ~	ตัวกระทำเชิงตรรกะให้เป็นเท็จ. หรือกลับค่าบิต.
**	ยกกำลัง.
+ - * /	บวก, ลบ, คูณ, หาร.
%	เศษของการหาร.
<< >>	เลื่อนบิตไปทางซ้ายหรือขวา.
<= >= < > != ==	เปรียบเทียบค่า.
&	ตัวปฏิบัติการบิต AND.
^	ตัวปฏิบัติการบิต exclusive OR.
|	ตัวปฏิบัติการบิต OR.
&&	ตัวปฏิบัติการตรรกะ AND.
||	ตัวปฏิบัติการตรรกะ OR.

คำสั่ง `factor` จะแสดงตัวประกอบของเลขจะนวนเต็มที่เป็นอาร์กิวเมนต์. ❑ factor อ้างอิงหน้า 421

ตัวอย่างที่ 4.21: หาตัวประกอบของจำนวนที่ต้องการ.

```
$ factor 20 9 2004 23354523↵
20: 2 2 5
9: 3 3
2004: 2 2 3 167
23354523: 3 3 2594947
```

เนื่องจากตัวประกอบของเลขจำนวนเฉพาะ (prime) คือตัวมันเอง. ดังนั้นเราสามารถใช้คำสั่ง factor ตรวจสอบจำนวนที่ต้องการว่าเป็นจำนวนเฉพาะหรือไม่ได้.

คำสั่งที่เกี่ยวข้องกับคณิตศาสตร์อันสุดท้ายคือ seq และมีประโยชน์ใช้ในการเขียนเชลล์สคริปต์ด้วย. คำสั่ง seq จะแสดงจำนวนตัวเลขเป็นลำดับไปเรื่อย ๆตัวอย่างเช่น.

seq อ้างอิงหน้า 424

ตัวอย่างที่ 4.22: แสดงลำดับของจำนวนไปเรื่อย ๆ.

```
$ seq 3↵                                    ← ถ้ามีอาร์กิวเมนต์ตัวเดียวจะเริ่มตั้งแต่ 1
1
2
3
$ seq -s ' ' 10 19↵                         ← ลำดับเพิ่มทีละ 1 ตั้งแต่ 10 ถึง 19
10 11 12 13 14 15 16 17 18 19
$ seq -s ' ' -w 5 5 50↵                     ← ลำดับเพิ่มทีละ 5 ตั้งแต่ 5 ถึง 50
05 10 15 20 25 30 35 40 45 50
```

ตัวเลือก -s ใช้ระบุสายอักขระที่ต้องการใช้เป็นตัวคั่นระหว่างจำนวน. ถ้าไม่ระบุจะใช้ \n เป็นตัวคั่นระหว่างจำนวนโดยปริยาย. ตัวเลือก -w จะเติมเลข 0 ก่อนหน้าตัวเลขทำให้จำนวนที่แสดงทั้งหมดมีขนาดเท่ากัน (ใช้จำนวนตัวอักษรเท่ากัน). คำสั่ง seq นี้มีประโยชน์ใช้ในเชลล์สคริปต์ร่วมกับไวยกรณ์ for.

ตัวอย่างที่ 4.23: ใช้ seq ร่วมกับ for

```
$ for i in `seq -f '%03g' 10`↵              ← ใช้ตัวเลือก -f จัดรูปแบบจำนวน
> do↵
> wget -nd http://somewhere.net/doc/$i.html↵ ← ใช้โปรแกรมคำสั่ง wget ดาวน์โหลดไฟล์
> done↵
```

ตัวอย่างข้างบนเป็นการใช้คำสั่ง seq ร่วมกับไวยกรณ์ for เพื่อดาวน์โหลดเอกสารที่เป็นตัวเลขเรียงกันเช่น 001.html, 002.html ฯลฯ ด้วยคำสั่ง wget. ในตัวอย่างจะใช้ตัวเลือก -f เพื่อระบุรูปแบบที่ต้องการคือให้เติมศูนย์ให้จำนวนเต็มมีขนาด 3 ตัวอักษร.

4.2.10 ซ้ำ

yes อ้างอิงหน้า 426

yes เป็นคำสั่งที่แปลกคือจะแสดงตัวอักษร y ไปเรื่อย ๆบรรทัดต่อบรรทัดจนกว่าจะกด C-c เพื่อหยุดการทำงาน. ถ้าให้อากิวเมนต์เป็นคำว่า Hello ก็จะแสดงคำว่า Hello ไปเรื่อย ๆ. คำสั่งนี้สามารถใช้ร่วมกับคำสั่งอื่นที่ต้องการคำตอบและผู้ใช้ขี้เกียจพิมพ์ y (หรือสายอักขระใด ๆก็ได้) ไปเรื่อย ๆเพื่อตอบคำถาม. ตัวอย่างเช่นคำสั่งที่ใช้ลบไฟล์ rm -i จะถามย้ำให้เราตอบ y หรือ n ทุกครั้งที่จะลบไฟล์. แทนที่เราต้องพิมพ์ y ตอบด้วยมือ, สามารถทำตามตัวอย่างต่อไปนี้ได้.

ตัวอย่างที่ 4.24: ใช้คำสั่ง yes ตอบคำถามที่ซ้ำ ๆกัน.

```
$ yes | rm *↵                               ← สมมติว่า rm คือ alias ของ rm -i
```

4.2.11 เปลี่ยนตัวเองเป็นผู้ใช้อื่น

คำสั่ง su ใช้สำหรับเปลี่ยน ID จากผู้ใช้คนหนึ่งไปเป็นผู้ใช้ที่ต้องการ. โดยปรกติจะมักใช้เปลี่ยน ID ของผู้ใช้ทั่วไปเป็น root เพื่อสั่งคำสั่งที่ผู้ใช้ทั่วไปไม่สามารถสั่งได้. นอกจากการเปลี่ยน ID ของผู้ใช้แล้วยังสามารถใช้คำสั่ง su สั่งคำสั่งที่ต้องการโดยใช้ ID ของคนอื่นเป็นผู้กระทำได้ด้วยโดยที่ไม่ต้องเปลี่ยน ID เป็นคนนั้น.

ตัวอย่างที่ 4.25: ใช้ su สั่งคำสั่งด้วย ID ของคนอื่น.

```
$ su -c "cat /etc/shadow"↵
Password:                                          ← ใส่พาสเวิร์ดของ root
...
halt:*:9797:0:::::
operator:*:9797:0:::::
shutdown:*:9797:0:::::
...
```

เนื่องจาก ID ที่ใช้สั่งคำสั่ง cat คือ root, จึงสามารถเปิดอ่านไฟล์ที่ root ดูได้แต่คนอื่นอ่านไม่ได้เช่นไฟล์ /etc/shadow เป็นต้น. การที่เปลี่ยน ID ในลักษณะนี้เรียกว่า *effective user ID* คือ ID ที่มีผลจริงขณะปฏิบัติงาน.

4.3 แพ็กเกจ textutils

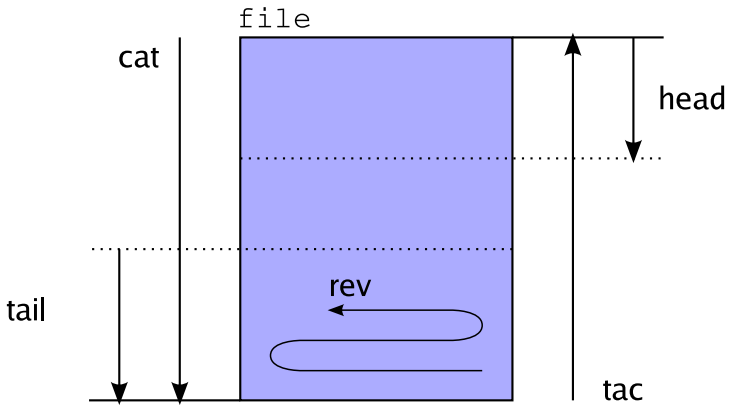
จุดเด่นของคำสั่งในระบบปฏิบัติการยูนิกซ์ซึ่งสืบทอดมาถึงลินุกซ์ได้แก่คำสั่งที่ใช้จัดการไฟล์เท็กซ์ทั้งหลาย. เนื่องจากคำสั่งเหล่านี้รับข้อมูลเป็นเท็กซ์และให้ผลลัพธ์เป็นเท็กซ์, การใช้คำสั่งต่าง ๆร่วมกันโดยใช้ไปป์จึงสะดวกและมีประสิทธิภาพในหลาย ๆด้าน. ในช่วงนี้จะแนะนำคำสั่งที่ประมวลผลข้อมูลเท็กซ์ซึ่งจะมีประโยชน์ไม่ว่าจะใช้ช่วยการพัฒนาซอฟต์แวร์, ดูแลระบบ, หรือใช้งานทั่วไป.

ตารางที่ 4.4: โปรแกรมคำสั่งในแพ็กเกจ textutils.

คำสั่ง	คำอธิบาย	อ้างอิง
cat	รวมข้อมูลแล้วแสดงผลทาง stdout.	หน้า 384
cksum	แสดงผลรวมตรวจสอบ (checksum) และนับจำนวนไบต์ในไฟล์.	หน้า 384
comm	เปรียบเทียบไฟล์สองไฟล์บรรทัดต่อบรรทัด.	หน้า 384
csplit	แยกไฟล์ออกเป็นไฟล์ย่อย ๆตามเนื้อหาที่ระบุ.	
cut	ตัดส่วนที่ไม่ต้องการออกจากทุกบรรทัดในไฟล์.	หน้า 385
expand	เปลี่ยน tab ให้เป็น space.	หน้า 386
fmt	จัดรูปแบบของข้อความให้เหมาะสม.	
fold	ตัดบรรทัดของข้อมูลเข้าตามความกว้างที่ระบุ.	
join	รวมบรรทัดของไฟล์สองไฟล์เมื่อเจอส่วนที่เหมือนกัน.	หน้า 389

`md5sum`	คำนวณและตรวจสอบค่า MD5.	หน้า 389
`nl`	แสดงเลขบรรทัดของไฟล์ที่ต้องการ.	หน้า 390
`od`	เทข้อมูล (dump) ออกมาให้อยู่ในรูปแบบเลขฐานแปดและอื่น ๆ.	หน้า 414
`paste`	นำข้อมูลในไฟล์เป็นบล็อกมาต่อกันในแนวนอน.	หน้า 390
`pr`	แปลงข้อความในไฟล์เพื่อพิมพ์ออกทางเครื่องพิมพ์.	
`ptx`	สร้าง permuted index.	
`sort`	เรียงลำดับบรรทัดในไฟล์.	หน้า 390
`split`	แบ่งไฟล์ออกเป็นไฟล์ย่อย ๆ.	หน้า 391
`sum`	คำนวณผลรวมตรวจสอบ (checksum) และนับบล็อก.	
`tac`	รวมไฟล์และแสดงผลแบบกลับบรรทัด (ตรงข้ามกับ `cat`)	
`tail`	แสดงช่วงท้ายของไฟล์.	หน้า 391
`tr`	แปลงและลบอักขระที่ต้องการ.	หน้า 392
`tsort`	จัดลำดับ topological sort.	
`unexpand`	เปลี่ยน space เป็น tab.	หน้า `unexpand`
`uniq`	ลบบรรทัดที่ซ้ำกัน. มักใช้ร่วมกับคำสั่ง `sort`.	หน้า 393
`wc`	แสดงจำนวนไบต์, คำ, และบรรทัดในไฟล์.	หน้า 393
`head`	แสดงส่วนต้น ๆของไฟล์.	หน้า 388

4.3.1 แสดงข้อมูล



รูปที่ 4.2: คำสั่งที่ใช้แสดงข้อมูลส่วนต่าง ๆ.

จากบทที่ผ่านมามีตัวอย่างการใช้งานคำสั่ง `cat` มากพอและไม่จำเป็นต้องอธิบายแล้วว่าคำสั่ง `cat` ใช้ทำอะไร. คำสั่ง `cat` สามารถใช้แสดงเลขบรรทัดทุกบรรทัดด้วยตัวเลือก

-n ซึ่งในแพ็กเกจ textutils ก็มีคำสั่งที่ใช้เลขบรรทัดในไฟล์ด้วยเหมือนกันแต่มีลูกเล่นมากกว่า.

ถ้าใช้คำสั่ง nl โดยไม่มีอาร์กิวเมนต์จะแสดงเลขบรรทัดเฉพาะบรรทัดที่มีข้อมูลเท่านั้น. คำสั่ง nl สามารถแสดงเลขบรรทัดได้หลายแบบขึ้นอยู่กับตัวเลือกที่ระบุ. การแสดงเลขบรรทัดที่คำสั่ง cat ไม่สามารถทำได้เช่นการเติมศูนย์ที่ตัวเลขให้มีจำนวนตัวอักษรเท่ากัน.

nl อ้างอิงหน้า 390

ตัวอย่างที่ 4.26: ใช้ nl แสดงเลขบรรทัด.

```
$ yes | head -n 5 | nl -n rz -b a -w 3↵        ← ใช้คำสั่ง yes สร้างข้อมูล 5 บรรทัด
001     y
002     y
003     y
004     y
005     y
```

จากตัวอย่างเป็นการแสดงเลขบรรทัดโดยจัดให้เลขบรรทัดชิดขวาและเติมเลขศูนย์ด้วยตัวเลือก -n rz, แสดงตัวเลขทุกบรรทัดด้วยตัวเลือก -b a และให้เลขบรรทัดมีสามหลักด้วยตัวเลือก -w 3. คำสั่ง nl จะมีประโยชน์เมื่อต้องการดูเลขบรรทัดเช่นเวลาพิมพ์รหัสต้นฉบับพร้อมเลขบรรทัดออกทางเครื่องพิมพ์.

คำสั่ง tac เป็นการเล่นคำโดยกลับตัวอักษรของคำสั่ง cat เป็น tac. คำสั่งนี้จะแสดงข้อมูลเป็นบรรทัดโดยจะแสดงข้อมูลที่อยู่ในบรรทัดสุดท้ายก่อนไล่ขึ้นไปเรื่อย ๆจนถึงบรรทัดแรกซึ่งการทำงานนี้จะตรงกันข้ามกับคำสั่ง cat. นอกจากนี้ยังมีคำสั่ง rev สำหรับแสดงข้อมูลจากข้างหลังไปข้างหน้า, ไบต์ต่อไบต์.

4.3.2 จัดรูปแบบข้อมูล

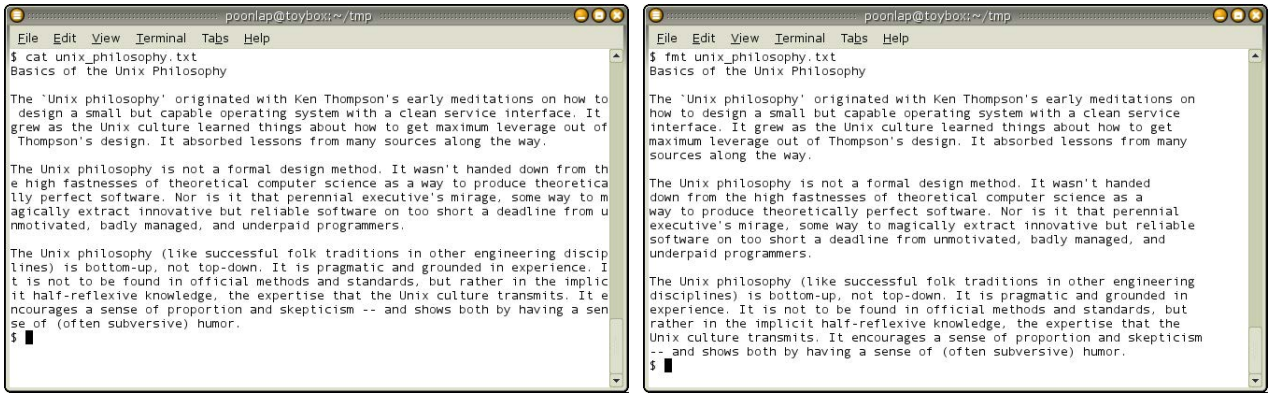
คำสั่งที่ใช้จัดรูปแบบข้อมูลเท็กซ์ได้แก่ fmt, pr และ fold. คำสั่งเหล่านี้สร้างมาเพื่อใช้กับข้อมูลเท็กซ์ภาษาอังกฤษแต่ก็ควรจะรู้ไว้เพราะเป็นคำสั่งที่ใช้งานได้จริงและมีประโยชน์.

รูปที่ 4.3 แสดงความแตกต่างระหว่างข้อมูลเท็กซ์ที่ยังไม่ได้จัดย่อหน้ากับข้อมูลที่จัดย่อหน้าแล้วด้วยคำสั่ง fmt.

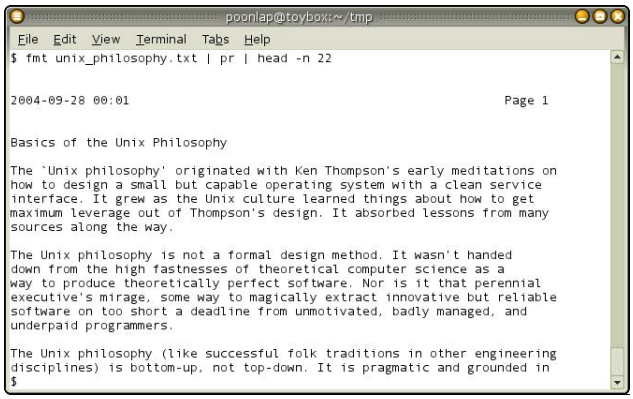
คำสั่ง fmt จะพยายามตัดเรียงข้อมูลเป็นบรรทัด ๆ โดยที่แต่ละบรรทัดมีความยาวไม่เกิน 75 ตัวอักษร. จำนวนอักษรนี้สามารถกำหนดได้ด้วยตัวเลือก -w (width). ถ้าใช้ตัวเลือก -u จะเป็นการลบช่องไฟที่เกินและไม่จำเป็นได้ด้วย. ตัวอย่างเช่น “a pencil” เป็น “a pencil”. บรรทัดว่างที่อยู่ในข้อมูลนำเข้าจะถือว่าเป็นการเริ่มต้นย่อหน้า. คำสั่ง fmt จะแบ่งบรรทัดที่ยาวเกินจำนวนอักษรที่กำหนดและจะรวมบรรทัดที่สั้น ๆให้มีความกว้างไม่เกินที่กำหนด.

fmt อ้างอิงหน้า 387

คำสั่ง pr ใช้สำหรับจัดเรียงหน้าเพื่อพิมพ์ออกทางเครื่องพิมพ์. การจัดเรียงหน้านี้เป็นการจัดเรียงแบบง่าย ๆตามความกว้างของบรรทัด (72 ตัวอักษร) และความยาวของหน้าในหน่วยบรรทัด (66 บรรทัด). นอกจากนี้คำสั่ง fmt จะพิมพ์หัวกระดาษให้ด้วย.



รูปที่ 4.3: ใช้ `fmt` จัดรูปแบบข้อมูล.



รูปที่ 4.4: ใช้คำสั่ง `fmt` และ `pr` จัดหน้ากระดาษแบบง่าย ๆ.

คำสั่งสุดท้ายที่ใช้จัดข้อมูลคือ `fold`. คำสั่ง `fold` จะตัดบรรทัดที่ยาว ๆให้มีความกว้าง (ตัวอักษร) ไม่เกินความกว้างที่กำหนด. คำสั่งนี้จะคล้ายกับคำสั่ง `fmt` แต่จะไม่มีการรวมบรรทัดที่ติดกัน. การตัดบรรทัดของคำสั่ง `fold` จะตัดตามจำนวนอักษรไม่ได้ตัดแบ่งบรรทัดตามช่องว่างทำให้ไม่สวยงาม. ถ้าต้องการตัดบรรทัดด้วยช่องว่าง, คือตัดบรรทัดแบ่งตามคำภาษาอังกฤษให้ใช้ตัดเลือก `-s`.

4.3.3 หัวหาง

คำสั่งที่ใช้แสดงบางส่วนของไฟล์ในแพ็กเกจ textutils ได้แก่ `head` และ `tail`. คำสั่ง `head` ใช้แสดงส่วนต้นของไฟล์เป็นบรรทัด, ในทางตรงกันข้ามคำสั่ง `tail` ใช้แสดงส่วนท้ายของไฟล์เป็นบรรทัด.

เราสามารถกำหนดจำนวนบรรทัดที่ต้องการให้แสดงได้ด้วยตัวเลือก `-n` แล้วตามด้วยจำนวนบรรทัดที่ต้องการ. ตัวเลือกนี้ใช้ได้ทั้งคำสั่ง `head` และ `tail`. คำสั่ง `tail` มีประโยชน์ใช้ดู *log* ของ *daemon* ซึ่ง daemon จะเขียนข้อความลงในไฟล์ล็อกต่อท้ายไฟล์เรื่อย ๆถ้ามีเหตุการณ์สำคัญเกิดขึ้น. เราสามารถดูการทำงาน, ข้อผิดพลาดของ dea-

daemon ▶
โปรเซสแบบ background ที่มีโปรเซส ID แม่ (PPID) เป็น 1 (โปรเซส `init`). โปรเซสเหล่านี้มักจะเป็นโปรเซสที่ทำงานโดยอัตโนมัติถูกสร้างตอนที่ระบบเริ่มทำงาน. โปรเซสเหล่านี้บางตัวอาจเป็นโปรเซสที่ช่วยดูแลระบบ, เซิฟร์เวอร์ เช่น `cron`, `sendmail` เป็นต้น.

mon ได้จาก log. ตัวอย่างเช่น log เก็บข้อผิดพลาดของเว็บเซิฟร์เวอร์ apache ได้แก่ ไฟล์ /var/log/apache/error_log เราสามารถติดตามดูข้อความที่เขียนในไฟล์นี้ได้เรื่อยโดยใช้ตัวเลือก -f.

ไฟล์ log ของ apache อาจจะแตกต่างกันแล้วแต่ระบบและดิสทริบิว, ตลอดจนรุ่นของ apache ที่ใช้.

-f ย่อมาจาก follow.

ตัวอย่างที่ 4.27: ใช้ tail ติดตามดู log.

```
# tail -f /var/log/apache/error_log↵
[Thu Aug 05 01:53:49 2004] [notice] caught SIGTERM, shutting down
[Thu Aug 05 23:02:43 2004] [notice] Digest: generating secret for digest aut
hentication ...
[Thu Aug 05 23:02:43 2004] [notice] Digest: done
[Thu Aug 05 23:02:44 2004] [notice] Apache/2.0.50 (Gentoo/Linux) configured -- r
esuming normal operations
[Thu Aug 05 23:32:58 2004] [error] [client 219.178.56.123] File does not exist:
/var/www/localhost/htdocs/default.ida
[Fri Aug 06 00:46:18 2004] [error] [client 219.154.229.60] request failed: URI t
oo long (longer than 8190)
[Fri Aug 06 02:17:14 2004] [notice] caught SIGTERM, shutting down
[Tue Sep 28 21:38:46 2004] [notice] Digest: generating secret for digest authent
ication ...
[Tue Sep 28 21:38:46 2004] [notice] Digest: done
[Tue Sep 28 21:38:47 2004] [notice] Apache/2.0.50 (Gentoo/Linux) configured -- r
esuming normal operations
                                        ← คำสั่งทำงานแบบ foreground
```

จากตัวอย่างถ้าเซิฟร์เวอร์มีข้อความ notice, error เพิ่มเติมก็จะเขียนใส่ไฟล์ log เพิ่มและ tail ก็จะแสดงบรรทัดที่เขียนเพิ่มเข้ามาโดยอัตโนมัติ. เราอาจจะสั่งคำสั่งแบบ background ก็ได้, แล้วใช้เทอร์มินอลนั้นทำงานอื่น ๆต่อไป. ถ้ามีข้อความเพิ่มเติมเข้ามาใน log ก็จะปรากฏในเทอร์มินอลนั้น. การใช้งานแบบนี้เหมาะสำหรับหาสาเหตุ, ข้อผิดพลาดเวลาเซิฟร์เวอร์หรือโปรแกรมที่มีไฟล์ log ทำงานผิดพลาดและเรากำลังแก้ไขอยู่.

บางครั้งถ้าใช้คำสั่ง tail กับตัวเลือก -f ดู log ค้างไว้นาน, มีโอกาสที่ระบบจะเปลี่ยนล็อกไฟล์ที่กำลังดูอยู่ไปเป็นชื่ออื่นแล้วสร้าง log ตัวใหม่ด้วยชื่อเดียวกัน. ตัวอย่างเช่นระบบอาจจะเปลี่ยนชื่อไฟล์ error_log ไปเป็น error_log.1 แล้วสร้าง log ใหม่ด้วยชื่อ error_log เหมือนเดิม. สำหรับ log ที่เก่าเกินไปก็อาจจะถูกลบทิ้งไปเพื่อประหยัดพื้นที่. ในกรณีที่มีการเปลี่ยน log แบบนี้คำสั่ง tail กับตัวเลือก -f จะไม่สามารถดู log ที่สร้างใหม่ได้ถึงแม้ว่าจะเป็นชื่อเดียวกันเพราะคำสั่ง tail จะยึดกับ file description เป็นหลัก, ไม่ได้ดูที่ชื่อ. ถ้าต้องการดูไฟล์ที่มีชื่อเดียวกันต่อไปเรื่อย ๆต้องใช้ตัวเลือก --follow=name แทน.

4.3.4 แบ่งไฟล์, รวมไฟล์

ในปัจจุบันอาจจะไม่มีความจำเป็นแบ่งไฟล์ที่มีขนาดใหญ่ให้เป็นไฟล์เล็ก ๆเท่าไรนัก, เพราะมีอุปกรณ์พกพาเก็บข้อมูลขนาดใหญ่เช่นฮาร์ดดิสก์พกพาแบบ USB เป็นต้น. อย่างไรก็ตามผู้อ่านก็ควรจะรู้วิธีการแบ่งไฟล์ให้เป็นไฟล์ย่อยด้วยสำหรับกรณีจำเป็นเช่น การแบ่งไฟล์ขนาดใหญ่ให้เป็นไฟล์ขนาดเล็กหลาย ๆไฟล์เก็บลงฟล็อปปี้ดิสก์.

การแบ่งไฟล์เป็นไฟล์ย่อย ๆสามารถใช้คำสั่ง split. ตัวอย่างเช่นการแบ่งไฟล์

split อ้างอิงหน้า 391

ใหญ่ ๆให้เป็นไฟล์ย่อย ๆมีขนาดพอดีกับฟล็อปปี้ดิสก์ 1440MB ทำได้ดังนี้.

ตัวอย่างที่ 4.28: แบ่งไฟล์ใหญ่ ๆให้มีขนาดพอกับแผ่นฟล็อปปี้ดิสก์.

```
$ split -b 'bc <<<'1440*1024'' bigfile small↵
$ ls -l↵
total 4108
-rw-r--r--    1 poonlap  users     2097152 Sep 22 23:19 bigfile
-rw-r--r--    1 poonlap  users     1474560 Sep 22 23:24 smallaa
-rw-r--r--    1 poonlap  users      622592 Sep 22 23:24 smallab
```

ตัวเลือก `-b` จะแบ่งไฟล์แต่ละไฟล์ให้มีขนาดตามที่กำหนด (ถ้าเป็นไปได้). ในกรณีจะพยายามแบ่งไฟล์ให้มีขนาดเป็น $1440 \times 1024 = 1474560$ ไบต์. การกำหนดจำนวนไบต์ในตัวอย่างใช้การแทนค่าคำสั่ง `bc` ซึ่งจะคำนวณผลลัพธ์เป็นไบต์ให้. ในตัวอย่างมีการกำหนดชื่อนำหน้าไฟล์ย่อย (prefix) เป็น small. ไฟล์ย่อยที่แบ่งแล้วจะเป็น smallaa และ smallab. aa และ ab คือส่วนตามหลังไฟล์ (suffix) ซึ่งจะเป็นตัวอักษร 2 ตัว. ถ้าต้องการเป็นตัวเลขแทนให้ใช้ตัวเลือก `-d`.

การรวมไฟล์ย่อย ๆให้เป็นไฟล์เดียวกันให้ใช้คำสั่ง `cat`.

ตัวอย่างที่ 4.29: รวมไฟล์หลาย ๆไฟล์ให้เป็นไฟล์เดียวกัน.

```
$ cat small* > combine↵    ← เขียนชื่อไฟล์ small* ได้เพราะเรียงลำดับอยู่แล้ว (aa, ab)
$ ls -l↵
total 6160
-rw-r--r--    1 poonlap  users     2097152 Sep 22 23:19 bigfile
-rw-r--r--    1 poonlap  users     2097152 Sep 22 23:59 combine
-rw-r--r--    1 poonlap  users     1474560 Sep 22 23:58 smallaa
-rw-r--r--    1 poonlap  users      622592 Sep 22 23:58 smallab
```

เราสามารถตรวจสอบจำนวนไบต์ของไฟล์ที่รวมแล้วโดยใช้คำสั่ง `ls`. แต่การดูจำนวนไบต์เป็นเพียงแค่การตรวจดูขนาด, ไม่สามารถบอกได้ว่าเนื้อหานั้นถูกต้องหรือไม่. สำหรับการตรวจสอบเนื้อหานั้นอาจจะใช้คำสั่ง `cksum` หรือ `md5sum`.

cksum อ้างอิงหน้า 384
md5sum อ้างอิงหน้า 389

ตัวอย่างที่ 4.30: ตรวจสอบดูค่าเฉพาะของไฟล์.

```
$ cksum bigfile combine↵
1742489887 2097152 bigfile
1742489887 2097152 combine
$ md5sum bigfile combine↵
b2d1236c286a3c0704224fe4105eca49  bigfile
b2d1236c286a3c0704224fe4105eca49  combine
```

คำสั่งทั้งสองจะหาค่า *checksum* ของข้อมูลแต่ใช้วิธีการคำนวณต่างกัน. ปัจจุบัน `md5sum` ใช้ในการตรวจสอบไฟล์อิมเมจของซีดีว่าข้อมูลที่ดาว์นโหลดมานั้นถูกต้องหรือไม่, โดยจะมีไฟล์ที่ชื่อ MD5SUMS อยู่ที่เดียวกันกับไฟล์ที่ดาว์นโหลดด้วย. ในไฟล์นี้จะมีผลลัพธ์ของคำสั่ง `md5sum` บันทึกอยู่. คนที่ดาว์นโหลดไฟล์ที่ต้องการใช้ให้ใช้คำสั่ง `md5sum` หาค่า checksum ของไฟล์นั้นแล้วดูเปรียบเทียบกับไฟล์ที่มีค่า checksum ไว้ให้

คำสั่ง `sum` ก็ใช้หาค่า checksum เช่นกันแต่ไม่ให้ผลไม่ค่อยดีนัก (จำนวนบิตที่ใช้คำนวณน้อย).

checksum ▶ วิธีการตรวจสอบความผิดพลาดของข้อมูล, หรือค่าที่ได้จากการตรวจสอบ. ค่าเหล่านี้สามารถบอกได้ว่าข้อมูลที่ได้รับเช่นข้อมูลผ่านทางเน็ตเวิร์กมีข้อผิดพลาดหรือไม่. วิธีการคำนวณและหาค่าเหล่านี้มีหลายวิธีเช่น cyclic redundancy checks, cryptographic message digest. วิธีแบบ cryptographic message digest ยังมี

แล้วด้วยตา. หรือจะดาว์นโหลดไฟล์ MD5SUMS มาด้วยแล้วใช้ตัวเลือก -c ตรวจสอบว่าค่า checksum ตรงกันหรือไม่.

ตัวอย่างที่ 4.31: ใช้ md5sum ตรวจสอบความถูกต้องของไฟล์ที่ดาว์นโหลด.

```
$ ls -l sarge-i386-netinst.iso MD5SUMS↵
-rw-r--r--    1 poonlap  users          57 Sep 23 23:25 MD5SUMS
-rw-------    1 poonlap  users    119470080 Sep 12 15:16 sarge-i386-netinst.iso
$ cat MD5SUMS↵
1068812b8de80f05c55119b0dc64e488  sarge-i386-netinst.iso
$ md5sum -c MD5SUMS↵
sarge-i386-netinst.iso: OK
```

4.3.5 จัดการข้อมูลที่แบ่งเป็นคอลัมน์

ในระบบปฏิบัติการลินุกซ์มักใช้ไฟล์เท็กซ์เก็บข้อมูลต่าง ๆเช่น ไฟล์ /etc/shadow, /etc/services เป็นต้น. ไฟล์เหล่านี้อาจจะเป็นไฟล์ที่เก็บข้อมูลของระบบ, ไฟล์ตั้งค่าเริ่มต้นของโปรแกรม, หรือไฟล์ล็อก (log file) เก็บบันทึกรายงานการทำงานของโปรแกรม ฯลฯ. บางครั้งเราต้องการสกัดข้อมูลจากไฟล์เหล่านี้โดยระบุคอลัมน์ที่ต้องการ. ในกรณีเราสามารถใช้คำสั่ง cut เพื่อเลือกเอาส่วนที่ต้องการออกจากบรรทัดได้.

ตัวอย่างเช่นไฟล์ /etc/passwd เก็บข้อมูลของผู้ใช้ในระบบ. ข้อมูลต่าง ๆของผู้ใช้หนึ่งคนจะเก็บเป็นบรรทัดโดยมีรูปแบบเป็น

```
account:password:UID:GID:comment:directory:shell
```

จะเห็นว่าข้อมูลที่เกี่ยวกับผู้ใช้จะเก็บอยู่ในหน่วยของบรรทัด, และข้อมูลต่าง ๆที่เกี่ยวข้องกับผู้ใช้นั้นจะเก็บอยู่ในหน่วยของคอลัมน์. สำหรับไฟล์ /etc/passwd จะใช้เครื่องหมาย : เป็นตัวแบ่งคอลัมน์ซึ่งแต่ละคอลัมน์เก็บข้อมูลดังต่อไปนี้.

- account
  ชื่อล็อกอินของผู้ใช้ในระบบ.

- password
  ในระบบยูนิกซ์สมัยเก่ารหัสผ่านของผู้ใช้ในระบบจะ*เข้ารหัส (encrypt)* และเก็บไว้ในไฟล์ /etc/passwd. ไฟล์ /etc/passwd นี้เป็นไฟล์ที่ใครก็ได้สามารถอ่านได้ดังนั้นจึงไม่ปลอดภัยถ้ามีผู้ประสงค์ร้ายเอารหัสผ่านของ root ที่เข้ารหัสไว้ไปถอดรหัส. ระบบยูนิกซ์รุ่นใหม่และลินุกซ์จึงเก็บรหัสผ่านที่เข้ารหัสแล้วไว้ในไฟล์ต่างหากคือไฟล์ /etc/shadow ซึ่ง root เท่านั้นที่อ่านได้. ส่วนรหัสผ่านที่บันทึกไว้ในไฟล์ /etc/passwd จะแทนด้วยตัวอักษร x.

- UID
  ตัวเลข User ID ของผู้ใช้.

- GID

ตัวเลข Group ID หลักที่ผู้ใช้เป็นสมาชิก. ชื่อกลุ่ม, GID และสมาชิกสามารถดูได้จากไฟล์ /etc/group.

- comment
  ส่วนที่เป็นหมายเหตุ. อาจจะเป็นชื่อเต็มของผู้ใช้ก็ได้.
- directory
  หมายถึงโฮมไดเรกทอรีของผู้ใช้.
- shell
  เชลล์เมื่อผู้ใช้ล็อกอิน. ถ้าไม่ต้องการให้ผู้ใช้นั้นล็อกอินก็อาจจะใช้ /bin/false แทนก็ได้.

ไฟล์อื่นเช่นไฟล์ /etc/services ก็เหมือนกับไฟล์ /etc/passwd คือใช้เก็บข้อมูล. เก็บชื่อโปรโตคอลอินเทอร์เน็ต, หมายเลขพอร์ต, ชื่ออื่น ๆ, และหมายเหตุ. แต่ไฟล์นี้จะใช้ tab เป็นตัวแบ่งคอมลัมน์แทนที่จะใช้ colon เป็นตัวแบ่งคอลัมน์.

ตัวอย่างที่ 4.32: ตัวอย่างไฟล์ /etc/passwd และไฟล์ /etc/services.

```
$ cat /etc/passwd↵
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false
daemon:x:2:2:daemon:/sbin:/bin/false
adm:x:3:4:adm:/var/adm:/bin/false
...
$ cat /etc/services↵
...
finger          79/tcp
www             80/tcp          http            # WorldWideWeb HTTP
www             80/udp                          # HyperText Transfer Protocol
link            87/tcp          ttylink
...
```

ตัด — cut

สมมติเราต้องการดูแค่ชื่อล็อกอินและ uid จากไฟล์ /etc/passwd, เราสามารถใช้ cut เลือกเอาเฉพาะคอลัมน์ที่ 1 ที่เป็นชื่อล็อกอินและคอลัมน์ที่ 3 ที่เป็น uid แสดงบนหน้าจอได้ดังนี้.

ตัวอย่างที่ 4.33: ใช้ cut เลือกคอลัมน์ที่ต้องการมาแสดง.

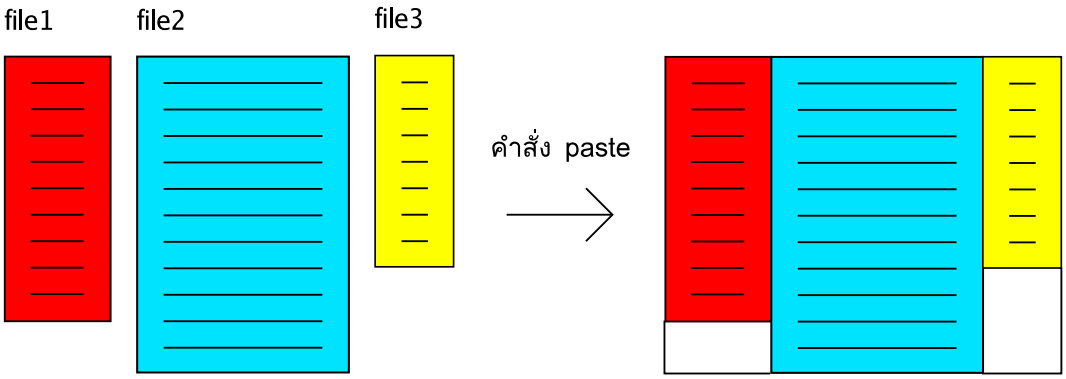
```
$ cut -f 1,3 -d : /etc/passwd↵
root:0
bin:1
daemon:2
adm:3
lp:4
...
```

ตัวเลือก -f ใช้ระบุคอลัมน์ที่ต้องการแสดง. ตัวเลือก -d ใช้ระบุตัวแบ่งคอลัมน์ซึ่งในที่นี้ได้แก่เครื่องหมาย colon.

-f มาจากคำว่า field และ -d มาจากคำว่า delimiter.

ต่อ — paste

cut & paste ในความหมายของเวิร์ดโปรเซสเซอร์คือการตัดแปะข้อความ. แต่คำสั่ง cut และ paste มีความหมายที่เกี่ยวกับการกระทำเป็นคอลัมน์, เป็นบล็อก. cut เป็นการแสดงคอลัมน์ที่ต้องการ, paste เป็นต่อไฟล์เป็นบล็อกเข้าเป็นไฟล์เดียว.



รูปที่ 4.5: การทำงานของคำสั่ง paste.

สมมติว่าเรามีไฟล์อยู่สองไฟล์. ไฟล์ที่หนึ่งชื่อ address.txt เป็นไฟล์เท็กซ์ที่เก็บชื่อและที่อยู่แบบ *csv (comma separated values)*. ไฟล์ที่สองชื่อ email.txt เก็บชื่อและเมลในรูปแบบเดียวกัน. เราสามารถรวมไฟล์สองไฟล์ให้เป็นไฟล์เดียวได้โดยคงคอลัมน์เหมือนเดิมไว้ได้ด้วยคำสั่ง paste.

csv ▶
เป็นคำย่อของ comma separated values. เป็นรูปแบบไฟล์ใช้บันทึกรายการ (record) เป็นบรรทัด ๆ. ภายในบรรทัดสามารถมีค่าได้หลายคอลัมน์ (field) และมักใช้เครื่องหมาย comma เป็นต้นแบ่งคอลัมน์. ไฟล์แบบนี้สามารถสร้างได้ง่าย ๆด้วยบรรณาธิกรณ์ทั่วไปหรือโปรแกรม spreadsheet. ในระบบลินุกซ์มีไฟล์แบบนี้เหมือนกันเช่นไฟล์ /etc/passwd, /etc/group แต่จะใช้เครื่องหมาย colon เป็นตัวแบ่งคอลัมน์.

ตัวอย่างที่ 4.34: รวมคอลัมน์ด้วยคำสั่ง paste

```
$ cat address.txt↵
ชื่อ, จังหวัด
สมชาย, กรุงเทพฯ
สมหมาย, เชียงใหม่
สมยศ, ภูเก็ต
$ cat email.txt↵
ชื่อ, เมล
สมชาย, somchai@gmail.com
สมหมาย, sommai@yahoo.com
สมยศ, somyod@hotmail.com
$ cut -f 2 -d , email.txt | paste -d , address.txt -↵
ชื่อ, จังหวัด, เมล
สมชาย, กรุงเทพฯ, somchai@gmail.com
สมหมาย, เชียงใหม่, sommai@yahoo.com
สมยศ, ภูเก็ต, somyod@hotmail.com
```

จากตัวอย่างจะใช้คำสั่ง cut เลือกเอาคอลัมน์ที่สองของไฟล์ email.txt ออกมาก่อนแล้วส่งให้คำสั่ง paste ทางไปป์. คำสั่ง paste รวมคอลัมน์จากไฟล์ address.txt และไฟล์ - ซึ่งหมายถึงข้อมูลที่รับมาทาง stdin (ข้อมูลจาก cut) แล้วรวมคอลัมน์บรรทัดต่อบรรทัด.

ในไฟล์ address.txt และ email.txt มีคอลัมน์ที่เหมือนกันคือคอลัมน์ที่เก็บชื่อคน. การรวมคอลัมน์ในกรณีนี้จะง่ายขึ้นถ้าใช้คำสั่ง join.

paste อ้างอิงหน้า 390

join อ้างอิงหน้า 389

ตัวอย่างที่ 4.35: รวมคอลัมน์ด้วยคำสั่ง join

```
$ join -t , address.txt email.txt↵
ชื่อ, จังหวัด, เมล
ลมชาย, กรุงเทพฯ, somchai@gmail.com
ลมหมาย, เชียงใหม่, sommai@yahoo.com
ลมยศ, ภูเก็ต, somyod@hotmail.com
```

คำสั่ง join จะต่อคอลัมน์ที่สองของไฟล์ที่สองในไฟล์ที่หนึ่ง, ถ้าคอลัมน์ที่หนึ่งของทั้งสองไฟล์มีค่าเหมือนกัน. ตัวเลือก -t ใช้กำหนดค่าตัวแบ่งคอลัมน์, มิฉะนั้นคำสั่ง join จะถือว่าช่องว่าเป็นตัวแบ่งคอลัมน์โดยปริยาย.

4.3.6 การเรียง, จัดลำดับข้อมูลในไฟล์

สมมติว่าเราต้องการจะดูว่าไดเรกทอรีต่าง ๆใต้ /usr ใช้เนื้อที่ไปเท่าไรสามารถใช้คำสั่ง du -sm /usr/*. ผลลัพธ์ที่ได้จะเรียงตามลำดับชื่อของไดเรกทอรีที่อยู่ใต้ไดเรกทอรี /usr. เพื่อความสะดวกในการดูผลเราสามารถใช้คำสั่ง sort เพื่อเรียงลำดับของข้อมูลบรรทัดต่อบรรทัดได้ดังนี้.

du อ้างอิงหน้า 395

sort อ้างอิงหน้า 390

ตัวอย่างที่ 4.36: การใช้ sort เรียงลำดับข้อมูล.

```
# du -sm /usr/* | sort -nr↵
1343    /usr/portage
1126    /usr/share
752     /usr/lib
609     /usr/src
331     /usr/kde
162     /usr/X11R6
152     /usr/bin
86      /usr/local
54      /usr/include
26      /usr/qt
15      /usr/sbin
4       /usr/libexec
3       /usr/i686-pc-linux-gnu
0       /usr/tmp
0       /usr/man
0       /usr/info
0       /usr/doc
```

จากตัวอย่างจะใช้ตัวเลือก -n เพื่อให้เรียงลำดับตามตัวเลข. ถ้าไม่ใช้ตัวเลือกนี้แล้วตัวเลขจะถือเป็นตัวอักขระและจะได้ผลที่ไม่ตรงตามต้องการ. ส่วนตัวเลือก -r ใช้เพื่อเรียงลำดับกลับจากมากไปหาน้อย, ซึ่งโดยปริยายแล้วการเรียงลำดับจะแสดงผลจากน้อยไปหามาก.

ลบบรรทัดที่ซ้ำ

uniq อ้างอิงหน้า 393

คำสั่งที่เกี่ยวข้องกับคำสั่ง sort ได้แก่คำสั่ง uniq ซึ่งใช้สำหรับลบบรรทัดที่ซ้ำออกจากข้อมูลที่เรียงลำดับแล้ว.

ในระบบยูนิกซ์มักจะมีไฟล์ที่เรียกว่า dictionary file ได้แก่ไฟล์ /usr/share/dict/words และ /usr/share/dict/web2. ไฟล์ทั้งสองเป็นไฟล์ที่รวมคำภาษาอังกฤษเอาไว้

ในระบบที่ผู้เขียนใช้มีไฟล์ทั้งสองเหมือนกันทุกประการ. แต่มีไฟล์อีกไฟล์คำศัพท์อีกไฟล์หนึ่งคือ /usr/share/dict/web2a ด้วย.

บรรทัดต่อบรรทัด. และโปรแกรมที่ใช้ไฟล์นี้ได้แก่โปรแกรม look ซึ่งจะแสดงคำที่ขึ้นต้นด้วยอักษรที่ใส่เป็นอาร์กิวเมนต์ของคำสั่ง. เพื่อที่จะแนะนำการใช้คำสั่ง uniq จะขอใช้ไฟล์คำศัพท์สองไฟล์นี้เป็นกรณีประกอบ.

look อ้างอิงหน้า 413

สมมติว่าเราต้องการรวมไฟล์ words กับไฟล์ web2a ให้เป็นไฟล์เดียวกัน. ปัญหามีอยู่ว่าเราจะแน่ใจได้อย่างไรว่าถ้ารวมคำศัพท์ที่อยู่ในไฟล์ทั้งสองเข้าด้วยกันแล้วจะไม่มีคำศัพท์ที่ซ้ำกัน? ในกรณีนี้เราสามารถตรวจสอบหรือลบบรรทัดที่ซ้ำได้ด้วยคำสั่ง uniq.

ตัวอย่างที่ 4.37: ลบบรรทัดที่ซ้ำด้วยคำสั่ง uniq.

```
$ wc -l /usr/share/dict/{words,web2a}↵
 234937 /usr/share/dict/words
  76205 /usr/share/dict/web2a
 311142 total
$ cat /usr/share/dict/words,web2a | sort | uniq | wc -l↵
311142
```

จากตัวอย่างจะเห็นว่าไฟล์ words มีคำศัพท์อยู่ 234,937 คำ, และไฟล์ web2a มีคำศัพท์อยู่ 76,205 คำ. เรารวมไฟล์สองไฟล์เข้าด้วยกันด้วยคำสั่ง cat หลังจากนั้นเรียงลำดับคำศัพท์ด้วย sort และสุดท้ายลบคำที่ซ้ำออกด้วยคำสั่ง uniq. ผลจากการนับบรรทัด (คำ) ปรากฏว่าจำนวนคำศัพท์เท่ากับจำนวนคำศัพท์ของไฟล์สองไฟล์รวมกันแสดงว่าไม่มีคำซ้ำกันในไฟล์ทั้งสอง.

คำสั่ง uniq ยังมีตัวเลือกสำหรับแสดงบรรทัดที่ซ้ำกันเท่านั้นได้แก่ -d. และตัวเลือกสำหรับแสดงบรรทัดที่ซ้ำกันเท่านั้นได้แก่ -u.

4.3.7 การเปรียบเทียบไฟล์

การเปรียบเทียบไฟล์เป็นสิ่งที่เกิดขึ้นบ่อยครั้งเมื่อทำงานกับคอมพิวเตอร์เช่นถ้าเป็นผู้ดูแลระบบบางครั้งต้องดูความแตกต่างระหว่างไฟล์ตั้งค่าเริ่มต้นของโปรแกรมต่าง ๆ, ถ้าเป็นผู้พัฒนาซอฟต์แวร์ก็อาจจะมีความจำเป็นต้องหาความแตกต่างระหว่างรหัสต้นฉบับเดิมกับรหัสต้นฉบับใหม่ เป็นต้น. ในระบบปฏิบัติการยูนิกซ์มีคำสั่งอำนวยความสะดวกในการหาความแตกต่างได้แก่ cmp, diff, comm เป็นต้น.

สำรวจความแตกต่างแบบง่าย ๆ

คำสั่งที่ใช้สำรวจความแตกต่างระหว่างไฟล์สองไฟล์อย่างง่ายคือคำสั่ง cmp. คำสั่ง cmp จะตรวจสอบความแตกต่างระหว่างไฟล์สองไฟล์, ไบต์ต่อไบต์.

cmp อ้างอิงหน้า 385

cmp ย่อมาจากคำว่า compare.

ตัวอย่างที่ 4.38: ตรวจสอบความแตกต่างระหว่างไฟล์แบบง่าย ๆ.

```
$ cat hello_01.sh↵
#!/bin/sh
echo What is you name?
echo -n "> "
read n
echo Hello $n
$ cat hello_02.sh↵
```

```
#!/bin/sh
echo What is you name?
echo -n "> "
read name
echo Hello $name!
$ cmp hello_01.sh hello_02.sh↵
hello_01.sh hello_02.sh differ: char 53, line 4
```

คำสั่ง cmp จะบอกว่าไฟล์ที่ต้องการเปรียบเทียบนั้นมีความแตกต่างกันหรือไม่. ถ้าไม่มีความแตกต่างก็จะไม่แสดงผลอะไร. ถ้ามีความแตกต่างก็จะส่งข้อความออกทาง stdout และแสดงตำแหน่งไบต์และบรรทัดตรวจพบความแตกต่างที่เจอเป็นที่แรก.

เนื่องจากระบบปฏิบัติการตระกูลยูนิกซ์ไม่มีการแยกแยะไฟล์ไบนารีกับไฟล์เท็กซ์, คำสั่ง cmp ก็เหมือนกับคำสั่งอื่น ๆคือสามารถใช้ได้กับไฟล์ไบนารีด้วย.

diff อ้างอิงหน้า 386

แสดงความแตกต่างบรรทัดต่อบรรทัด

คำสั่ง diff ใช้แสดงความแตกต่างระหว่างไฟล์สองไฟล์หรือไดเรกทอรีสองไดเรกทอรี. ถ้าอาร์กิวเมนต์ของคำสั่งเป็นชื่อไฟล์, คำสั่ง diff ก็จะแสดงความแตกต่างของไฟล์ทั้งสองดังตัวอย่างต่อไปนี้.

diff ย่อมาจากคำว่า different.

ตัวอย่างที่ 4.39: การใช้คำสั่ง diff ดูความแตกต่างระหว่างไฟล์.

```
$ diff hello_01.sh hello_02.sh↵
4,5c4,5
< read n
< echo Hello $n
---
> read name
> echo Hello $name!
```

คำสั่ง diff จะแสดงเลขบรรทัดและเครื่องตัวอักษรย่อเช่น 2,3c2,3 หมายถึงข้อมูลบรรทัดที่ 2 ถึงบรรทัดที่ 3 เปลี่ยนไป. หลังจากนั้นก็จะแสดงบรรทัดที่แตกต่างของไฟล์ที่หนึ่งก่อนโดยเติมเครื่องหมายมากกว่านำหน้า, และแสดงบรรทัดที่แตกต่างของไฟล์ที่สองถัดมาตามลำดับ.

เนื่องจากคำสั่ง diff เป็นคำสั่งที่แสดงความแตกต่างหรือความเปลี่ยนแปลงระหว่างไฟล์สองไฟล์, ผู้พัฒนาซอฟต์เสรีมักจะใช้คำสั่ง diff ในการสร้าง *patch*. และผู้ที่ใช้ไฟล์ patch จะใช้คำสั่ง patch ในการเปลี่ยนรหัสต้นฉบับเดิมให้เป็นรหัสต้นฉบับใหม่ด้วยไฟล์ patch.

patch ▶
แพทช์. ไฟล์ที่มีเนื้อหาความแตกต่างระหว่างไฟล์ต้นฉบับก่อนเปลี่ยนแปลงกับไฟล์ที่แก้ไขแล้ว. ผู้พัฒนาซอฟต์แวร์จะสร้างไฟล์ patch ด้วยโปรแกรม diff และผู้ที่ต้องการใช้ไฟล์ patch นี้จะทำการ patch ไฟล์ดังกล่าวให้แก้ไขไฟล์ต้นฉบับที่ดั้งเดิมให้มีเนื้อหาเป็นเนื้อหาใหม่ที่แก้ไขแล้วด้วยโปรแกรม patch.

การสร้างไฟล์ patch ด้วยคำสั่ง diff, อาร์กิวเมนต์มักจะเป็นชื่อไดเรกทอรีและมักจะใช้ตัวเลือก -Naur.

ตัวอย่างที่ 4.40: การสร้างไฟล์ patch ด้วยคำสั่ง diff.

```
$ diff -Naur hello_01.sh hello_02.sh > hello_01.diff↵
$ cat hello_01.diff↵
--- hello_01.sh 2004-08-10 16:45:36.000000000 +0900
+++ hello_02.sh 2004-08-10 16:46:38.000000000 +0900
@@ -1,5 +1,5 @@
 #!/bin/sh
 echo What is you name?
 echo -n "> "
```

ตารางที่ 4.5: ตัวเลือกสำหรับ diff ที่ใช้ในการสร้างไฟล์ patch.

ตัวเลือก	ความหมาย
N	ถ้าในไดเรกทอรี่ที่เปรียบเทียบ, ถ้าในไดเรกทอรีหนึ่งมีไฟล์แต่ในไดเรกทอรีอีกที่ไม่มีไฟล์, ให้ถือว่าไฟล์นั้นเป็นไฟล์ใหม่.
a	แสดงความแตกต่างของไฟล์ไม่ว่าไฟล์นั้นจะเป็นไฟล์แบบไบนารี.
u	แสดงบรรทัดที่เหมือนกันด้วย. บรรทัดที่มีการตัดออกจากไฟล์เดิมจะมีเครื่องหมาย - หน้าบรรทัด. และบรรทัดที่มีการเพิ่มเติมจะมีเครื่องหมาย + หน้าบรรทัด.
r	สำหรับเปรียบเทียบไฟล์แบบ recursive. ใช้สำหรับเปรียนเทียบไฟล์และไดเรกทอรีย่อยในไดเรกทอรี.

```
-read n                                    ← บรรทัดที่ลบออกจากไฟล์ที่หนึ่ง
-echo Hello $n                             ← บรรทัดที่ลบออกจากไฟล์ที่หนึ่ง
+read name                                 ← บรรทัดที่เพิ่มเข้าไปจากไฟล์ที่สอง
+echo Hello $name!                         ← บรรทัดที่เพิ่มเข้าไปจากไฟล์ที่สร้าง
```

ดูส่วนที่เหมือนกันของไฟล์

ในทางกลับกันถ้าต้องดูส่วนที่เหมือนกันของไฟล์, ให้ใช้คำสั่ง comm

comm อ้างอิงหน้า 384

comm ย่อมาจาก common.

ตัวอย่างที่ 4.41: ใช้ comm ดูส่วนที่เหมือนกันของไฟล์.

```
$ comm hello_01.sh hello_02.sh↵
                #!/bin/sh
                echo What is you name?
                echo -n "> "
read n
echo Hello $n
        read name
        echo Hello $name!
```

ผลลัพธ์ของคำสั่งจะแบ่งเป็น 3 คอลัมน์โดยใช้ tab เป็นตัวแยกคอลัมน์. คอลัมน์แรก (ซ้ายมือ) จะแสดงข้อมูลที่มีเฉพาะไฟล์ที่หนึ่ง. คอมลัมภ์ที่สอง (ตรงกลาง) จะแสดงข้อมูลที่มีเฉพาะไฟล์ที่สอง. คอลัมน์ที่สาม (ขวามือ) จะแสดงข้อมูลร่วมที่เหมือนกันทั้งสองไฟล์.

คำสั่ง comm มีตัวเลือก -1, -2 และ -3 ให้ผู้ใช้สามารถระงับการแสดงผลของคอลัมน์ที่ต้องการได้. ดังนั้นถ้าต้องการข้อมูลส่วนที่เหมือนกันในไฟล์ของสองไฟล์ให้ใช้ตัวเลือก -12 ก็จะแสดงผลลัพธ์ที่เป็นคอลัมน์ที่สามอย่างเดียว.

4.4 การจัดการข้อมูลเท็กซ์และ regular expression

ในช่วงที่ผ่านมาจะสังเกตเห็นได้ว่าคำสั่งที่แนะนำไปเป็นคำสั่งที่กระทำกับกับข้อมูลในหน่วยของบรรทัดหรือไม่ก็คอลัมน์. ในช่วงนี้ยังคงแนะนำคำสั่งบางตัวที่ยังอยู่ในแพ็กเกจ textutils อยู่แต่เป็นคำสั่งที่เน้นเกี่ยวกับตัวข้อมูลเช่นการแก้ไขข้อมูล, หรือคำสั่งที่ใช้ regular expression.

4.4.1 การแก้ไขตัวอักษร

การเปลี่ยนตัวอักษรบางตัวให้เป็นตัวอักษรที่ต้องการ, หรือลบตัวอักษรที่ไม่ต้องการออกจากไฟล์มักใช้คำสั่ง `tr`. ตัวอย่างการใช้งานจริงเช่นการแปลงอักขระขึ้นบรรทัดใหม่ของไฟล์ที่ใช้ในระบบปฏิบัติการวินโดวส์ให้เป็นอักขระขึ้นบรรทัดใหม่ที่ใช้ในระบบปฏิบัติการยูนิกซ์หรือลินุกซ์.

tr อ้างอิงหน้า 392

ตัวอย่างที่ 4.42: การเปลี่ยนอักขระขึ้นบรรทัดใหม่จาก DOS ให้เป็น UNIX.

```
$ cat dosfile.txt↵
line1
line2
$ od -c dosfile.txt↵                                          ← ดูไฟล์โดยใช้ od
0000000   l   i   n   e   1  \r  \n   l   i   n   e   2  \r  \n
0000016
$ tr -d '\r' < dosfile.txt > unixfile.txt↵
$ od -c unixfile.txt↵
0000000   l   i   n   e   1  \n   l   i   n   e   2  \n
0000014
```

จากตัวอย่างเป็นการลบอักขระ `'\r'` อักษรที่สามารถระบุให้คำสั่ง `tr` เป็นอักษรที่พิมพ์ได้ด้วยแป้นพิมพ์ทั่วไปและอักษรที่ขึ้นต้นด้วย backslash (\) ที่แสดงในตารางที่ 4.6.

\r คืออักขระ carriage return.

ตารางที่ 4.6: อักษรที่แสดงด้วยเครื่องหมาย backslash (\).

สัญลักษณ์	ความหมาย
NNN	อักขระที่เขียนด้วยเลขฐานแปด.
\\	เครื่องหมาย backslash (\).
\a	audible BEL. เสียงกระดิ่ง (beep).
\b	backspace.
\f	form feed.
\n	new line.
\r	carriage return.
\t	แคร่ (tab).
\v	vertical tab (ปัจจุบันไม่ค่อยมีความหมายเพราะใช้น้อยมาก).

นอกจากการระบุตัวอักขระที่ต้องการเปลี่ยนหรือลบแล้วยังสามารถระบุประเภทของอักขระด้วยรูปแบบ `[:class:]`. *class* คือชื่อที่กำหนดไว้แล้วแทนประเภทของอักษรที่แสดงในตารางที่ 4.7.

ตารางที่ 4.7: ชื่อประเภทของอักขระ.

ชื่อประเภทอักขระ	คำอธิบาย
[:alnum:]	ตัวอักษร (ภาษาอักกฤษ) และตัวเลข.
[:alpha:]	ตัวอักษร.
[:blank:]	ช่องว่างเช่นช่องไฟหรือแคร่.
[:cntrl:]	อักขระควบคุม.
[:digit:]	ตัวเลข.
[:graph:]	อักขระที่แสดงผลทางหน้าจอได้, ไม่รวมช่องว่าง.
[:lower:]	ตัวอักษรตัวเล็ก (ภาษาอังกฤษ).
[:print:]	ตัวอักษรที่แสดงผลทางหน้าจอได้, รวมช่องว่าด้วย.
[:punct:]	เครื่องหมายวรรคตอน.
[:space:]	ช่องว่างทั้งทางแนวนอนและแนวตั้ง (บรรทัดว่างเปล่า)
[:upper:]	ตัวอักษรตัวใหญ่ (ภาษาอังกฤษ).
[:xdigit:]	เลขฐานสิบหก.

การใช้ชื่อประเภทของอักขระช่วยให้ทำงานได้คล่องขึ้นเช่นเมื่อต้องการเปลี่ยนชื่อไฟล์ที่เป็นอักษรตัวใหญ่ให้เป็นอักษรตัวเล็กสามารถทำได้ตามตัวอย่างต่อไปนี้.

ตัวอย่างที่ 4.43: เปลี่ยนชื่อไฟล์จากอักษรตัวใหญ่ให้เป็นตัวเล็กด้วยคำสั่ง `tr`.

```
$ ls↵
P7070001.JPG  P7070002.JPG  P7070004.JPG
$ for i in *↵
> do↵
> mv -v $i ‘echo $i | tr ’[:upper:]’ ’[:lower:]’‘↵
> done↵
‘P7070001.JPG’ -> ‘p7070001.jpg’
‘P7070002.JPG’ -> ‘p7070002.jpg’
‘P7070004.JPG’ -> ‘p7070004.jpg’
```

ในบางกรณีที่เราต้องการเปลี่ยน tab ให้เป็น space สามารถใช้คำสั่งเฉพาะสำหรับงานนี้ได้แก่คำสั่ง `expand`. ในทางกลับกันถ้าต้องการเปลี่ยน space ให้เป็น tab ให้ใช้คำสั่ง `unexpand`. โดยปรกติคำสั่ง `expand` จะเปลี่ยน tab ทุกตัวให้เป็น space แปดตัว. คำสั่ง `unexpand` จะเปลี่ยน space ที่อยู่ต้นบรรทัดให้เป็น tab และจะถือว่า space แปดตัวคือ tab หนึ่งตัว.

4.4.2 Regular expression

บางครั้งเรียกย่อ ๆว่า regex หรือ RE. ต่อไปนี้จะใช้ RE แทนคำว่า regular expression.

Regular expression คือวิธีการแสดงคำ, สายอักขระ (character string) ทั่วไปด้วยแบบอย่าง (pattern) โดยใช้อักขระและไวยกรณ์ที่กำหนดไว้. RE มีบทบาทสำคัญในการประมวลข้อมูลเท็กซ์. เราสามารถใช้ RE หาคำหรือบรรทัดที่ต้องการในไฟล์ด้วยคำสั่ง `egrep`. ใช้ RE เปลี่ยนคำที่ต้องการเป็นคำอื่น ๆด้วยคำสั่ง `sed`. ใช้ RE ในเพจเจอร์ `less` หาคำที่ต้องการได้อย่างมีประสิทธิภาพและรวดเร็ว. RE มีบทบาทมากในลินุกซ์เพราะคำสั่งหลายคำสั่งสามารถใช้ RE, และเป็นเรื่องที่ต้องรู้สำหรับผู้ที่ต้องการใช้ลินุกซ์อย่างจริงจังไม่ว่าจะเป็นผู้ดูแลระบบหรือนักพัฒนาซอฟต์แวร์.

เดิมที RE เริ่มใช้กันในระบบปฏิบัติการยูนิกซ์ในโปรแกรมบรรณาธิกรณ์ `ed`. เนื่องจากสมรรถภาพสูงของ RE ที่สามารถแทนคำต่าง ๆได้ตามที่ต้องการจึงมีการนำไปใช้ในโปรแกรมคำสั่งอื่น ๆอย่างแพร่หลายในเวลาต่อมา. ในปัจจุบัน RE ยังนิยมใช้ในภาษาคอมพิวเตอร์ทั้งแบบอินเทอร์เพรเตอร์เช่น `perl`, `python`, `ruby` และใช้ในภาษาคอมพิวเตอร์แบบคอมไพเลอร์เช่น `c++` ด้วย. อักขระที่ใช้ใน RE เปลี่ยนไปจากเดิมแตกต่างจากที่ใช้ในบรรณาธิกรณ์ `ed`. เราจะเรียก RE ที่ใช้ในบรรณาธิกรณ์ `ed` ว่าเป็น RE แบบพื้นฐาน (basic regular expression) และเรียก RE ที่นิยามโดยมาตรฐาน POSIX 1003.2 ว่า RE แบบเสริม (extended regular expression).

มาตรฐาน POSIX 1003.2 ได้กำหนดอักขระที่ใช้ใน re ดังนี้.

- `.` แทนอักขระใด ๆก็ได้หนึ่งตัว.
  ตัวอย่างเช่น `a.c` แทน aab, abc, acc, a c, a.c, apc ฯลฯ.
- `^` แสดงตำแหน่งต้นบรรทัด. ถ้าตัวอักษรนี้อยู่ในวงเล็บ `[]` จะมีความหมายตรรกะปฏิเสธ (not).
  ตัวอย่างเช่น `^abc` หมายถึงบรรทัดที่ขึ้นต้นด้วย abc. `a[^b]c` หมายถึง aac, acc, a c, alc ฯลฯ.
- `$` แสดงตำแหน่งท้ายบรรทัด.
  ตัวอย่างเช่น `abc$` หมายถึงบรรทัดที่ลงท้ายด้วย abc.
- `?` บอกจำนวนของอักขระที่อยู่หน้าเครื่องหมายนี้ว่าเป็น 0 (ไม่มี) หรือ 1 (มี), คือมีหรือไม่มีก็ได้.
  ตัวอย่างเช่น `ab?c` หมายความว่าจะมีตัวอักษร b อยู่ระหว่างตัวอักษร a และ c หรือไม่ก็ได้. ถ้ามีตัวอักษร b, สามารถมีได้ตัวเดียวเท่านั้น. กล่าวคือ `ab?c` ใช้แทน ac, abc แต่ไม่ใช่ abbc.
- `*` บอกจำนวนอักขระที่อยู่หน้าเครื่องหมายนี้ว่ามากกว่าหรือเท่ากับ 0.
  ตัวอย่างเช่น `ab*c` หมายความว่าจะมีตัวอักษร b อยู่ระหว่างตัวอักษร a และ c หรือไม่ก็ได้. ถ้ามีตัวอักษร b, จะมีกี่ตัวก็ได้. กล่าวคือ `ab*c` ใช้แทน ac, abc, abbc, abbbc ฯลฯ.
- `+` บอกจำนวนอักขระที่อยู่หน้าเครื่องหมายนี้ว่ามีจำนวนอย่างน้อย 1 ตัว.
  ตัวอย่าง `ab+c` หมายความว่าต้องมีอักษร b อยู่ระหว่างตัวอักษร a และ b อย่างน้อยหนึ่งตัวเช่น abc, abbc, abbbc ฯลฯ.
- `\` ใช้เป็น escape character.

ตัวอย่างเช่น `a\.c` คือ “a.c”. `a\?c` คือ “a?c”.

`|` สัญลักษณ์หรือ.
ตัวอย่างเช่น `ab|bc` แทนคำที่มี ab หรือ bc อยู่ในนั้นเช่น ab, bc, abc, abs, abbb ฯลฯ.

`[]` ใช้แสดงกลุ่มหรือประเภทของอักขระ. ตารางที่ 4.7 แสดงประเภทของอักขระ (character class) ที่มาตร POSIX 1003.2 กำหนดไว้.
ตัวอย่างเช่น `[abc]` หมายถึงอักขระตัวเดียวที่เป็น a หรือ b หรือ c. ถ้าเป็นอักขระที่ระบุในเครื่องหมาย `[]` เป็นอักขระที่เรียงติดกัน, สามารถเขียนย่อได้โดยใช้เครื่องหมาย `-` เช่น `[abc]` มีความหมายเหมือนกับ `[a-c]`. `[[:digit:]]` หมายถึงอักขระที่เป็นตัวเลขได้แก่ 0 ถึง 9.

`()` ใช้จับกลุ่มของอักขระ.
ตัวอย่างเช่น `(a|b)c` จะหมายถึง a หรือ b แล้วตามด้วย c เช่น ab, cb แต่ `a|bc` จะหมายถึง a หรือ bc.

นอกจากการบอกจำนวนอักขระด้วย `?`, `*` และ `+` แล้วเรายังสามารถกำหนดจำนวนการซ้ำของอักขระได้ละเอียดยิ่งขึ้นได้แก่

$\{n\}$ บอกจำนวนตัวอักขระที่อยู่หน้า $\{n\}$ ว่ามีจำนวน n ตัวพอดี.
ตัวอย่างเช่น `ab{3}c` ใช้แทน abbbc.

$\{n,\}$ บอกจำนวนตัวอักขระที่อยู่หน้า $\{n\}$ ว่ามีจำนวน n ตัวขึ้นไป.
ตัวอย่างเช่น `ab{3,}c` ใช้แทน abbbc, abbbbc, abbbbbc ฯลฯ.

$\{n,m\}$ บอกจำนวนตัวอักขระที่อยู่หน้า $\{n,m\}$ ว่ามีจำนวนอย่างน้อย n แต่ไม่เกิน m ตัว.
ตัวอย่างเช่น `ab{3,4}c` ใช้แทน abbbc และ abbbbc.

re แบบพื้นฐานต่างจาก re แบบเสริมที่ไม่สามารถใช้อักขระบางตัวที่ re แบบเสริมใช้ได้เช่น `|`, `+`, `{}` เป็นต้น.

อ่านรายอะเอียดเพิ่มเติมได้จาก `regexp(7)`

4.4.3 หาคำ, บรรทัดที่ต้องการในไฟล์

การหาคำหรือบรรทัดที่ต้องการในไฟล์เกิดบ่อยครั้งเมื่อผู้ใช้ต้องการหาข้อมูลในไฟล์แต่ไม่รู้ว่าอยู่ส่วนไหน, หรือมีไฟล์หลายไฟล์ไม่รู้ว่าคำที่ต้องการหาอยู่ในไฟล์ไหน.

คำสั่ง `grep` เป็นคำสั่งที่จะแสดงบรรทัดที่มีคำหรือแบบอย่าง (pattern) ที่ต้องมาแสดงทางหน้าจอ. คำสั่ง `grep` ที่ใช้ในลินุกซ์เป็นโปรแกรมจากโครงการ GNU และมี 3 แบบได้แก่

grep อ้างอิงหน้า 387

- `grep` ใช้หาคำหรือแบบอย่างที่อยู่ในไฟล์. แบบอย่างที่สามารถใช้ได้คือ re แบบพื้นฐาน.
- `egrep` ใช้หาคำหรือแบบอย่างที่อยู่ในไฟล์, และสามารถใช้ re แบบเสริมได้. อักษร “e” ที่อยู่หน้าชื่อคำสั่ง “grep” หมายถึง extended regular expression.

egrep เป็นเพียงแค่ซอฟต์ลิงค์ไปหาคำสั่ง grep เท่านั้น. ในความเป็นจริงแล้วคือการใช้ grep กับตัวเลือก -E ซึ่งเป็นการระบุให้ใช้ re แบบเสริมได้.

- fgrep ใช้หาคำที่อยู่ในไฟล์. ถ้าในคำที่ต้องการหามีอักขระที่ใช้ re, fgrep จะถือว่าอักขระนั้นมีความหมายตามที่เห็น, คือไม่ใช่ re. คำสั่ง grep และตัวเลือก -F จะให้ผลเหมือนกับคำสั่ง fgrep.

การใช้คำสั่งนี้แบบง่ายที่สุดคือกำหนดคำที่ต้องการหาเป็นอาร์กิวเมนต์, และชื่อไฟล์เป็นอาร์กิวเมนต์ตัวถัดไป. ในตัวอย่างเป็นการใช้ grep เพื่อหาคำว่า fail ที่อยู่ในไฟล์ /var/log/messages.

ไฟล์ /var/log/messages เป็นไฟล์ที่เก็บ log ของระบบ. หากมีข้อผิดพลาดที่เกิดขึ้นกับระบบปฏิบัติการหรือโปรแกรมสำคัญต่าง ๆก็จะถูกบันทึกไว้ในไฟล์นี้. ในตัวอย่างจะอ่านไฟล์นี้ด้วย root.

ตัวอย่างที่ 4.44: การใช้ grep เบื้องต้น.

```
# grep fail /var/log/messages↵
...
Jun  6 19:18:39 toybox FAT: Directory bread(block 274) failed
Jun  6 19:18:39 toybox FAT: Directory bread(block 275) failed
Jun  6 19:18:39 toybox FAT: Directory bread(block 276) failed
Jun  6 19:18:39 toybox FAT: Directory bread(block 277) failed
Jun  6 19:18:39 toybox FAT: Directory bread(block 278) failed
Jun 13 17:00:28 toybox hub 1-0:1.0: connect-debounce failed, port 2 disabled
Jun 15 23:44:00 toybox dictd[11844]: :D: foldoc "failure-directed testing" 1
Jun 26 00:53:39 toybox su(pam_unix)[27228]: authentication failure;
logname=poonlap uid=1000 euid=0 tty=pts/0 ruser=poonlap rhost=  user=root
Jun 26 00:53:41 toybox su[27228]: pam_authenticate: Authentication failure
Jul 11 12:36:08 toybox gdm(pam_unix)[7437]: authentication failure;
logname= uid=0 euid=0 tty=:0 ruser= rhost=
Jul 18 19:00:01 toybox su(pam_unix)[3314]: authentication failure;
logname=poonlap uid=1000 euid=0 tty=pts/2 ruser=poonlap rhost=  user=root
Jul 18 19:00:03 toybox su[3314]: pam_authenticate: Authentication failure
```

-i เป็นตัวเลือกแบบสั้นของ --ignore-case

ในกรณีที่ไม่แน่ใจว่าคำที่ต้องการหาเขียนด้วยตัวใหญ่หรือตัวเล็ก, ให้ใช้ตัวเลือก -i เพื่อให้คำสั่งไม่แยกแยะความแตกต่างระหว่างอักษรตัวใหญ่กับตัวเล็ก. ถ้าหาคำว่า “fail” ก็จะได้ “Fail”, “FAIL” ฯลฯ ด้วย.

-v เป็นตัวเลือกแบบสั้นของ --invert-match

ในทางกลับกัน, ถ้าไม่ต้องการบรรทัดที่มีคำที่ระบุให้ใช้ตัวเลือก -v กรองบรรทัดที่มีคำนั้นออกไป. ตัวอย่างเช่นกรองข้อมูลจากผลลัพธ์ของตัวอย่างที่แล้วโดยไม่เอาบรรทัดที่มีคำว่า “FAT”.

ตัวอย่างที่ 4.45: ใช้ grep กรองข้อมูลที่ไม่ต้องการ.

```
# grep fail /var/log/messages | grep -v FAT↵
...
Jun 13 17:00:28 toybox hub 1-0:1.0: connect-debounce failed, port 2 disabled
Jun 15 23:44:00 toybox dictd[11844]: :D: foldoc "failure-directed testing" 1
Jun 26 00:53:39 toybox su(pam_unix)[27228]: authentication failure;
logname=poonlap uid=1000 euid=0 tty=pts/0 ruser=poonlap rhost=  user=root
Jun 26 00:53:41 toybox su[27228]: pam_authenticate: Authentication failure
Jul 11 12:36:08 toybox gdm(pam_unix)[7437]: authentication failure;
logname= uid=0 euid=0 tty=:0 ruser= rhost=
Jul 18 19:00:01 toybox su(pam_unix)[3314]: authentication failure;
```

```
logname=poonlap uid=1000 euid=0 tty=pts/2 ruser=poonlap rhost=  user=root
Jul 18 19:00:03 toybox su[3314]: pam_authenticate: Authentication failure
```

ให้พิจารณาผลลัพธ์ของตัวอย่างการใช้ grep หาคำที่มีเครื่องหมาย - นำหน้าตัวอักษรต่อไปนี้.

ตัวอย่างที่ 4.46: การใช้ grep หาคำที่มีเครื่องหมาย - นำหน้า.

```
$ ps -ef | grep '-bash'↵                    ← คำสั่ง grep ถือว่า -bash เป็นตัวเลือก
Usage: grep [OPTION]... PATTERN [FILE]...
Try 'grep --help' for more information.
$ ps -ef | grep -e '-bash'↵                              ← หรือ grep -- '-bash'
poonlap   7777  7775  0 13:30 pts/0    00:00:00 -bash
poonlap   8087  7775  0 13:47 pts/1    00:00:00 -bash
poonlap   8092  7775  0 13:47 pts/2    00:00:00 -bash
poonlap  12751  8087  0 16:33 pts/1    00:00:00 grep -e '-bash'
```

การใช้ grep หาคำ -bash ครั้งแรกจะล้มเหลวและเกิด error เพราะจะถือว่า -bash เป็นตัวเลือกถึงแม้ว่าจะใช้ quote แล้วก็ตาม. ในกรณีเช่นนี้ grep ให้ใช้ตัวเลือก -e เพื่อระบุคำหรือแบบอย่างให้ชัดเจน.

โปรเซสที่มีเครื่องหมาย - นำหน้ามีความหมายว่าโปเซสนั้นเป็นเชลล์ล็อกอิน.

ถ้าเรามีคำที่ต้องการหามากกว่า 2 คำ, และคำเหล่านั้นอาจจะไม่อยู่ในบรรทัดเดียวกัน ให้ใช้ตัวเลือก -e ระบุคำหรือแบบอย่างที่ต้องการ, คำต่อคำ.

-e *pattern* เป็นตัวเลือกแบบสั้นของ --regexp=*pattern*. อย่างสับสนระหว่างตัวเลือก -e กับตัวเลือก -E ซึ่งมีความหมายต่างกัน.

ตัวอย่างที่ 4.47: หาคำตั้งแต่สองคำที่อยู่ในไฟล์.

```
$ ps -ef | grep -e tty1 -e pts/2↵
root      7531     1  0 13:27 tty1     00:00:00 /sbin/agetty 38400 tty1 linux
poonlap   8092  7775  0 13:47 pts/2    00:00:00 -bash
poonlap   1453  8092  0 14:55 pts/2    00:00:01 /usr/lib/mozilla/mozilla-bin
poonlap   1478  1453  0 14:55 pts/2    00:00:00 /usr/lib/mozilla/mozilla-bin
root      2779  8092  0 15:05 pts/2    00:00:00 su -
root      2782  2779  0 15:05 pts/2    00:00:00 -bash
poonlap   3221  8087  0 15:19 pts/1    00:00:00 grep -e tty1 -e pts/2
```

จากตัวอย่างคือการหาบรรทัดที่มีคำว่า tty1 หรือ pts/2. หรือจะใช้ re ในการหาซึ่งจะให้ผลเหมือนกัน.

ตัวอย่างที่ 4.48: การใช้ re หาคำสองคำด้วยข้อแม้หรือ (OR).

```
$ ps -ef | egrep 'tty1|pts/2'↵              ← หรือ ps -ef | grep -E 'tty1|pts/2'
root      7531     1  0 13:27 tty1     00:00:00 /sbin/agetty 38400 tty1 linux
poonlap   8092  7775  0 13:47 pts/2    00:00:00 -bash
poonlap   1453  8092  0 14:55 pts/2    00:00:01 /usr/lib/mozilla/mozilla-bin
poonlap   1478  1453  0 14:55 pts/2    00:00:00 /usr/lib/mozilla/mozilla-bin
root      2779  8092  0 15:05 pts/2    00:00:00 su -
root      2782  2779  0 15:05 pts/2    00:00:00 -bash
poonlap   6622  8087  0 15:34 pts/1    00:00:00 egrep tty1|pts/2
```

หาคำที่ต้องการว่าอยู่ในไฟล์ไหน

เรามาลองดูตัวอย่างการใช้ grep ช่วยหาคำที่มีอยู่ในไฟล์โดยที่เราไม่รู้ว่าคำที่ต้องการหานั้นอยู่ในไฟล์ไหน. สมมติว่าไดเรกทอรีที่ทำงานอยู่คือ /usr/lib/mozilla. ไดเรกทอรีนี้เป็นไดเรกทอรีที่เก็บไฟล์ต่าง ๆที่จำเป็นสำหรับเบราเซอร์ Mozilla มีทั้งไฟล์และไดเรกทอรีย่อยอยู่มากมายในไดเรกทอรีนี้. สมมติว่าเราต้องการหาว่ามีไฟล์ที่อะไรบ้างที่เกี่ยวข้องกับภาษาไทย, ก็อาจจะใช้แบบอย่างที่ต้องการหาเป็นคำว่า “thai”.

ตัวอย่างที่ 4.49: หาไฟล์ที่มีคำที่ต้องการ.

```
$ pwd↵
/usr/lib/mozilla
$ grep -ri thai .↵                                    ← . เป็นการระบุไดเรกทอรีปัจจุบัน
Binary file ./chrome/en-US.jar matches                ← บอกว่าเป็นไฟล์แบบไบนารี
Binary file ./chrome/comm.jar matches
... ย่นย่อ ...
./res/charsetData.properties:x-thaittf-0.LangGroup              = th
./res/fonts/fontEncoding.properties:# Thai TTFs
./res/fonts/fontEncoding.properties:# code points used : Unicode Thai block +
about 10 PUA code points in U+F700,
... ย่นย่อ ...
```

จากตัวอย่างข้างบน, คำสั่ง grep และตัวเลือก -r จะหาคำว่า “thai” ที่อยู่ใต้ไดเรกทอรีที่ทำงานทั้งหมด. และตัวเลือก -i จะไม่แยกแยะตัวอักษรตัวใหญ่หรือตัวเล็ก. เนื่องจากเป็นการใช้ grep หาคำในไฟล์หลายไฟล์, ผลลัพธ์ที่ได้จะแสดงชื่อไฟล์และนำหน้าตามด้วยบรรทัดที่มีคำที่ต้องการหา. ผลลัพธ์จะดูลำบากเพราะมีข้อมูลมากเกินไป. ถ้าต้องการดูแค่ชื่อไฟล์แต่ไม่ต้องการให้แสดงบรรทัดที่มีคำนั้นอยู่ให้ใช้ตัวเลือก -l และหลังจากนั้นค่อยดูเนื้อหาในแต่ละไฟล์ทีหลังก็ได้.

-r เป็นตัวเลือกแบบสั้นของ --recursive. ตัวเลือก -R มีความหมายเหมือนกัน.

-l เป็นตัวเลือกย่อของ --files-with-matches.

ตัวอย่างที่ 4.50: หาไฟล์ที่มีคำที่ต้องการโดยแสดงแค่ชื่อไฟล์.

```
$ grep -lri thai .↵
./chrome/en-US.jar
./chrome/comm.jar
... ย่นย่อ ...
./res/fonts/fontEncoding.properties
... ย่นย่อ ...
```

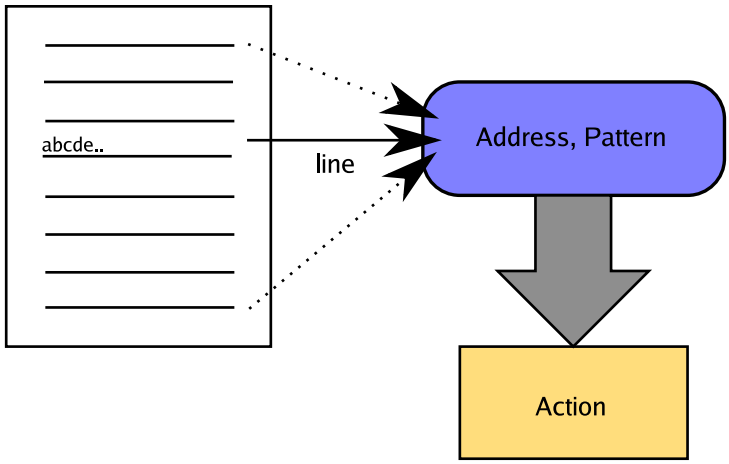
ตัวอย่างการหาไฟล์ที่มีคำที่ต้องการอยู่นี้มีประโยชน์เช่นเวลามีรหัสต้นฉบับของโปรแกรมและต้องการแก้ไขโปรแกรมนั้น. บางครั้งอาจจะเริ่มต้นโดยการหาชื่อฟังก์ชัน, คีย์เวิร์ดว่าอยู่ในไฟล์ไหน. จากนั้นค่อยดูเนื้อหาเป็นไฟล์ ๆไปว่าไฟล์ไหนเป็นไฟล์ที่น่าจะแก้ไข.

4.5 sed และ awk

ในหน้าที่ 162 ได้แนะนำการแก้ไขไฟล์แบบง่าย ๆด้วยคำสั่ง tr ไปแล้ว. การแก้ไขไฟล์เท็กซ์โดยการใช้บรรทัดคำสั่งแบบนี้มีข้อดีที่รวดเร็วและสามารถเขียนเป็นสคริปต์ให้

ทำงานอัตโนมัติได้. กล่าวคือเหมาะสำหรับการ*ประมวลผลแบบ batch (batch processing)* และแก้ไขไฟล์หลาย ๆไฟล์พร้อม ๆกัน. สำหรับการทำงานที่ไม่ได้ประโยชน์จากการใช้บรรทัดสั่งตามที่ได้กล่าวไปแล้วก็ให้ใช้บรรณาธิกรณ์ในการแก้ไข.

batch ▶ การประมวลผลข้อมูลโดยรวบรวมกระทำตามลำดับที่กำหนดไว้.



รูปที่ 4.6: การทำงานของโปรแกรมคำสั่ง sed และ awk.

ในช่วงนี้จะแนะนำโปรแกรมคำสั่ง 2 อย่างที่ใช้ในการเปลี่ยนแปลงแก้ไขไฟล์เท็กซ์ได้ดีกว่าคำสั่ง tr. คำสั่งที่จะแนะนำนี้ได้แก่ sed และ awk ซึ่งเป็นภาษาคอมพิวเตอร์แบบหนึ่งที่ยอมรับว่ามีประโยชน์ในการประมวลผลไฟล์เท็กซ์แล้วมักใช้ร่วมกับเชลล์สคริปต์.

4.5.1 sed

sed คือ *stream editor* หมายถึงโปรแกรมคำสั่งที่สามารถแก้ไขข้อมูลได้โดยรับข้อมูลเป็นสายอักษร (stream) บรรทัดต่อบรรทัดแล้วประมวลผลตามคำสั่งที่เตรียมไว้. เนื่องจาก sed เป็นโปรแกรมที่ซับซ้อนถึงขั้นมีหนังสือเฉพาะสำหรับคำสั่งนี้ [32], ในที่นี่จะแนะนำการใช้งานเบื้องต้นเท่านั้น.

คำสั่ง sed มักใช้อยู่เชลล์สคริปต์ในรูปแบบ

```
sed [-n] [-r] {-e script | -f script_file} [filename]
```

ตัวเลือก -e หมายถึงรับคำสั่ง (*script*) ของ sed จากอาร์กิวเมนต์เชลล์. สคริปต์ที่ว่านี้จะมีรูปแบบเป็น [address]command โดยที่ address คือตำแหน่งบรรทัดที่ต้องการประมวลผล, และ command คือคำสั่งของ sed ที่ต้องการกระทำกับข้อมูลบรรทัดนั้น. โดยปรกติจะใช้เครื่องหมาย quote คร่อมสคริปต์เพื่อไม่ให้เชลล์ตีความหมายในกรณีที่มีอักษรพิเศษสำหรับเชลล์. *filename* คือไฟล์ที่ต้องการแก้ไขหรือถ้าไม่ระบุก็จะรับข้อมูลจาก stdin. ตัวเลือก -r ใช้สำหรับระบุ RE แบบเสริม.

-r ย่อมาจาก --regexp-extended

คำสั่งแรกที่จะแนะนำคือ p ซึ่งจะแสดงข้อมูลบรรทัดต่อบรรทัด. simple.html HTML .

ตัวอย่างที่ 4.51: ใช้ sed

```
$ cat -n simple.html↵
     1  <html>
     2  <head>
     3  <title>Simple HTML</title>
     4  </head>
     5  <body>
     6  This is a basic simple HTML file.
     7  <!-- This line is a comment -->
     8  </body>
     9  </html>
$ sed -e 'p' simple.html↵
<html>
<html>
<head>
<head>
<title>Simple HTML</title>
<title>Simple HTML</title>
</head>
</head>
<body>
<body>
This is a basic simple HTML file.
This is a basic simple HTML file.
<!-- This line is a comment -->
<!-- This line is a comment -->
</body>
</body>
</html>
</html>
```

คำสั่ง sed จะมีพื้นที่ในหน่วยความจำสำหรับรับข้อมูลเข้าบรรทัดต่อบรรทัด. พื้นที่หน่วยความจำนี้เรียกว่า pattern space เป็นพื้นที่สำหรับประมวลผลตามที่ได้รับคำสั่ง. โดยปรกติแล้วคำสั่ง sed จะแสดง pattern space ที่ได้รับการประมวลแล้วออกทาง stdout โดยปริยาย. เนื่องจากเราใช้คำสั่ง p, คำสั่ง sed จึงแสดงผลที่อยู่ใน pattern space อีกครั้ง.

ในกรณีที่ใช้คำสั่ง p มักจะใช้ตัวเลือก -n เพื่อระงับการแสดง pattern space เพื่อไม่ให้แสดงผลสองครั้ง.

การระบุตำแหน่งบรรทัดขึ้นพื้นฐาน

การระบุตำแหน่งบรรทัดแบบง่ายที่สุดคือการระบุหมายเลขบรรทัด. เช่นถ้าต้องการแสดงบรรทัดที่หนึ่งของข้อมูล, ให้สั่งคำสั่ง 1p. 1 คือบรรทัดที่ 1, p คือแสดงบรรทัด (print). การใช้ sed จะใช้ตัวเลือก -n ด้วยเพื่อไม่แสดง pattern space โดยปริยาย.

ผู้อ่านลองไม่ใช้ตัวเลือก -n ดูเองว่าจะเกิดอะไรขึ้น.

ตัวอย่างที่ 4.52: ใช้ sed แสดงบรรทัดที่ต้องการ.

```
$ sed -ne '1p' simple.html↵
<html>
```

การแสดงบรรทัดแบบต่อเนื่องเช่นบรรทัดที่ 1 ถึง 4 ให้ใช้เครื่องหมาย comma (,) คั่นระหว่างเลขบรรทัด.

ตัวอย่างที่ 4.53: ใช้ sed แสดงบรรทัดที่ต้องการแบบต่อเนื่อง.

```
$ sed -ne '1,4p' simple.html↵
<html>
<head>
<title>Simple HTML</title>
</head>
```

การแสดงบรรทัดแบบต่อเนื่องยังสามารถจำนวนบรรทัดที่ต้องแสดงได้ด้วยเช่น `2,+3p` หมายถึงการแสดงบรรทัดที่ 2 และบรรทัดถัดไปอีก 3 บรรทัด.

ในกรณีทั่วไปเรามักจะระบุบรรทัดที่ต้องการประมวลผลโดยระบุคำที่อยู่ในบรรทัดนั้น. ตัวอย่างเช่นถ้าต้องการแสดงผลบรรทัดที่เป็นหมายเหตุ, เราไม่สามารถระบุด้วยหมายเลขบรรทัดได้เพราะไม่สามารถรู้ล่วงหน้าว่าบรรทัดที่เป็นหมายเหตุนั้นเป็นบรรทัดที่เท่าไร. ในกรณีนี้จะใช้ regular expression ระบุตำแหน่งบรรทัด.

ตัวอย่างเช่นต้องการแสดงบรรทัดตั้งแต่ `<body>` ถึง `</body>` สามารถทำได้ดังนี้.

ตัวอย่างที่ 4.54: การระบุตำแหน่งบรรทัดด้วย regular expression

```
$ sed -ne '/<body>/,/<\/body>/p' simple.html↵
<body>
This is a basic simple HTML file.
<!-- This line is a comment -->
</body>
```

ส่วนที่เป็น regular expression จะคร่อมด้วยเครื่องหมาย slash (`/`).

ตารางที่ 4.10: การระบุตำแหน่งบรรทัดของ sed เบื้องต้น.

ตำแหน่งบรรทัด	ความหมาย
N	บรรทัดที่ *N*.
N1,N2	บรรทัดที่ *N1* ถึงบรรทัดที่ *N2*.
N,+n	บรรทัดที่ *N* และบรรทัดถัดไปอีก *n* บรรทัด.
$	บรรทัดสุดท้าย.
/re/	บรรทัดที่ตรงกับ regular expression (*re*) ที่กำหนด.

คำสั่งพื้นฐานของ sed

คำสั่งแสดงบรรทัดได้แก่คำสั่ง `p` ซึ่งได้แสดงตัวอย่างไปแล้ว. คำสั่งมักจะใช้ร่วมกับตัวเลือก `-n` เพื่อให้แสดงเฉพาะบรรทัดที่ต้องการ. คำสั่งลบบรรทัดได้แก่คำสั่ง `d` ใช้ลบบรรทัดที่ต้องการ.

ตารางที่ 4.11: คำสั่งพื้นฐานของ sed.

คำสั่ง	คำอธิบาย
p	แสดงข้อมูลที่อยู่ในบรรทัดบรรทัด.
d	ลบบรรทัด.
i	แทรกบรรทัด.
a	เพิ่มบรรทัด.
c	เปลี่ยนบรรทัด.
s/*re*/*word*/[g]	สับเปลี่ยนคำโดยระบุคำที่ต้องสับเปลี่ยนเป็น regular expression ด้วยคำ *word* ที่ระบุ.

ตัวอย่างที่ 4.55: การใช้ sed ลบบรรทัดที่ต้องการ.

```
$ sed -e '/<!--.*-->/'↵
<html>
<head>
<title>Simple HTML</title>
</head>
<body>
This is a basic simple HTML file.
</body>
</html>
```

จากตัวอย่างเป็นการลบบรรทัดที่เป็นหมายเหตุโดยใช้ regular expression <!--.*-->. . จะแทนตัวอักขระใด ๆและ * เป็นการบอกว่าอักขระสามารถมีต่อไปได้เรื่อยหรือไม่มีก็ได้.

การแทรกบรรทัดด้วยคำสั่ง sed มีสองแบบคือแทรกบรรทัดก่อนหน้าบรรทัดที่ต้องการ (insert), และแทรกบรรทัดหลังบรรทัดที่ต้องการ (append). การแทรกบรรทัดก่อนหน้าบรรทัดที่ต้องการจะใช้คำสั่ง i ที่แสดงในตัวอย่างต่อไปนี้.

ตัวอย่างที่ 4.56: การใช้ sed แทรกบรรทัดใหม่.

```
$  sed -e '/<\/head>/i\↵
> <meta http-equiv="Content-Type" content="text/html">' simple.html↵
<html>
<head>
<title>Simple HTML</title>
<meta http-equiv="Content-Type" content="text/html">
</head>
<body>
This is a basic simple HTML file.
<!-- This line is a comment -->
</body>
</html>
```

จากตัวอย่างเป็นการแทรกบรรทัด `<meta http-equiv="Content-Type" content="text/html">` ก่อนบรรทัด `</head>`. เครื่องหมาย backslash (\) ที่เขียนหลังคำสั่ง `i` เป็น escape sequence เพื่อเขียนสิ่งที่ต้องการแทรกในบรรทัดใหม่แทนที่จะเขียนยาว ๆติดกันไป.

การแทรกบรรทัดหลังบรรทัดทีต้องการทำได้ด้วยคำสั่ง `a` ซึ่งมีการใช้งานคล้ายกับคำสั่ง `i`.


ให้ผู้อ่านลองปฏิบัติด้วยตัวเอง.

การเปลี่ยนข้อมูลทั้งบรรทัดจะใช้คำสั่ง `c` ซึ่งย่อมาจากคำว่า change หมายถึงลบข้อมูลในบรรทัดนั้นออกแล้วแทรกข้อมูลใหม่ที่ป้อนให้เข้าไปในบรรทัดนั้นแทน.

ตัวอย่างที่ 4.57: การเปลี่ยนข้อมูลทั้งบรรทัดด้วย sed.

```
$ sed -e '/This is a/c\↵
> Hello World!' simple.html↵
<html>
<head>
<title>Simple HTML</title>
</head>
<body>
Hello World!
<!-- This line is a comment -->
</body>
</html>
```

จะเห็นว่าบรรทัดที่มีคำว่า `This is a` จะถูกแทนด้วย `Hello World!` ตามที่ต้องการ.

`sed` เป็นคำสั่งที่นิยมในใช้ในการสับเปลี่ยนคำ (substitute) มากที่สุดเพราะสามารถระบุส่วนที่ต้องการสับเปลี่ยนกับคำที่ต้องการได้ด้วย regular expression, และคำสั่ง `sed` ประมวลผลได้รวดเร็ว. การสับเปลี่ยนคำด้วย `sed` มีรูปแบบดังนี้.

```
s/re/replacement/[g]
```

`s` เป็นส่วนเริ่มคำสั่งซึ่งเป็นคำย่อของ substitute. ส่วนที่ต้องการสับเปลี่ยนและคำที่ต้องการแทนจะคล่อมด้วยเครื่องหมาย slash (/). `g` เป็นคำสั่งเพิ่มเติมใช้สำหรับสับเปลี่ยนคำที่ต้องการหลายครั้ง. โดยปรกติถ้าไม่ระบุตัวเลือกนี้ `sed` จะสับเปลี่ยนคำเฉพาะคำที่เจอครั้งแรกเท่านั้น.


`g` ย่อมาจากคำสั่ง global.

ตัวอย่างที่ 4.58: การใช้ sed สับเปลี่ยนคำ.

```
$  sed -ne 's/title/TITLE/p' simple.html↵
<TITLE>Simple HTML</title>
$  sed -ne 's/title/TITLE/gp' simple.html↵
<TITLE>Simple HTML</TITLE>
```

ในส่วนของ regular expression เราสามารถจับกลุ่มคำที่ต้องการแล้วอ้างอิงใช้ในภายหลังได้. ตัวอย่างต่อไปนี้เป็นการใช้การจับกลุ่มคำแล้วนำมาใช้ภายหลังในช่วงคำที่จะเปลี่ยน.

ตัวอย่างที่ 4.59: การจำกลุ่มคำใน regular expression.

```
$ sed -e 's/\(<\S+>\)/\U\1/g' simple.html↵
<HTML>
<HEAD>
<TITLE>Simple HTML</TITLE>
</HEAD>
<BODY>
This is a basic simple HTML file.
<!-- This line is a comment -->
</BODY>
</HTML>
```

คำที่ต้องการจับกลุ่มจะคล่อมด้วยเครื่องหมาย backslash และวงเล็บ, \(\). การอ้างอิงส่วนที่จับกลุ่มไว้ในภายหลังจะใช้ \N โดยที่ N เป็นตัวเลข 1 ถึง 9. เช่นถ้ามีการจับกลุ่มไม่หนึ่งกลุ่มและต้องการอ้างอิงภายหลังก็จะเป็น \1 เหมือนในตัวอย่าง.

สำหรับส่วนที่เป็น regular expression, \S แทนอักษรใด ๆที่ไม่ใช่ช่องว่างเช่นช่องไฟหรือแคร่. + เป็นการระบุว่ามีอักษร \S มากกว่าหนึ่งตัวขึ้นไป. ช่วงของคำที่จะเปลี่ยนมีการใช้คำสั่ง \U ให้เปลี่ยนคำที่ตามหลังมาให้เป็นตัวอักษรตัวใหญ่.

คำสั่งของ sed ก็คล้ายกับคำสั่งที่ใช้ในเชลล์คือมีอักขระที่บอกถึงการจบคำสั่งแต่ละคำสั่ง. อักขระที่เป็นตัวแบ่งคำสั่งได้แก่อักขระขึ้นบรรทัดใหม่, และอักขระ semicolon (;). ตัวอย่างต่อไปนี้จะแสดงการแบ่งคำสั่งด้วยอักขระทั้งสองแบบ.

ตัวอย่างที่ 4.60: ใช้คำสั่งของ sed หลายคำสั่งพร้อม ๆกัน.

```
$ sed -ne '1p;4p' simple.html↵
<html>
</head>
$ sed -ne '1p↵
> 4p' simple.html↵
<html>
</head>
```

วิธีอีกอย่างที่ใช้ได้คือการใช้ตัวเลือก -e แบ่งคำสั่งเช่น -e '1p' -e '4p' ก็จะให้ผลเหมือนกัน. ในการใช้งานจริงถ้ามีการสั่งคำสั่งเป็นชุดความจะเขียนเป็นสคริปต์เก็บไว้ในไฟล์แล้วใช้ตัวเลือก -f เรียกสคริปต์มาใช้แทนการสั่งคำสั่ง sed ทางบรรทัดคำสั่ง.

สมมติว่าเราต้องการสั่งคำสั่งเป็นชุดกับตำแหน่งที่ต้องการ, ให้ใช้การจับกลุ่มของคำสั่งด้วยเครื่องหมาย brace ({}).

ตัวอย่างที่ 4.61: การจับกลุ่มคำสั่งของ sed.

```
$ sed -e '/<body>/,/<\/body>/{↵
> s/HTML/html/;s/line //}' simple.html↵
<html>
<head>
<title>Simple html</title>
</head>
<body>
This is a basic simple html file.
<!-- This is a comment -->
```

```
</body>
</html>
```

จากตัวอย่างเป็นการใช้ sed ให้เปลี่ยนคำว่า HTML ให้เป็น html และลบคำว่า line ออกไปตั้งแต่บรรทัด <body> จนถึง <\body>.

ผู้อ่านจะสังเกตุเห็นว่าคำสั่ง sed จะแสดงผลลัพธ์ออกทาง stdout. เพราะฉะนั้นถ้าต้องการจะบันทึกผลลัพธ์นั้นลงไฟล์ก็ต้องใช้การรีไดเรก. สำหรับชื่อไฟล์ที่ต้องการเก็บผลลัพธ์นั้นต้องเป็นชื่อไฟล์ที่ไม่ใช่ไฟล์ที่กำลังประมวลผล, มิฉะนั้นข้อมูลจะหายไป. แต่การรีไดเรกผลลัพธ์ลงในไฟล์ใหม่นั้นก็ไม่สะดวกเช่นกันถ้าเราต้องการจะแก้ไฟล์นั้น. เพราะหลังจากที่รีไดเรกแล้วเราต้องทำการเปลี่ยนชื่อไฟล์ใหม่ให้เป็นไฟล์ที่ต้องการทีหลัง.

ข้อผิดพลาดที่ควรระวังของการใช้รีไดเรกให้ดูตัวอย่างจากหน้า 47

เพื่อแก้ปัญหาความไม่สะดวกนี้, เราสามารถใช้ตัวเลือก -i เพื่อให้ sed แก้ไขไฟล์ที่ต้องการเลย, ไม่ต้องแสดงผลออกทาง stdout.

ตัวอย่างที่ 4.62: ให้ sed แก้ไฟล์โดยบันทึกผลลัพธ์ในไฟล์นั้น.

```
$ sed -i.bak -e 's/\(<\S*>\)/\U\1/g' simple.html↵
$ ls↵
simple.html simple.html.bak
$ cat simple.html↵
<HTML>
<HEAD>
<TITLE>Simple HTML</TITLE>
</HEAD>
<BODY>
This is a basic simple HTML file.
<!-- This line is a comment -->
</BODY>
</HTML>
$ cat simple.html.bak↵
<html>
<head>
<title>Simple HTML</title>
</head>
<body>
This is a basic simple HTML file.
<!-- This line is a comment -->
</body>
</html>
```

จากตัวอย่าง, ตัวเลือก -i มีอาร์กิวเมนต์ .bak ซึ่งคำสั่ง sed จะสร้างไฟล์ชื่อ simple.html.bak ให้เป็นไฟล์สำรองที่มีเนื้อหาเดิมก่อนที่จะมีการเปลี่ยนแปลง. ส่วนเนื้อหาของไฟล์ simple.html เป็นเนื้อหาที่ sed ได้ทำการเปลี่ยนแปลงเรียบร้อยแล้ว. การใช้ sed กับตัวเลือก -i ทำให้ผู้ใช้ไม่ต้องรีไดเรกและยังสามารถเปลี่ยนไฟล์กลับเป็นอย่างเก่าโดยใช้ไฟล์สำรองที่สร้างไว้. ถ้าไม่มีการให้อาร์กิวเมนต์ส่วนขยายชื่อไฟล์ให้ -i, จะไม่มีการทำไฟล์สำรองให้.

4.5.2 awk

awk เป็นโปรแกรมคำสั่งสำหรับประมวลผลข้อมูลเท็กซ์และรายงานผล. คำสั่งสั่ง awk มีมานานพร้อมกับระบบปฏิบัติการยูนิกซ์สมัยแรก ๆ. โปรแกรมนี้แรกสุดสร้างโดย Alfred V. Aho, Peter J. Weinberger และ Brian Kernighan. คำสั่ง awk ที่ใช้กันในลินุกซ์เป็นโปรแกรมที่สร้างใหม่ให้เข้ากันกับ awk ดั้งเดิมโดย Free Software Foundation มีชื่อว่า gawk. ในระบบทั่วไปก็มักจะนิยมสร้าง awk ให้เป็นซอฟต์ลิงค์ของ gawk.

คำสั่ง awk เป็นภาษาคอมพิวเตอร์แบบหนึ่ง, มีหลักการการทำงานคล้าย sed คือรับข้อมูลเป็นบรรทัด ๆแต่มีความสามารถมากกว่า sed เช่นสร้างตัวแปรเก็บข้อมูลได้, มีฟังก์ชันคณิตศาตร์ช่วยประมวลผล, มีไวยกรณ์แยกเงื่อนไข ฯลฯ. การสั่งคำสั่ง awk สามารถเขียนไปคำสั่งส่งทางอาร์กิวเมนต์หรือจะเขียนเป็นไฟล์เก็บไว้แล้วรับคำสั่งจากไฟล์นั้นก็ได้.

```
awk [-f script] 'pattern {action}' [file_target]
```

การระบุ pattern ของคำสั่ง awk จะใช้ regular expression เหมือนกับคำสั่ง sed. action จะเป็นการกระทำที่เกิดขึ้นเมื่อเจอ pattern ที่ต้องการ. เราลองมาดูตัวอย่างการใช้ awk ร่วมกับคำสั่งอื่นดังนี้.

ตัวอย่างที่ 4.63: การใช้ awk เบื้องต้น.

```
$ du -s /usr/* | head | awk '/X11/ {print $0}'↵
164900  /usr/X11R6
```

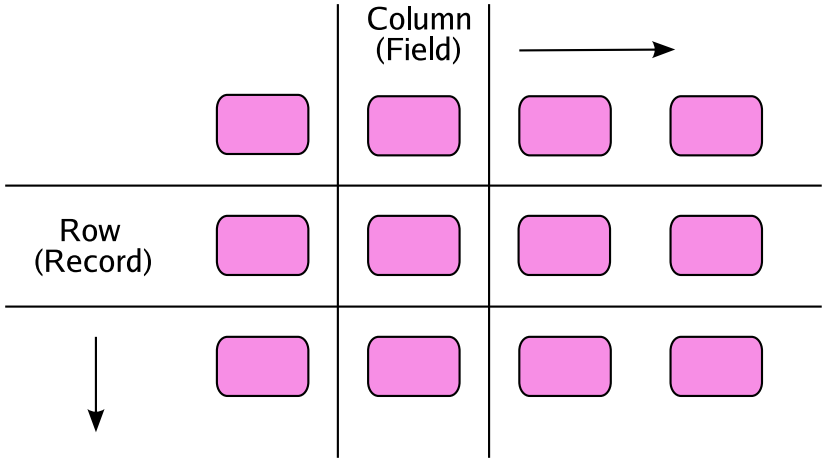
จากตัวอย่าง, awk รับข้อมูลจาก stdin ซึ่งเป็นผลลัพธ์ของคำสั่ง du ซึ่งแสดงการใช้พื้นที่ของไดเรกทอรีต่าง ๆใต้ /usr และส่งต่อให้คำสั่ง head เพื่อลดจำนวนผลลัพธ์ให้เหลือ 10 บรรทัด. จากนั้นใช้ awk แสดงบรรทัดที่มีคำว่า X11.

head จะแสดงข้อมูล 10 บรรทัดแรกโดยปริยายถ้าไม่ระบุจำนวนบรรทัดที่ต้องการ.

แถวและคอลัมน์

awk จะรับข้อมูลบรรทัดต่อบรรทัดเหมือนกับคำสั่ง sed. awk จะเรียกข้อมูลที่มีหน่วยเป็นบรรทัดว่าแถว (row) หรือ *record*. นอกจากนั้นยังแยกข้อมูลในบรรทัดต่อไปอีกเป็นคอลัมน์ (column) หรือ *field*. เราสามารถมองไฟล์ให้เป็นตารางข้อมูลแถวและคอลัมน์, และในคำสั่ง awk จะมีตัวแปรแทนสำหรับข้อมูลในแถวและคอลัมน์ที่ต้องการ. ข้อมูลคอลัมน์มีการแบ่งเป็นช่วง ๆแยกด้วยเครื่องหมายที่กำหนดเช่น ช่องว่าง, colon เป็นต้น. จากตัวอย่างที่ 4.63, อักษรที่แบ่งข้อมูลเป็น field ได้แก่ช่องว่าง. ตัวแปรที่ใช้แทนข้อมลแถวได้แก่ $0 ซึ่งจะหมายถึงบรรทัดทั้งบรรทัด. ส่วนตัวแปรที่ใช้แทนข้อมูลคอลัมน์แต่ละคอลัมน์ได้แก่ $1, $2 ไปเรื่อย ๆ.

คำสั่ง awk จะถือว่าอักขระ LF (line feed) เป็น*ตัวแบ่งแถว (record separator)* และถือว่าช่องว่างเป็น*ตัวแบ่งคอลัมน์ (field separator)* โดยปริยาย. ดังนั้นจากตัวอย่างที่ ?? ถ้าต้องการเปลี่ยนลำดับผลลัพธ์ของ du ให้ชื่อไดเรกทอรีนำหน้าทำได้ดังนี้.



รูปที่ 4.7: ความสัมพันธ์ระหว่างแถวและคอลัมน์ในไฟล์.

ตัวอย่างที่ 4.64: การใช้ข้อมูลคอลัมน์ใน awk.

```
$ du -s /usr/* | head -n 5 | awk '{print "line" NR, $2, $1}'↵
line1 /usr/X11R6 164900
line2 /usr/bin 156324
line3 /usr/doc 0
line4 /usr/i686-pc-linux-gnu 2532
line5 /usr/include 54984
```

จากตัวอย่างไม่มีการระบุรูปแบบ (pattern) ต้องการ, มีเฉพาะคำสั่ง (action) ซึ่งคำสั่งนี้จะอยู่ในวงเล็บปีกกาและกระทำทุกครั้งเมื่ออ่านข้อมูลบรรทัดต่อบรรทัด. `print` เป็นคำสั่ง (action) ที่ใช้บ่อย, สามารถใช้แสดงคำหรือตัวแปร. คำที่ต้องการแสดงจะต้องเขียนอยู่ใน single quote หรือ double quote. `NR` คือตัวแปรประกอบอยู่ภายใน (build-in variable) ใช้บอกจำนวนบรรทัดที่ผ่านการประมวลผลแล้ว. ส่วนเครื่องหมาย comma เป็นตัวแบ่งสิ่งที่ต้องการพิมพ์ด้วยอักขระที่กำหนดไว้ในตัวแปรพิเศษ `OFS` ซึ่งจะมีค่าปริยายเป็นช่องไฟ. ดังนั้นผลของคำสั่ง `print` จะแบ่งข้อมูลแต่ละคอลัมน์ด้วยช่องไฟ. ให้สังเกตว่าเวลาสั่งคำสั่ง `print` ที่หลังคำว่า “line” คือเครื่องหมายแคร่ไม่ใช่ช่องไฟเพราะไม่ได้เขียนแยกด้วย comma.

4.5.3 บล็อกพิเศษ

ในไวยกรณ์ของ `awk` จะมีบล็อก (block) พิเศษคล้ายกับบล็อกที่คล่อมด้วยเครื่องหมายวงเล็บปีกกาเหมือนบล็อกคำสั่ง. บล็อกพิเศษดังกล่าวนี้ได้แก่ `BEGIN{...}` และ `END{...}`. บล็อก `BEGIN` มักใช้สำหรับตั้งค่าตัวแปรเริ่มต้นหรือกระทำการก่อนที่จะประมวลผลข้อมูลในแต่ละบรรทัด. บล็อก `END` เป็นส่วนที่กระทำการหลังจากการประมวลผลของทุกบรรทัดเสร็จเรียบร้อย.

ตารางที่ 4.12: ตัวแปรประกอบอยู่ภายในคำสั่ง awk.

ตัวแปร	คำอธิบาย
$0	ข้อมูลที่อยู่ในบรรทัดที่ประมวลผลอยู่.
$1, $2, ...	ข้อมูลที่อยู่ในแต่ละคอลัมน์.
NR	จำนวนบรรทัดที่ประมวลผลไปแล้ว.
NF	จำนวนคอลัมน์ที่มีอยู่ในแถว.
FS	ตัวแบ่งคอลัมน์สำหรับการอ่านข้อมูลเข้า. ค่าโดยปริยายคือช่องว่าง.
OFS	ตัวแบ่งคอลัมน์สำหรับการแสดงผล. ค่าโดยปริยายคือช่องไฟ.
RS	ตัวแบ่งแถวสำหรับการอ่านข้อมูลเข้า. ค่าโดยปริยายคืออักขระ LF.
ORS	ตัวแบ่งแถว. ค่าโดยปริยายคืออักขระ LF.

ตัวอย่างที่ 4.65: การใช้บล็อกพิเศษใน awk.

```
$ awk 'BEGIN{FS=":"; ORS="\n-----\n"}↵
> /root/ {print "username:", $1 "\nuid:", $3}' /etc/passwd↵
username: root
uid: 0
-----
username: operator
uid: 11
-----
username: cvs
uid: 102
-----
```

จากตัวอย่างเป็นการประมวลผลไฟล์ /etc/passwd ซึ่งเป็นไฟล์ที่เก็บชื่อผู้ใช้, uid, gid ฯลฯ. ข้อมูลเหล่านี้จะมีเครื่องหมาย colon แบ่งเป็นคอลัมน์. ในตัวอย่างจะเป็นการสกัดเอาข้อมูลจากคอลัมน์ที่ 1 ซึ่งได้แก่ชื่อผู้ใช้และข้อมูลจากคอลัมน์ที่ 2 ซึ่งได้แก่ uid มาจัดเรียงแสดงผลใหม่. ในบล็อก BEGIN จะเป็นการตั้งค่าตัวแปรพิเศษ FS ซึ่งบอกให้ awk รู้ว่าตัวแบ่งคอลัมน์คือเครื่องหมาย colon, และตั้งค่า ORS ให้แสดง ----- หลังจบการแสดงข้อมูลแต่ละแถว. หลังจากนั้นจะเป็นการกระทำการกับบรรทัดที่มีคำว่า root โดยจัดรูปแบบใหม่ตามผลที่แสดงในตัวอย่าง.

4.5.4 การกระทำในคำสั่ง awk

การกระทำ (action) คือการประมวลผลที่เกิดขึ้นเมื่อเจอ pattern ที่ต้องการ. จากตัวอย่างที่ผ่านมา print เป็นการกระทำอย่างหนึ่งที่ใช้แสดงข้อมูลออกทาง stdout. นอกจากนี้แล้วโปรแกรม awk ยังรับรู้คำสั่งอื่น ๆอีกได้แก่.

operator ▶ *ตัวดำเนิการ* หรือ *ตัวปฏิบัติการ*. คือเครื่องหมายที่มีความหมายพิเศษใช้สำหรับการประมวลผล. เช่น + เป็นตัวปฏิบัติการทางคณิตศาสตร์สำหรับรวมค่าของจำนวนสองจำนวน. && เป็นตัวปฏิบัติการทางตรรกะ AND.

- *ตัวปฏิบัติการ (operator)* เช่นตัวปฏิบัติการเลขคณิต (+, -, *, /, %), ตัวปฏิบัติการตรรกะ (&&, ||) ฯลฯ.
- ฟังก์ชันคณิตศาสตร์เช่น cos, exp, log ฯลฯ.

- ฟังก์ชันประมวลผลสายอักษรเช่น gsub, index, length, match ฯลฯ.
- ฟังก์ชันประกอบภายใน awk อื่น ๆเช่น print, printf, system ฯลฯ.

ตัวอย่างที่ 4.66: การใช้ awk หาผลรวมของพื้นที่ไดเรกทอรีที่ต้องการ.

```
$ du -s /usr/X11R6 /usr/local |↵
> awk '{sum += $1} END {print "Total: " int(sum/1024) " KB"}'↵
Total: 246 KB
```

ตัวอย่างข้างบนเป็นการรวมค่าจำนวนไบต์จากคำสั่ง du และใช้ awk เลือกเอาตัวเลขจากคอลัมน์แรกมารวมกันแล้วจะแสดงผลให้ง่ายขึ้น.

ในที่ไม่สามารถอธิบายการใช้งาน awk ได้โดยละเอียดเนื่องจากความซับซ้อนของโปรแกรมภาษา awk. สำหรับผู้ที่ต้องการเรียนรู้การใช้โปรแกรมภาษา awk อย่างละเอียดอาจจะเริ่มจากการอ่าน manpage ของคำสั่ง awk เพื่อดูรายละเอียดเกี่ยวกับตัวเลือก, ตัวแปร, และการกระทำต่าง ๆ. หลังจากนั้นอาจจะหาข้อมูลทางอินเทอร์เน็ตหรือหาหนังสือที่เกี่ยวกับคำสั่ง awk โดยตรงต่อไป [32].

4.6 คำสั่งอื่น ๆ

4.6.1 การบีบอัดไฟล์

โปรแกรมคำสั่งสำหรับ*บีบอัด (compress)* ไฟล์ในลินุกซ์มีหลายประเภทเช่น compress, zip, gzip และ bzip2 เป็นต้น. โดยปรกติแล้วถ้ามีโปรแกรมบีบอัดแล้วก็ต้องมีโปรแกรมสำหรับ*ขยาย (decompress)* ไฟล์ด้วยเป็นคู่กันไปเช่น uncompress, unzip, gunzip และ bunzip2. แต่ในความเป็นจริงแล้วโปรแกรมบีบอัดมักจะมีความสามารถในการขยายไฟล์ได้ในตัวเอง, ไม่จำเป็นต้องมีโปรแกรมเฉพาะสำหรับการขยายไฟล์. ดังนั้นจะเห็นว่าโปรแกรมขยายไฟล์บางอย่างเช่น gunzip จะเป็น soft link ไปหาโปรแกรม gzip, ไม่ได้เป็นโปรแกรมแยกต่างหาก.

คำสั่งบีบอัดเหล่านี้มีส่วนที่เหมือนกันหลายอย่างได้แก่

- รับข้อมูลได้จาก stdin หรือไฟล์. ถ้าไม่มีชื่อไฟล์เป็นอาร์กิวเมนต์ของคำสั่งก็จะรับข้อมูลจาก stdin.
- ส่งข้อมูลที่บีบอัดแล้วให้ stout หรือบันทึกลงไฟล์.
- มีตัวเลือกที่เหมือน ๆกัน. กล่าวคือถ้าจำตัวเลือกของโปรแกรมบีบอัดอย่างหนึ่งได้, ก็เอาไปใช้กับโปรแกรมบีบอัดอื่น ๆได้ด้วย.

compress

โปรแกรมคำสั่ง compress เป็นโปรแกรมบีบอัดเก่าแก่ที่มาจากยูนิกซ์ซึ่งในปัจจุบันไม่ค่อยใช้กันแล้ว. ถ้าไม่มีอาร์กิวเมนต์เป็นชื่อไฟล์ก็จะรับข้อมูลจาก stdin.

ตารางที่ 4.13: ตัวเลือกสำหรับโปรแกรมบีบอัดข้อมูลทั่วไป.

ตัวเลือก	ความหมาย
-c	ส่งข้อมูลที่บีบอัดแล้วออกทาง stdout แทนที่จะบันทึกลงไฟล์.
-d	ขยายไฟล์. ถ้าตัวเลือกนี้ใช้ร่วมกับตัวเลือก -c ก็จะแสดงข้อมูลที่ขยายแล้วออกทาง stdout.
-t	ทดสอบไฟล์ที่บีบอัดแล้วว่าถูกต้องหรือไม่.
-v	ย่อมาจาก verbose. ใช้แสดงสถานะการทำงาน. โดยปรกติจะบอกสัดส่วนของข้อมูลที่บีบอัดแล้วเทียบกับข้อมูลเดิม.

ตัวอย่างที่ 4.67: การใช้ compress บีบอัดข้อมูล.

```
$ ls -l big_foo↵
-rw-r--r--    1 poonlap  users     1000000 Aug 11 22:39 big_foo
$ compress big_foo↵
big_foo:  -- replaced with big_foo.Z Compression: 99.81%
```

ในตัวอย่างมีการใช้ตัวเลือก -v เพื่อให้แสดงสถานะการทำงานของคำสั่ง. ในกรณีนี้จะแสดงสัดส่วนที่โปรแกรมสามารถบีบอัดได้. ถ้าเป็นไฟล์ที่ข้อมูลสามารถบีบอัดได้ก็จะสร้างไฟล์ชื่อเดียวกันโดยมีส่วนขยายชื่อไฟล์เป็น .Z.

ถ้าผู้อ่านเคยดาว์นโหลดรหัสโปรแกรมเก่า ๆที่มีส่วนขยายชื่อไฟล์เป็น .Z ก็จะรู้ว่าใช้ compress ในการบีบอัด.

ตัวเลือก -v เป็นตัวเลือกที่สามารถใช้ได้กับโปรแกรมบีบอัดอื่นได้ด้วยและมีประโยชน์ช่วยให้ผู้ใช้รู้ว่าข้อมูลนั้นบีบอัดไปได้เท่าไร. ถ้าต้องการขยายไฟล์ที่บีบอัดออกให้ใช้ตัวเลือก -d.

ตัวอย่างที่ 4.68: ขยายไฟล์ที่ถูกบีบอัดออกด้วย compress.

```
$ compress -dv big_foo.Z↵
big_foo.Z:  -- replaced with big_foo
```

จะเห็นว่าคำสั่ง compress จะขยายไฟล์แล้วเก็บข้อมูลในชื่อไฟล์เดียวกันแต่เอาส่วนขยายชื่อไฟล์ .Z ออก. ส่วนไฟล์ที่ถูกบีบอัดก็จะหายไป. ถ้าเราต้องการขยายไฟล์แล้วเก็บเป็นไฟล์ชื่ออื่นและไม่ต้องการให้ไฟล์ที่ถูกบีบอัดแล้วหายไป, ให้ใช้กลวิธีรีไดเรก.

ตัวอย่างที่ 4.69: ขยายไฟล์ที่ถูกบีบอัดและเก็บในไฟล์ด้วยการรีไดเรก.

```
$ compress -dc big_foo.Z > big_foo↵
$ ls -l big_foo.Z big_foo↵n
-rw-r--r--    1 poonlap  users     1000000 Aug 11 23:48 big_foo
-rw-r--r--    1 poonlap  users        1820 Aug 11 22:39 big_foo.Z
```

gzip

โปรแกรมคำสั่ง gzip[33] ย่อมากจา GNU zip ซึ่งเป็นโปรแกรมบีบอัดที่พัฒนาในโครงการ GNU, Free Software Foundation เพื่อมาแทนที่โปรแกรม compress. คำสั่ง gzip เป็นที่นิยมกว้างขวางสำหรับการบีบอัดข้อมูลพอ ๆกับคำสั่ง bzip2. ไฟล์ที่บีบอัด

ด้วยคำสั่ง gzip มักจะมีส่วนขยายชื่อไฟล์เป็น .gz.

การใช้งานของคำสั่ง gzip โดยพื้นฐานจะเหมือนกับคำสั่ง compress ที่ได้แนะนำไปแล้ว. เมื่อให้ชื่อไฟล์เป็นอาร์กิวเมนต์ของคำสั่งก็จะบีบอัดไฟล์นั้นและเพิ่มส่วนขยายชื่อไฟล์ .gz. การขยายไฟล์ที่ถูกบีบอัดก็ทำได้โดยใช้ตัวเลือก -d หรือคำสั่ง gunzip. ถ้าต้องการรับข้อมูลจาก stdin ให้ใช้ชื่อไฟล์เป็นเครื่องหมาย -.

gzip อ้างอิงหน้า 388

ไฟล์หลายไฟล์ในระบบปฏิบัติการลินุกซ์มันจะบีบอัดด้วย gzip เพื่อประหยัดพื้นที่ฮาร์ดดิสก์เช่น ไฟล์ของคู่มือใช้งานที่อยู่ใต้ไดเรกทอรี /usr/share/man, ไฟล์ข้อมูลของชุดอักขระต่าง ๆที่อยู่ใต้ไดเรกทอรี /usr/share/i18n/charmaps เป็นต้น. ไฟล์เหล่านี้มักจะเป็นไฟล์เท็กซ์. ถ้าต้องการดูเนื้อหาของไฟล์เหล่านี้เราสามารถใช้คำสั่ง zcat ดูเนื้อหาที่ขยายได้ทันที.

ตัวอย่างที่ 4.70: การใช้ zcat ดูข้อมูลจากไฟล์ที่บีบอัดไว้.

```
$ zcat /usr/share/i18n/charmaps/UTF-8.gz↵
<code_set_name> UTF-8
<comment_char> %
<escape_char> /
<mb_cur_min> 1
<mb_cur_max> 6

% alias ISO-10646/UTF-8
CHARMAP
<U0000>     /x00         NULL
<U0001>     /x01         START OF HEADING
...
```

เนื่องจาก gzip มีจุดประสงค์ที่สร้างขึ้นมาเพื่อแทน compress, ดังนั้นคำสั่ง gzip ยังสามารถขยายไฟล์ที่บีบอัดด้วยคำสั่ง compress ได้ด้วย.

ขยายไฟล์ที่บีบอัดด้วย zip ก็ได้.

ตัวเลือกที่น่าสนใจของคำสั่ง gzip ได้แก่ --fast ซึ่งเขียนแบบย่อเป็น -1, และตัวเลือก --best ซึ่งเขียนแบบย่อเป็น -9. ตัวเลือก -1 หมายถึงให้ gzip บีบอัดไฟล์ให้เร็วที่สุด, ความบีบอัดของข้อมูลต่ำ. ส่วน -9 หมายถึงให้ gzip บีบอัดไฟล์ให้มีความบีบอัดสูงเท่าที่จะเป็นไปได้, ทำให้ใช้การประมวลผลช้า. การทำงานโดยปริยายจะใช้ระดับความเร็วเป็น -6.

ตัวอย่างที่ 4.71: เปอร์เซนต์ความบีบอัดตามตัวเลือกของ gzip.

```
$ i=1↵
> while [ $i -lt 10 ]↵                          ← กระทำจนกว่าค่า i จะน้อยกว่า 10
> do↵
> echo -n $i: compression ratio↵
> zcat /usr/share/i18n/charmaps/UTF-8.gz | gzip -cv$i - > /dev/null↵
> i=$(($i+1))↵                                                  ← เพิ่มค่า i
> done↵
1: compression ratio 80.4%
2: compression ratio 80.5%
3: compression ratio 80.8%
4: compression ratio 81.6%
5: compression ratio 82.8%
6: compression ratio 82.9%
7: compression ratio 83.2%
```

```
8: compression ratio 83.2%
9: compression ratio 83.2%                    ← บีบอัดได้มากที่สุด
```

bzip2

bzip2 อ้างอิงหน้า 383

bzip2 เป็นโปรแกรมบีบอัดที่พัฒนาหลังจาก gzip, เผยแพร่ได้อย่างอิสระ, ไม่มีปัญหาละเมิดสิทธิบัตรที่เกี่ยวกับวิธีการบีบอัด, และสามารถบีบอัดได้แน่นกว่า gzip. โดยปรกติจะพยายามบีบอัดให้ได้ไฟล์มีขนาดประมาณ 10 ถึง 15 เปอร์เซนต์ของไฟล์เดิม [34]. การใช้และตัวเลือกโดยทั่วไปก็เหมือนกับคำสั่ง compress และ gzip ที่แนะนำไปแล้ว.

zip และ unzip

zip อ้างอิงหน้า 394

โปรแกรม zip[35] และ unzip ที่ใช้ในลินุกซ์มาจากลินุกซ์เป็นการโปรแกรมที่สร้างสำหรับบีบอัดขยายไฟล์ให้เข้ากันได้กับโปรแกรม zip/unzip โดย PKWARE ที่กันในระบบปฏิบัติการ DOS หรือวินโดวส์. เนื่องจากโปรแกรม zip และ unzip ในลินุกซ์เป็นโปรแกรมที่พัฒนาเองให้เข้ากันได้กันโปรแกรมที่มีอยู่แล้วบนวินโดวส์, ดังนั้นความสามารถบางอย่างที่โปรแกรม zip (pkzip) บนวินโดวส์ทำได้, โปรแกรม zip และ unzip ในลินุกซ์อาจจะทำไม่ได้. นอกจากคำสั่ง zip และ unzip แล้ว, ยังมีโปรแกรมที่เกี่ยวกับ zip ที่อยู่ในชุดเดียวกันด้วยในตารางที่ 4.14.

PKWARE เป็นบริษัทที่เชี่ยวชาญและผลิตซอฟต์แวร์ที่เกี่ยวกับการบีบอัดหรือขยายข้อมูล.

ตารางที่ 4.14: ชุดโปรแกรมที่เกี่ยวข้องกับ zip.

โปรแกรม	คำอธิบาย
zip	สำหรับบีบอัดข้อมูล.
unzip	สำหรับขยายข้อมูลที่บีบอัดด้วย zip.
zipnote	บันทึกหรือลบหมายเหตุในไฟล์ zip.
zipsplit	แบ่งไฟล์ zip เป็นไฟล์ย่อย ๆ.
zipcloak	เข้ารหัสหรือถอดรหัสสำหรับไฟล์ zip.

โปรแกรม zip ไปโปรแกรมที่สามารถบีบอัดไฟล์หลาย ๆไฟล์แล้วเก็บเข้าไว้ในไฟล์เดียว. เมื่อขยายไฟล์ที่เก็บไฟล์ไว้หลาย ๆไฟล์ออกมาก็จะคงสภาพโครงสร้างไดเรกทอรีไว้ให้ด้วย. จุดนี้แตกต่างจากโปรแกรมบีบอัดพวก compress ซึ่งไม่สามารถบีบอัดรวมไฟล์หลายไฟล์เข้าในไฟล์เดียว.

archive ▶
การเก็บไฟล์, เอกสารสำคัญไว้. ถ้าเป็นคำนามจะหมายถึงเอกสารที่เก็บไว้เช่นเอกสารสำรองหรือเอกสารสำคัญ.

การใช้ zip จะมีไวยกรณ์ที่ต่างจะ compress ที่ต้องตั้งชื่อไฟล์ที่จะเป็น archive ให้เป็นอาร์กิวเมนต์แล้วอาร์กิวเมนต์ถัดไปเป็นรายการไฟล์ที่ต้องการเก็บไว้ในไฟล์ zip (ไฟล์ archive) ที่ตั้งชื่อไว้.

ตัวอย่างที่ 4.72: การใช้ zip.

```
$ zip -j foo /etc/passwd /etc/X11/XF86Config↵
```

```
adding: passwd (deflated 65%)
adding: XF86Config (deflated 69%)
```

จากตัวอย่างมีการใช้ตัวเลือก `-j` เพื่อเก็บเฉพาะชื่อไฟล์อย่างเดียวโดยไม่เก็บส่วนที่เป็นชื่อ path เช่นชื่อไดเรกทอรี. หลังจากที่คำสั่ง `zip` ทำงานเรียบร้อยแล้วจะสร้างไฟล์ `foo.zip` ไว้ในไดเรกทอรีที่ทำงานอยู่. ถ้าขยายไฟล์ที่ถูกบีบอัดนี้แล้วก็จะได้ไฟล์ `passwd` และไฟล์ `XF86Config` อยู่ในไดเรกทอรีที่ทำการขยายไฟล์นั้น.

ถ้าต้องการจะบีบอัดและเก็บไฟล์เข้าในไฟล์ zip เพิ่มเติมก็ใช้คำสั่งแบบเดียวกัน. ตัวอย่างต่อไปนี้จะไม่ใช้ตัวเลือก `-j` ซึ่งจะเก็บเอาชื่อไฟล์แบบ full path ไว้ในไฟล์ zip.

ตัวอย่างที่ 4.73: การเพิ่มไฟล์เข้าในไฟล์ zip.

```
$  zip foo /etc/group↵
  adding: etc/group (deflated 47%)
```

ถ้าต้องการดูไฟล์ที่บีบอัดอยู่ในไฟล์ zip, ให้ใช้คำสั่งสั่ง `unzip` กับตัวเลือก `-t` เพื่อทดสอบ (test) ไฟล์ zip และแสดงรายการไฟล์ที่อยู่ในไฟล์ zip นั้น.

unzip อ้างอิงหน้า ??

หรือใช้ตัวเลือก `-l` เพื่อที่จะแสดงรายการไฟล์อย่างเดียว, ไม่ตรวจสอบไฟล์.

ตัวอย่างที่ 4.74:

```
$ unzip -t foo.zip↵                          ← จะไม่เขียนส่วนขยายชื่อไฟล์ .zip ก็ได้
Archive:  foo.zip
    testing: passwd                   OK
    testing: XF86Config               OK
    testing: etc/group                OK
No errors detected in compressed data of foo.zip.
```

จะเห็นได้ว่าถ้าไม่ใช้ตัวเลือก `-j` ตอนที่บีบอัดไฟล์, คำสั่ง `zip` จะเก็บชื่อไฟล์แบบตามที่ระบุโดยจะลบเครื่องหมาย slash (/) ถ้าชื่อไฟล์นั้นขึ้นต้นด้วยไดเรกทอรีรูท. สำหรับการขยายไฟล์ zip โดยปรกติจะใช้คำสั่ง `unzip` โดยที่ไม่มีตัวเลือก.

คำสั่ง `zip` และ `unzip` ยังมีตัวเลือกอื่น ๆมากมายซึ่งผู้อ่านสามารถอ่านรายละเอียดได้จากคู่มือการใช้งาน.

4.6.2 การทำสำรองไฟล์ทั้งไดเรกทอรี

คำสั่งบีบขอัดข้อมูลเช่น `compress`, `gzip` และ `bzip2` ไม่สามารถรวมไฟล์เป็นไดเรกทอรีได้. การที่จะบีบอัดไฟล์ทั้งไดเรกทอรีแล้วกระจายออกมาให้เป็นโครงสร้างไดเรกทอรีและไฟล์เหมือนเดิมต้องใช้โปรแกรมอื่นช่วย. และโปรแกรมที่นิยมใช้ในการแบบนี้ได้แก่ `tar`.

คำสั่ง `tar` ย่อมาจากคำว่า tape archive ซึ่งเดิมทีใช้สำหรับ*สำรอง (backup)* ไฟล์บันทึกลงในเทป. แต่คำสั่ง `tar` ยังสามารถรวมไฟล์ต่าง ๆเก็บไว้เป็นไฟล์เดียวก็ได้. หลังจากนั้นใช้ `gzip` หรือ `bzip2` บีบอัดต่อเพื่อให้ขนาดเล็กลง.

tar อ้างอิงหน้า 392

คำสั่ง `tar` เป็นคำสั่งสำหรับสร้างไฟล์ archive (`.tar`) และกระจายไฟล์ต่าง ๆที่อยู่ใน archive โดยจะแยกหน้าที่ตามตัวเลือกที่ใช้.

ตารางที่ 4.15: ตัวเลือกที่ใช้บ่อยของคำสั่ง tar.

ตัวเลือก	ความหมาย
-c	สร้างไฟล์ archive.
-x	กระจายไฟล์ archive.
-f *filename.tar*	ระบุชื่อไฟล์ archive.
-r	เพิ่มไฟล์เข้าไปในไฟล์ archive ที่มีอยู่แล้ว.
-t	ทดสอบและแสดงรายการไฟล์ที่อยู่ในไฟล์ archive.
-v	แสดงสถานะการทำงาน.
-z	บีบอัดหรือคลายไฟล์ archive ด้วย gzip.
-j	บีบอัดหรือคลายไฟล์ archive ด้วย bzip2.
-p	รักษาสิทธิ์การใช้ไฟล์ที่ทำการ archive.
-O	กระจายไฟล์ที่อยู่ใน archive ออกทาง stdout.

ต่อไปนี้เป็นตัวอย่างการสร้างไฟล์ archive ของไดเรกทอรีชื่อ bar และไฟล์ต่าง ๆที่อยู่ในไดเรกทอรี.

ตัวอย่างที่ 4.75: การสร้างไฟล์สำรองข้อมูลด้วย tar.

```
$ tar -czvf bar.tar.gz bar↵                    ← จะระบุตัวเลือกเป็น czvf ก็ได้
bar/
bar/foo1
bar/foo2
bar/foo3/
bar/foo3/foo4
```

ผู้ใช้ต้องเขียนส่วนขยายชื่อไฟล์ .tar.gz เองซึ่งจะไม่เหมือนคำสั่ง zip ที่จะเติมส่วนขยายชื่อไฟล์ให้. ถ้าต้องการส่งข้อมูลออกทาง stdout, ให้เปลี่ยนชื่อไฟล์เป็น -. จากตัวอย่างมีการบีบอัดด้วย gzip จึงเขียนส่วนขยายชื่อไฟล์เป็น .tar.gz. ถ้าใช้ตัวเลือก -j ก็อาจให้ใช้ส่วนขยายชื่อไฟล์เป็น bz2 เพื่อความชัดเจน.

บ้างก็เขียน .tar.gz เป็น .tgz แล้วแต่บุคคล.

เมื่อสร้างไฟล์ archive เรียบร้อยแล้วสามารถตรวจสอบไฟล์ที่สร้างได้ด้วยตัวเลือก -t.

ตัวอย่างที่ 4.76: ตรวจสอบไฟล์ archive.

```
$ tar tzvf bar.tar.gz↵              ← ต่อไปนี้จะไม่พิมพ์เครื่องหมาย - เวลาใช้ตัวเลือก
drwxr-xr-x poonlap/users      0 2004-08-15 19:47:53 bar/
-rw-r--r-- poonlap/users      0 2004-08-15 19:47:49 bar/foo1
-rw-r--r-- poonlap/users      0 2004-08-15 19:47:49 bar/foo2
drwxr-xr-x poonlap/users      0 2004-08-15 19:48:00 bar/foo3/
-rw-r--r-- poonlap/users      0 2004-08-15 19:48:00 bar/foo3/foo4
```

การใช้ตัวเลือก -t นอกจากจะเป็นการทดสอบว่าไฟล์ archive ที่สร้างขึ้นมีข้อผิดพลาดหรือไม่แล้วยังสามารถดูรายการไฟล์ที่อยู่ใน archive ได้ด้วย. ไฟล์ต้นฉบับหรือโปรแกรม

บางอย่างเผยแพร่ในรูปของไฟล์ archive, ดังนั้นก่อนที่จะกระจายไฟล์ควรจะตรวจสอบและดูรายการไฟล์ก่อนว่าข้างในว่ามีการวางโครงสร้างไฟล์อย่างไร, มีความผิดพลาดหรือไม่. เช่นถ้าเรารู้ว่าในไฟล์ archive ไม่มีไดเรกทอรีย่อย, เวลากระจายไฟล์ออกมาก็จะสร้างไฟล์ที่กระจายมาอยู่ในไดเรกทอรีที่ทำงานอยู่ซึ่งในบางครั้งไม่เป็นที่ต้องการ. ในกรณีนี้เราก็สามารถสร้างไดเรกทอรีแล้วไปกระจายไฟล์ archive ไว้ในไดเรกทอรีที่เตรียมไว้.

สำหรับการกระจายไฟล์ archive ให้เปลี่ยนตัวเลือกจาก `-c` ให้เป็น `x` ซึ่งจะหมายถึงการกระจายเอาไฟล์ทุกไฟล์ออกมาจากไฟล์ archive. ในกรณีที่ต้องการเลือกเฉพาะไฟล์ที่ต้องการออกมาจากไฟล์ archive เท่านั้นให้ระบุชื่อไฟล์ที่ต้องการเป็นอาร์กิวเมนต์ต่อท้ายชื่อไฟล์ archive. การสำรวจชื่อไฟล์ที่อยู่ใน archive ให้ใช้ตัวเลือก `-t` ที่แนะนำไปแล้ว.

ตัวอย่างเช่นถ้าต้องการเอาไฟล์ `bar/foo1` และไฟล์ `bar/foo3/foo4` ออกมาจากไฟล์ `bar.tar.gz` สามารถทำได้ตามตัวอย่างต่อไปนี้.

ตัวอย่างที่ 4.77: กระจายไฟล์ที่เลือกออกจาก archive.

```
$ tar xzvf bar.tar.gz bar/foo1  bar/foo3/foo4↵
bar/foo1
bar/foo3/foo4
```

4.6.3 แปลงไบนารีให้เป็นเท็กซ์

ในสมัยที่อีเมลเริ่มใช้ใหม่ ๆ, การส่งไฟล์ไบนารีเช่นรูปภาพไม่สามารถทำได้ง่าย ๆเหมือนปัจจุบันเพราะตอนนั้นยังไม่มีมาตรฐานแน่นอนรองรับ, และข้อมูลที่ส่งผ่านเมลนั้นจะเป็นอักขระ ASCII คือข้อมูลแบบ 7 บิตเท่านั้น [36]. วิธีที่นิยมใช้กันในตอนนั้นคือแปลงข้อมูลไบนารีให้เป็นอักขระ ASCII ทั้งหมด, แล้วส่งข้อมูลนั้นทางเมล. การแปลงข้อมูลไบนารีเป็นข้อมูลเท็กซ์ ASCII นี้จะใช้คำสั่ง `uuendcode`. ในทางกลับกัน, การแปลงข้อมูลเท็กซ์ ASCII กลับให้เป็นข้อมูลไบนารีจะใช้คำสั่ง `uudecode`.

uuencode อ้างอิงหน้า 393

ตัวอย่างที่ 4.78: แปลงข้อมูลไบนารีให้เป็นข้อมูลเท็กซ์ ASCII.

```
$ uuencode -m image.png output_image.png > image.png.txt↵
$ cat image.png.txt↵
begin-base64 644 output_image.png
iVBORw0KGgoAAAANSUhEUgAAAAwAAAAMCAYAAABWdVznAAAABmJLR0QA/wD/
AP+gvaeTAAAA/0lEQVR42p2RvUrDUBSAv0Q3dRXB3AQBXFzU9AtTnOKR9HB
wc21PkoH8QHM6lKKQiAV+wBC+iem5p5zEgeT0Ejj4IXDhcP3cf7gP293bzsA
sqW4rmMb+Z91724ACMMRr8M3fP+pAu4f7DDoB53mcnI8fWd9o83m1iHe2WlF
CMMRg37Qqwgvz0Me7v0/2y9bKhKXV+d43slK+PioQ1nh0e/hxGGqzD/mqCgi
iqggiTCZzgBoAmuFFMefqP6AKoqoYmI4ERKXlEK7EBZxjKjlsGGmqBgigjgp
hRZAFEWMJzMOF9QU1RRTxdKURfxVztICLn4dri5uGys2VrfNDMi+AQ4ck5cN
bJUqAAAAAElFTkSuQmCC
====
```

จากตัวอย่าง, ชื่อไฟล์ที่ต้องการเข้ารหัสได้แก่ `image.png` จะเป็นอาร์กิวเมนต์ของคำสั่ง. ส่วน `output_image.png` จะเป็นชื่อไฟล์ที่จะกำหนดในผลลัพธ์ของการเข้ารหัสซึ่งจะใช้ชื่ออะไรก็ได้หรือชื่อเหมือนกับไฟล์ไบนารีที่ต้องการเข้ารหัสก็ได้. ชื่อไฟล์นี้จะใช้เวลาถอดรหัสด้วยคำสั่ง `uudecode`. คำสั่ง `uudecode` จะสร้างไฟล์ตามที่ระบุในข้อ

มูลซึ่งในที่นี้คือไฟล์ output_image.png แล้วถอดรหัสข้อมูลลงในไฟล์นั้น. เนื่องจากผลลัพธ์ของคำสั่ง uuencode จะแสดงที่ stdout, ดังนั้นต้องรีไดเรกบันทึกลงไฟล์เอง.

base64 ▶ วิธีการเข้ารหัสข้อมูลไบนารีให้เป็นเท็กซ์ ASCII โดยใช้อักขระ 65 ตัวได้แก่ A-Z, a-z, 0-9, +, / และ = [54].

ตัวเลือก -m หมายถึงการใช้การเข้ารหัสแบบ base64 ถ้าไม่ระบุตัวเลือกนี้จะใช้การเข้ารหัสแบบ UU ซึ่งเป็นวิธีการเข้ารหัสที่เก่ากว่าและโดยทั่วไปการเข้ารหัสแบบ base64 จะเป็นที่นิยมมากกว่า UU.

ตัวอย่างต่อไปนี้เป็นการถอดรหัสข้อมูลจากผลลัพธ์ของคำสั่ง uuencode แปลงกลับเป็นข้อมูลไบนารีเดิม.

ตัวอย่างที่ 4.79: แปลงข้อมูลเท็กซ์ ASCII กลับไปไบนารี.

```
$ ls output_image.png↵                          ← ยังไม่มีไฟล์ชื่อ output_image.png
ls: output_image.png: No such file or directory
$ uudecode image.png.txt↵
$ file output_image.png↵
output_image.png: PNG image data, 12 x 12, 8-bit/color RGBA, non-interlaced
```

หลังจากที่คำสั่ง uudecode ทำงานเรียบร้อยแล้ว, จะได้ไฟล์ชื่อ output_image.png ซึ่งชื่อไฟล์นี้เป็นชื่อที่ระบุไว้ในไฟล์ image.png.txt. ถ้าต้องการเปลี่ยนเป็นไฟล์ชื่ออื่น, สามารถใช้ตัวเลือก -o ระบุชื่อไฟล์ผลลัพธ์ที่ต้องการได้.

4.6.4 บันทึกสิ่งที่แสดงในเทอร์มินอล

การทำงานโดยใช้เทอร์มินอลมีข้อดีอย่างหนึ่งคือถ่ายทอดที่เห็น, สิ่งที่พิมพ์ให้คนอื่นรับรู้ได้ง่ายเพราะทุกอย่างเป็นอักขระที่มนุษย์เข้าใจอ่านได้, ก็อปปี้ได้. การบันทึกสิ่งที่พิมพ์หรือสิ่งที่แสดงในหน้าจอทำได้ด้วยคำสั่ง script. คำสั่ง script จะสร้างไฟล์และบันทึกสิ่งที่ผู้ใช้พิมพ์, ผลลัพธ์ที่แสดงทางหน้าจอทุกอย่าง. ผู้ใช้สามารถเก็บไฟล์นี้ไว้ดูภายหลังเก็บไว้เป็นล็อก, หรือส่งให้คนอื่นดู.

script อ้างอิงหน้า 415

ตัวอย่างที่ 4.80: บันทึกหน้าจอเทอร์มินอลด้วย script.

```
$ script↵                          ← ถ้าไม่ระบุชื่อไฟล์จะเก็บทุกอย่างไว้ในไฟล์ typescript
Script started, file is typescript          ← บันทึกทุกอย่างที่เห็นเริ่มจากบรรทัดนี้
$ echo "Everything will be in typescript"↵
Everything will be in typescript
$ exit↵                                                          ← หรือ C-d
Script done, file is typescript                                  ← จบการบันทึก
$ cat typescript↵
Script started on Sun Oct  3 23:52:44 2004      ← เริ่มเนื้อหาของไฟล์ typescript
$ echo "Everything will be in typescript"
Everything will be in typescript
$ exit

Script done on Sun Oct  3 23:53:06 2004          ← จบเนื้อหาของไฟล์ typescript
```

คำสั่ง script บันทึกทุกอย่างที่ปรากฏบนหน้าจอไม่ว่าจะเป็นข้อมูลเข้า, ข้อมูลออกลงในไฟล์ชื่อ typescript ถ้าไม่มีการระบุชื่อไฟล์เป็นอาร์กิวเมนต์. ไฟล์ที่ใช้บันทึกข้อมูลบนเทอร์มินอลไปครั้งหนึ่งแล้วสามารถนำมาเขียนข้อมูลต่ออีกทีด้วย script โดยใช้ตัว

เลือก -a.

-a หมายถึง append.

4.6.5 ส่งอาร์กิวเมนต์ด้วยคำสั่ง xargs

การใช้ไวล์คาร์ดเพื่อแทนชื่อไฟล์หลายไฟล์พร้อมกันเป็นอาร์กิวเมนต์ส่งให้คำสั่งมีความสะดวกอย่างยิ่งถ้าต้องการทำงานรวดเดียว. เช่นคำสั่ง rm * จะลบไฟล์ทุกไฟล์ที่อยู่ในไดเรกทอรีที่ทำงานอยู่. ในบางครั้งถ้าจำนวนไฟล์ในไดเรกทอรีมีมากเกินไปจะเกินข้อผิดพลาดว่า Argument list too long. ในกรณีเช่นนี้สามารถใช้คำสั่ง xargs ช่วยรับข้อมูลจาก stdin แล้วส่งเป็นอาร์กิวเมนต์ในจำนวนที่เหมาะสมให้กับคำสั่งที่ต้องการสั่ง.

xargs อ้างอิงหน้า 419

ตัวอย่างที่ 4.81: ใช้ xargs ส่งอาร์กิวเมนต์ให้คำสั่งอื่น.

```
$ rm -f *↵
-bash: /bin/rm: Argument list too long
$ ls | xargs rm -f↵
```

จากตัวอย่างข้างบน, การสั่งคำสั่ง rm -f * จะเกิดข้อผิดพลาดซึ่งข้อผิดพลาดนี้มาจากเชลล์ซึ่งไม่สามารถแทนค่าไวล์การ์ด * ด้วยจำนวนไฟล์ทั้งหมดที่มีอยู่ในไดเรกทอรีปัจจุบันได้เพราะมีจำนวนมากเกินไป. ในกรณีที่ใช้คำสั่ง xargs, เราจะใช้คำสั่ง ls โดยส่งชื่อไฟล์ทั้งหมดที่มีอยู่ในไดเรกทอรีให้ xargs แล้ว xargs จะแบ่งรายการไฟล์ในจำนวนที่เหมาะสมส่งเป็นอาร์กิวเมนต์ของคำสั่ง rm -f ต่อไป. ข้อมูลนำเข้าของคำสั่ง xargs ในที่นี้เป็นรายการไฟล์ในไดเรกทอรี, ดังนั้นอาจจะใช้คำสั่ง find . แทน ls ก็ได้.

ในระบบปฏิบัติการลินุกซ์จะมีข้อจำกัดของหน่วยความจำที่สามารถใช้กับรายการอาร์กิวเมนต์ของโปรแกรม. ด้วยเหตุนี้เองถ้ามีการใช้ไวล์คาร์ดแทนชื่อไฟล์, เชลล์จะขยายความให้เป็นรายการชื่อไฟล์. ถ้าจำนวนไฟล์มีมากเกินขีดจำกัดก็จะเกิดข้อผิดพลาดตามตัวอย่างข้างบน.

คำสั่ง getconfเป็นคำสั่งใช้แสดงค่าเฉพาะของระบบเช่นถ้าต้องการสำรวจขีดจำกับหน่วยความจำที่ใช้เก็บรายการอาร์กิวเมนต์สามารถทำได้ตามตัวอย่างต่อไปนี้.

ตัวอย่างที่ 4.82: สำรวจขีดจำกัดหน่วยความจำที่ใช้เก็บรายการอาร์กิวเมนต์.

```
$ getconf ARG_MAX↵
131072
```

ARG_MAX คือชื่อตัวแปรที่เก็บค่าขีดจำกัดหน่วยความจำสำหรับรายการอาร์กิวเมนต์ของระบบ. จะเห็นว่าขีดจำกัดหน่วยความจำอยู่ที่ 131,072 ไบต์, หรือแปลงเป็นหน่วนกิโลไบต์คือ 128 กิโลไบต์. ถ้าอาร์กิวเมนต์ทั้งหมดมีขนาดเกิน 128 กิโลไบต์, เชลล์จะแสดงข้อผิดพลาด Argument list too long. ส่วนคำสั่ง xargs จะส่งอาร์กิวเมนต์ให้คำสั่งที่ต้องการโดยมีขนาดไม่เกินขีดจำกัดหน่วยความจำที่ว่านี้. หมายความว่าคำสั่ง xargs จะเรียกใช้คำสั่งเช่นในตัวอย่างได้แก่ rm หลาย ๆครั้งเพื่อไม่ให้เกิดข้อผิดพลาดจำนวนอาร์กิวเมนต์มากเกินไป.

$131072 \div 1024 = 128$

4.7 เชลล์สคริปต์

เชลล์สคริปต์ (shell script) คือไฟล์เท็กซ์ที่รวมคำสั่งเชลล์ต่าง ๆเพื่อใช้ทำงานอย่างใดอย่างหนึ่ง. เชลล์สคริปต์สร้างขึ้นมาเพื่องานเฉพาะอย่างหรือสร้างโปรแกรมใหม่ขึ้นมาโดยอาศัยคำสั่งต่าง ๆที่ใช้ได้ในเชลล์. งานที่เหมาะกับเชลล์สคริปต์มักเป็นงานแบบ batch คือเป็นงานที่มีขั้นตอนที่แน่นอน, และมักไม่ต้องการการโต้ตอบจากผู้ใช้ (หรือจะมีการโต้ตอบกับผู้ใช้ก็ได้). ชื่อเชลล์สคริปต์มีความหมายในตัวอยู่แล้วว่าใช้เชลล์เป็นตัวแปรคำสั่งต่าง ๆที่อยู่ในไฟล์. เชลล์ที่ใช้ในหนังสือเล่มนี้จะหมายถึงเชลล์ bash. ไวย์กรณ์ต่าง ๆจะเข้ากันได้หรือคล้ายกับเชลล์ Bourn, Ksh. สำหรับผู้ที่ต้องการเขียนเชลล์สคริปต์ด้วย csh ต้องอ่านคู่มือหรือหนังสือเกี่ยวกับ csh โดยเฉพาะเนื่องจากมีไวย์กรณ์ที่ต่าง ๆกัน.

4.7.1 สร้างเชลล์สคริปต์

เชลล์สคริปต์เป็นไฟล์เท็กซ์ธรรมดา, สามารถสร้างด้วยบรรณาธิกรณ์ใด ๆก็ได้แล้วแต่สะดวก. บรรทัดแรกของไฟล์ต้องขึ้นต้นด้วยเครื่องหมาย `#!` ตามด้วย full path ของโปรแกรมเชลล์ซึ่งโดยปรกติคือ `/bin/sh`. สำหรับระบบยูนิกซ์ที่มีเชลล์ bash ไว้ในที่อื่น ๆเช่น `/usr/local/bash` ก็ใช้ให้ full path ที่เหมาะสม.

หนังสือบางเล่มเรียก `#!` ว่า Shabang.

ตัวอย่างที่ 4.83: ตัวอย่างการสร้างเชลล์สคริปต์และรัน.

```
$ cat <<EOF > test↵     ← นี่เป็นแค่ตัวอย่าง. ในความเป็นจริงให้ใช้บรรณาธิกรณ์ที่ถนัดสร้าง.
> #!/bin/sh↵                                          ← บรรทัดแรกของเชลล์สคริปต์
> # This is line is a comment.↵
> echo "Hello world!" # From here is comment.↵
> EOF↵
$ ls -l test↵
-rw-r--r--    1 poonlap  users          30 Oct  5 22:20 test
$ chmod +x test↵                                        ← อนุญาตให้กระทำการได้
$ ls -l test↵
-rwxr-xr-x    1 poonlap  users          30 Oct  5 22:20 test
$ ./test↵                  ← เขียน ./ นำหน้าเพื่อระบุว่าอยู่ที่ไดเรกทอรีปัจจุบันให้ชัดเจน
Hello world!
```

ไฟล์ที่เป็นสคริปต์นั้นจะมีชื่ออะไรก็ได้. อาจจะเติมส่วนขยายชื่อไฟล์ด้วย `.sh` ก็ได้เพื่อให้สื่อความหมายให้ชัดเจนว่าเป็นเชลล์สคริปต์. ส่วนขยายชื่อไฟล์นี้จะเป็นอะไรก็ได้ `.pl`, `.exe`, `.a` ฯลฯ หรือไม่มีก็ได้เพราะลินุกซ์ไม่ได้แยกแยกประเภทไฟล์ด้วยชื่อไฟล์. หลังจากที่สร้างไฟล์เรียบร้อยแล้ว, ยังไม่สามารถกระทำการได้ทันที. ต้องใช้คำสั่ง `chmod` เพื่อตั้งสิทธิ์การใช้ไฟล์ให้กระทำการได้. หลังจากนั้นจึงสามารถรันเชลล์สคริปต์ได้เหมือนโปรแกรมทั่วไป. ให้สังเกตว่าในตัวอย่างจะรันเชลล์สคริปต์ด้วย `./test` แทนที่จะใช้ `test` เป็นการระบุให้ชัดเจนว่า `test` เป็นไฟล์ที่อยู่ในไดเรกทอรีปัจจุบัน `./` มิเช่นนั้น `test` จะหมายถึงคำสั่ง `/usr/bin/test`.

สรุปขั้นตอนการสร้างเชลล์สคริปต์.

1. ใช้บรรณาธิกรณ์ที่ถนัดสร้างเชลล์สคริปต์. บรรทัดแรกต้องขึ้นต้นด้วย `#!` แล้วตามด้วย full path ของเชลล์เช่น `#!/bin/sh`.

2. ตั้งค่าสิทธิ์การใช้ไฟล์เชลล์สคริปต์ให้กระทำการได้ (executable) ด้วยคำสั่ง `chmod +x`.

3. รันเชลล์สคริปต์. เพื่อความชัดเจนให้ระบุเป็น path สัมพันธ์เช่น `./test` เป็นต้น.

4.7.2 หมายเหตุ

หมายเหตุ (comment) คือส่วนที่เชลล์จะไม่ตีความ. เราสามารถใช้หมายเหตุเขียนบันทึกเตือนจำ, หรือเขียนอธิบายการทำงานของเชลล์สคริปต์ก็ได้. ส่วนที่เป็นหมายเหตุในเชลล์จะเป็นส่วนที่อยู่ถัดจากเครื่องหมาย # ไปจนสุดบรรทัด. เครื่องหมาย # ไม่จำเป็นต้นอยู่ต้นบรรทัดก็ได้.

4.7.3 ตัวแปร

ตัวแปร (variable) มีความสำคัญอย่างยิ่งในการเขียนโปรแกรมรวมถึงเชลล์สคริปต์ด้วย. ชื่อตัวแปรไม่ควรมีตัวอักษรที่มีความหมายพิเศษสำหรับเชลล์. ตัวแปรในเชลล์สคริปต์ไม่มี*ประเภท (type)*, กล่าวคือตัวแปรสามารถเก็บข้อมูลเช่นเท็กซ์, ตัวเลข ได้โดยที่ไม่ต้องระบุให้เชลล์รู้แน่ชัดว่าค่าที่อยู่ในตัวแปรนั้นเป็นอะไร (ประเภทอะไร).

การกำหนดตัวแปร

การกำหนดตัวแปรในเชลล์จะใช้เครื่องหมาย = มีไวยกรณ์ดังนี้

```
variable=value
```

ให้สังเกตว่าตำแหน่งข้างหน้าและหลังเครื่องหมาย = ต้องไม่มีช่องว่างเพราะเชลล์จะถือว่าช่องว่างเป็นตัวแบ่งอาร์กิวเมนต์. ค่าตัวแปรที่เป็นสายอักขระมักจะใช้ double quote หรือ single quote คล่อม. ส่วนค่าตัวแปรที่เป็นตัวเลขมักจะเขียนเป็นตัวเลขโดด ๆ. นอกจากนั้นค่าของตัวแปรอาจจะตั้งได้จากผลลัพธ์ของคำสั่งอื่น ๆโดยใช้การแทนค่าคำสั่ง (command substitution). ชื่อของตัวแปรไม่ควรใช้ตัวอักขระที่มีความหมายพิเศษสำหรับเชลล์. แนะนำให้ใช้ตัวอักษรภาษาอังกฤษผสมกับตัวเลขหรือเครื่องหมาย _ เวลาตั้งชื่อตัวแปร.

ตัวอย่างที่ 4.84: การกำหนดตัวแปรในเชลล์สคริปต์.

```
c="Hello world"                                    ← ใช้ quote เพราะมีช่องว่าง
file=test.sh                                              ← ไม่ใช้ quote
my_count=1                          ← ค่าตัวแปรเป็นตัวเลข. จะใช้ quote หรือไม่ก็ได้
day1=‘date‘          ← ค่าตัวแปรเป็นผลลัพธ์ของคำสั่ง date มีผลเหมือนกับ day1=${date}
```

การอ้างอิงค่าตัวแปร

เวลาแสดงค่าตัวแปรจะใช้เครื่องหมาย $ นำหน้าชื่อตัวแปร.

ตัวอย่างที่ 4.85: การแสดงค่าตัวแปรเชลล์.

```
echo $c                                                    ← ผลลัพธ์ Hello world
echo ${file}ell                                             ← ผลลัพธ์ test.shell
echo "$my_count"                                                    ← ผลลัพธ์ 1
echo Today is $day1                 ← ผลลัพธ์ Today is Tue Oct 5 23:59:33 JST 2004
```

ผู้ใช้สามารถระบุช่วงที่เป็นตัวแปรได้ด้วยเครื่องหมาย {} คร่อมส่วนที่เป็นชื่อตัวแปรที่ต้องการ. เช่น `${file}ell` หมายถึงการแสดงค่าของตัวแปร `$file` ถ้าไม่มีเครื่องหมายวงเล็บ {} จะตีความเป็นค่าของตัวแปร `$fileell` แทน. สำหรับค่าของตัวแปรที่ที่ไม่มีจริง, หรือไม่ได้กำหนดไว้จะเป็นความว่างเปล่า.

ตัวแปรพิเศษ

ในเชลล์ bash จะมีตัวแปรพิเศษซึ่งใช้อ้างอิงและไม่สามารถตั้งค่าได้. ตัวแปรเหล่านี้ได้แก่

- $0 หมายถึงชื่อของเชลล์สคริปต์.
- $1, $2, ... ค่าอาร์กิวเมนต์ของเชลล์สคริปต์ตัวที่ 1, 2, ... ไปเรื่อย ๆ.
- $* ค่าของอาร์กิวเมนต์ทั้งหมดแบ่งด้วยเครื่องหมาย space. ถ้าใช้เครื่องหมาย double quote คล่อมจะเป็นการรวมอาร์กิวเมนต์ทุกตัวให้เป็นค่าค่าเดียวและใช้เครื่องหมายที่กำหนดในตัวแปร IFS แบ่งระหว่างอาร์กิวเมนต์.
- $@ ค่าของอาร์กิวเมนต์ทั้งหมดแบ่งด้วยเครื่องหมาย space. ถ้าใช้เครื่องหมาย double quote คล่อมจะเป็นการรวมอาร์กิวเมนต์ทุกตัวให้เป็นค่าค่าเดียวและใช้เครื่องหมาย space แบ่งระหว่างอาร์กิวเมนต์.
- $# จำนวนอาร์กิวเมนต์ของเชลล์สคริปต์.
- $? ค่าสถานะจบการทำงานของคำสั่งล่าสุด.
- $$ โปรเซส ID ของเชลล์ที่ทำงานอยู่.

ตัวอย่างที่ 4.86: แสดงค่าตัวแปรพิเศษของเชลล์.

```
$ cat -n special_vars.sh↵
     1  #!/bin/sh
     2  IFS=":"
     3  echo Script name: $0
     4  echo Arguments: "$*"
     5  echo Arguments: "$@"
     6  echo Number of arguments: $#
     7  echo Process ID: $$
$ ./special_vars.sh arg1 arg2↵
Script name: ./special_vars.sh
Arguments: arg1:arg2
Arguments: arg1 arg2
Number of arguments: 2
Process ID: 28842
```

นอกจากนี้ยังมีตัวแปรเชลล์ซึ่งเป็นสื่อความหมายเช่น

- $BASH ชื่อไฟล์เต็มของเชลล์ที่กระทำการอยู่.
- $BASH_VERSION รุ่นของเชลล์ bash.
- $HOSTNAME ชื่อโฮสของเครื่องที่ทำงานอยู่.
- $OLDPWD ไดเรกทอรีก่อนไดเรกทอรีปัจจุบัน.
- $PPID ID ของโปรเซสแม่ที่กระทำการเชลล์.
- $RANDOM ค่าสุ่มตัวเลขระหว่าง 0 ถึง 32767.
- $SECONDS จำนวนวินาทีตั้งแต่เชลล์ทำงาน.
- $UID uid ของผู้ที่ใช้เชลล์.
- $IFS ย่อมาจากคำว่า Internal Field Separator. เป็นตัวแปรที่กำหนดอักขระที่ใช้แบ่งคำ. ค่าปริยายจะเป็น space, tab และ newline.

รายการตัวแปรทั้งหมดและรายละเอียดของตัวแปรเหล่านี้สามารถอ่านได้จากคู่มือของ bash(1).

4.7.4 การใช้เครื่องหมายคำพูด

Quoting คือการเขียนสายอักขระโดยใช้เครื่องหมายคำพูดคล่อม. เครื่องหมายคำพูดนี้ยังแบ่งเป็น double quote และ single quote ซึ่งให้ผลต่างกัน.

สายอักขระที่คร่อมด้วยเครื่องหมาย double quote, สิ่งที่เขียนจะมีความหมายตามตัวอักษรยกเว้น $, ‘ และ \ ที่เชลล์จะขยายความพิเศษ. ถ้ามีเครื่องหมาย $ อยู่ใน double quote, เชลล์จะขยายความว่ามีการอ้างอิงตัวแปร. ถ้ามีเครื่องหมาย ‘ (back quote) , เชลล์จะขยายความว่ามีการแทนผลลัพธ์คำสั่ง. สำหรับเครื่องหมาย \ ที่อยู่ใน double quote, เชลล์จะขยายความพิเศษก็ต่อเมื่อมีเครื่องหมาย $, ‘, ", \ หรืออักขระ newline (เกิดจากการกดคีย์ Enter). ถ้าอักขระที่ตามหลัง \ เป็นอักขระอื่น ๆนอกเหนือกจากนี้เช่น \n, ก็จะมีความหมายตามตัวอักษรคือเครื่องหมาย \ แล้วตามด้วย n. สายอักขระที่คร่อมด้วยเครื่องหมาย single quote จะมีความหมายตามตัวอักษรทุกอย่าง.

ตัวอย่างที่ 4.87: Quoting แบบต่าง ๆ.

```
$ cat -n quote.sh↵
     1  #!/bin/sh
     2  echo "Today is ‘date‘."
     3  echo "UID is $UID."
     4  echo "Dollar sign \$, Back quote \‘, Double quote \", Backslash \\"
     5  echo "There is no special meaning \n"
     6  echo 'Dollar signe \$, Back quote \‘, Double quote \", Backslash \\'
$ ./quote.sh↵
Today is Thu Oct  7 22:24:24 JST 2004.
UID is 1000.
```

```
Dollar sign $, Back quote `, Double quote ", Backslash \
There is no special meaning \n
Dollar signe \$, Back quote \`, Double quote \", Backslash \\
```

สำหรับอักขระที่ไม่สามารถพิมพ์ด้วยแป้นพิมพ์สามารถเขียนได้ด้วยเขียนเครื่องหมาย $ แล้วตามด้วย *'string'* โดยที่ *string* เป็นสายอักขระ escape sequence ที่แสดงตารางที่ 4.16.

ตารางที่ 4.16: วิธีใช้อักขระพิเศษในเชลล์.

สายอักขระ	วิธีใช้	ความหมาย
\a	$'\a'	alert เสียงกระดิ่ง.
\b	$'\b'	backspace.
\e	$'\e'	escape.
\f	$'\f'	form feed.
\n	$'\n'	new line.
\r	$'\r'	carriage return.
\t	$'\t'	(horizontal) tab.
\v	$'\v'	vertical tab.
\'	$'\''	single quote.
nnn	$'*nnn*'	อักขระแสดงด้วยเลขฐานแปด *nnn*.
\x*HH*	$'\x*HH*'	อักขระแสดงด้วยเลขฐานสิบหก *HH*.
\c*x*	$'\c*x*'	อักขระ control-*x*.

ตัวอย่างที่ 4.88: ใช้ตัวอักขระพิเศษในเชลล์สคริปต์.

```
$ cat -n tab_nl.sh↵
     1  #!/bin/sh
     2  echo a$'\t'b # a<tab>b
     3  echo a$'\n'b # a<newline>b
$ ./tab_nl.sh↵
a       b
a
b
```

4.7.5 แถวลำดับ

แถวลำดับ (array) เป็นตัวแปรที่เก็บข้อมูลเป็นลำดับโดยใช้เลข*ดรรชนี (index)* ระบุตำแหน่งของข้อมูลที่ต้องการ. ดรรชนีของตัวแปรแถวลำดับจะเริ่มต้นจาก 0. การสร้างแถวลำดับในเชลล์จะใช้คำสั่งภายใน `declare` ประกาศตัวแปรให้เป็นตัวแปรแถวลำดับ.

```
declare -a myArr                                    ← -a หมายถึง array
```

หรือจะกำหนดค่าไปเลย

```
myArr[index]=value                                    ← index จะเริ่มจากตัวเลขอะไรก็ได้
```

เราสามารถกำหนดค่าของสมาชิกในแถวลำดับพร้อม ๆกันได้โดยใช้เครื่องหมายวงเล็บ, แล้วกำหนดค่าของแถวลำดับแต่ละตำแหน่งโดยใช้ช่องว่างคั่น.

```
myArr=(value1 value2 ...)
```

การอ้างอิงตัวแปรแถวลำดับจะใช้วงเล็บปีกกาคล่อมตัวแปร.

```
${myArr[index]}
```

ตัวอย่างเชลล์สคริปต์ต่อไปจะแสดงผลสุ่มซึ่งจะให้ผลไม่เหมือนกันแต่ละครั้งที่กระทำการ.

ตัวอย่างที่ 4.89: การใช้ตัวแปรแถวลำดับในเชลล์.

```
$ cat lucky_color.sh↵
     1  #!/bin/sh
     2  color=( "red"
     3           "green"
     4           "blue"
     5           "white"
     6           "black"
     7           "yellow"
     8           "pink" )
     9  echo Your lucky color is ${color[$(($RANDOM % 7))]}.
$ ./lucky_color.sh↵
Your lucky color is blue.
$ ./lucky_color.sh↵
Your lucky color is pink.
```

เชลล์สคริปต์นี้ในช่วงแรกเป็นการสร้างตัวแปรแถวลำดับชื่อ `color` มีสมาชิก 7 ตัว. การแสดงผลใช้ `echo` แสดงค่าของตัวแปรโดยที่ดรรชนีของแถวลำดับมาจากค่าสุ่มตัวแปร `$RANDOM`. เนื่องจากเราต้องการดรรชนีของแถวลำดับให้มีค่าสุ่มตั้งแต่ 0 ถึง 6 จึงใช้การหาเศษจากการหารด้วย 7 เป็นดรรชนี.

4.7.6 การควบคุมขั้นตอนคำสั่ง

ภาษาคอมพิวเตอร์ทั่วไปจะมีไวยกรณ์สำหรับควบคุมขั้นตอนการทำงาน. เชลล์ก็มีไวยกรณ์สำหรับควบคุมการทำงานเช่นกันแต่จะเรียกว่าเป็น*คำสั่งผสม (compound command)* ไม่ใช่คำสั่งที่พิมพ์หลังเชลล์พรอมต์แล้วใช้งานได้ทันที. ต้องมีคำ, อาร์กิวเมนต์ส่วนอื่น ๆประกอบกันจึงจะใช้ได้.

เงื่อนไข

การแบ่งการทำงานโดยใช้เงื่อนไขในเชลล์จะใช้คำสั่ง `if`

```
if list; then list; [ elif list; then list; ] ... [ else list; ] fi
```

เครื่องหมาย ; เป็นตัวแบ่งคำสั่งซึ่งจะเป็นการขึ้นบรรทัดใหม่ก็ได้. ดังนั้นจึงสามารถเขียนให้เข้าใจง่ายขึ้นดังนี้.

```
if list
    then
    list
    ...
[ elif list
    then
    list ]
    ...
[ else
    list ]
    ...
fi
```

คำสั่งที่อยู่ในเครื่องหมายปีกกาเหลี่ยมเป็นตัวเลือกคือจะมีหรือไม่มีก็ได้แล้วแต่กรณี. เวลาใช้งานจริงไม่ต้องเขียนวงเล็บเหลี่ยม. *list* เป็นคำสั่งที่ใช้ในเชลล์.

บรรทัด `if` *list* เป็นการสร้างเงื่อนไข, ถ้าสถานะจบการทำงานของ *list* เป็นศูนย์ (ไม่มีข้อผิดพลาด) ก็จะกระทำการคำสั่ง *list* ที่ตามหลัง `then` ต่อไป. ในทางกลับกันถ้าสถานะจบการทำงานของ *list* ในบรรทัด `if` *list* ไม่เป็นศูนย์ (มีข้อผิดพลาด), ก็จะกระทำการบรรทัด `elif` หรือ `else` ต่อไป. การจบคำสั่ง `if` จะลงท้ายด้วย `fi` ซึ่งเป็นการกลับตัวอักษรของ `if`.

ตัวอย่างที่ 4.90: การใช้ `if` ในเชลล์สคริปต์.

```
$ cat -n if_sample.sh↵
     1  #!/bin/sh
     2  if [ $UID -eq 0 ]
     3      then
     4      echo You are root.
     5  else
     6      echo You are ‘awk -F: "/^.*:.*:$UID:/ print \\$1" /etc/passwd‘.
     7  fi
$ ./if_sample.sh↵
You are poonlap.
```

คำสั่งที่ใช้กำหนดเงื่อนไขอีกแบบคือ `case`. การใช้ `case` จะเป็นการจับคู่ระหว่างคำกับแบบอย่างที่กำหนดไว้, สะดวกกว่าการใช้ `if` ในกรณีที่แบ่งกรณีเน้นการจับคู่. เครื่องหมาย ;; เป็นการบอกให้เชลล์รู้ว่าจบคำสั่งของแบบอย่างที่จับคู่ได้.

```
case word
in
```

```
    pattern)
    list;;
    ...
esac
```

ต่อไปนี้เป็นตัวอย่างการใช้ case พิจารณาค่าตัวแปรที่รับมาจากการป้อนข้อมูลของผู้ใช้แล้วแยกกรณีตามอักษรที่รับมา.

ตัวอย่างที่ 4.91: การใช้ case ในเชลล์สคริปต์.

```
$ cat -n case_sample.sh↵
     1  #!/bin/sh
     2  echo Please select one letter
     3  echo "a b c"
     4  while [ 1 ]                                    ← วงวนไม่รู้จบ
     5  do
     6  echo -n "> "
     7  read x
     8  case $x
     9  in
    10      a)
    11      echo You selected a.
    12      break;;                              ← สำหรับออกจากวงวน while
    13      b)
    14      echo You selected b.
    15      break;;
    16      c)
    17      echo You selected c.
    18      break;;
    19      *)                                       ← จับคู่ $x กับอะไรก็ได้
    20      echo Please select again.;;
    21  esac
    22  done
$ ./case_sample.sh↵
Please select one letter
a b c
> d
Please select again.
> a
You selected a.
```

ให้สังเกตว่าถ้าค่า $x จับคู่กับคำที่แบ่งเงื่อนไขเช่น a แล้วก็จะกระทำการเฉพาะคำสั่งที่อยู่ในช่วงนั้นเท่านั้น, จะไม่จับคู่กับ * ซึ่งอยู่ถัดไปอีก.

วงวน

วงวนที่ใช้ในเชลล์มีหลายเหมือนหรือคล้ายกับภาษาคอมพิวเตอร์อื่น ๆ ได้แก่ for, while และ until.

วงวนคำสั่ง for มีไวยกรณ์ 2 แบบ. แบบแรกจะสร้างตัวแปร *variable* สำหรับใช้ในวงวน for. ในแต่ละรอบของการวนค่าของตัวแปร *variable* จะเปลี่ยนตามค่าที่กำหนดไว้ใน *list* ตามลำดับ. list นี้จะเป็นชุดของคำหรือค่าหลายตัวแบ่งด้วยช่องว่าง. วงวน for แบบแรกมีรูปแบบดังนี้.

```
for variable in list
do
    command
    ...
done
```

ตัวอย่างต่อไปนี้เป็นสคริปต์สำหรับแปลงไฟล์รูปภาพ gifให้เป็นไฟล์รูปภาพ jpg.

gif ▶
เป็นคำย่อของ Graphics Interchange Format ฟอร์แมตไฟล์รูปภาพบิตแมปสี. มีการอัดบีบข้อมูล, ใช้ดรรชนีสีแสดงสีที่มีอยู่ในรูป. ไฟล์รูปแบบนี้เป็นนิยมใช้ในเว็บแต่หลังจากที่มีปัญหาสิทธิบัตรเกี่ยวกับการบีบอัดข้อมูล, ไฟล์รูปแบบอื่นเช่น jpg และ png ก็เป็นที่นิยมถัดมา.

jpg ▶
เป็นคำย่อของ Joint Photographic Experts Group. รูปภาพแบบบิตแมปที่มีการอัดบีบข้อมูลและมักใช้กับไฟล์รูปภาพบนอินเทอร์เน็ต.

ตัวอย่างที่ 4.92: ตัวอย่างเชลล์สคริปต์ที่ใช้ for.

```
$ cat -n gif2jpg.sh↵
     1  #!/bin/sh
     2  CONVERT=/usr/bin/convert
     3  for file in $*
     4  do
     5      echo Converting $file to jpg ...
     6      $CONVERT $file `basename $file .gif`.jpg
     7  done
$ ls -F↵
gif2jpg.sh*  pic01.gif  pic02.gif
$ ./gif2jpg.sh pic*gif↵
Converting pic01.gif to jpg ...
Converting pic02.gif to jpg ...
$ ls -F↵
gif2jpg.sh*  pic01.gif  pic01.jpg  pic02.gif  pic02.jpg
```

วงวน for แบบที่สองคล้ายกับวงวน for ในภาษาซี, คือจะมีนิพจน์สามส่วนซึ่งมีรูปแบบดังต่อไปนี้.

```
for (( expr1 ; expr2 ; expr3 ))
do
    command
    ...
done
```

นิพจน์ expr1, expr2 และ expr3 เป็นนิพจน์ที่เกี่ยวกับการคำนวณตัวแปรในแต่ละรอบ, สามารถใช้ตัวปฏิบัติที่แสดงในตารางที่ 4.3. เมื่อเข้าสู่วงวน for, เชลล์จะตีความ expr1 เป็นอันดับแรก, และจะตีความ expr2 เป็นเรื่อย ๆจนกว่าค่าสถานะจบการทำงานของ expr2 เป็นศูนย์, ถ้าไม่เป็นศูนย์ก็จะตีความ expr3 ต่อไป.

ตัวอย่างที่ 4.93: เชลล์สคริปต์แสดงลำดับ Fibonacci

```
$ cat -n fibonacci.sh↵
     1  #!/bin/sh
     2  n=$1
     3  for (( i=0 ; $i < $n ; i++ ))
     4  do
     5          if [ $i = 0 ]
     6          then
     7                  echo -n "1 "
```

```
 8                  aa=0
 9                  a=1
10          else
11                  w=$(($aa+$a))
12                  echo -n "$w "
13                  aa=$a
14                  a=$w
15
16          fi
17  done
$ ./fibonacci.sh 20↵
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

เชลล์สคริปต์ที่แสดงในตัวอย่างที่ 4.93 จะแสดงลำดับ Fibonacci ตามจำนวนที่ระบุเป็นอาร์กิวเมนต์ของสคริปต์. ในสคริปต์จะใช้วงวน `for` จะเปลี่ยนค่าตัวแปร `i` จนครบตามจำนวนที่ระบุ.

Fibonacci เป็นลำดับที่ค่าของลำดับตัวสุดท้ายจะเป็นผลบวกสองตัวของลำดับก่อนหน้านั้น, เช่น $5 = 3 + 2$.

วงวน `while` และ `until` เป็นวงวนที่กระทำการไปเรื่อย ๆจนกว่าข้อแม้ที่กำหนดจะเป็นเท็จ (ค่าสถานะจบการทำงานไม่เท่ากับ 0). วงวน `until` จะคล้ายกับ `while` เพียงแต่ช่วงที่ตรวจสอบข้อแม้จะเป็นการตรวจสอบแบบนิเทส (negation).

```
while condition
do
    command
    ...
done
```

```
until condition
do
    command
    ...
done
```

ตัวอย่างที่ 4.94: การใช้ `while` ในเชลล์สคริปต์.

```
$ cat -n fibonacci_while.sh↵
 1  #!/bin/sh
 2  n=$1
 3  i=0 # กำหนดค่า i ก่อนเข้าวงวน
 4  while (($i < $n)) # หรือ [ $i -lt $n]
 5  do
 6          if (($i == 0)) # หรือ [ $i = 0 ]
 7          then
 8                  echo -n "1 "
 9                  aa=0
10                  a=1
11          else
12                  w=$(($aa+$a))
13                  echo -n "$w "
14                  aa=$a
15                  a=$w
16
```

```
    17          fi
    18          ((i++)) # เพิ่มค่า i เพื่อใช้ในรอบต่อไป
    19  done
    20  echo
$ ./fibonacci_while.sh 20↵
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

ตัวอย่างข้างบนแสดงลำดับ Fibonacci เหมือนกับตัวอย่างที่ 4.93 แต่ใช้วงวน `while` แทน `for`.

4.7.7 ฟังก์ชัน

เราสามารถนิยามฟังก์ชันได้ด้วยคำสั่ง `funtion` คล้ายกับภาษาคอมพิวเตอร์อื่น ๆ.

```
[ function ] name () {
    command
    ...
}
```

คำว่า `function` เป็นตัวเลือกซึ่งจะเขียนหรือไม่ก็ได้. หลังจากที่นิยามฟังก์ชันแล้วจะสามารถใช้ชื่อฟังก์ชันนั้นได้เหมือนกับคำสั่งทั่วไป. ตัวแปร $1, $2, ... เป็นตัวแปรที่ใช้ในการอ้างอิงค่าของอาร์กิวเมนต์ของฟังก์ชัน.

ตัวอย่างที่ 4.95: การนิยามฟังก์ชันในเชลล์.

```
$ cat -n fibonacci_function.sh↵
     1  #!/bin/sh
     2  function fibonacci () {
     3      local n=$1
     4      local i=0
     5      local aa a w
     6
     7      while (($i < $n))
     8        do
     9        if (($i == 0))
    10            then
    11            echo -n "1 "
    12            aa=0
    13            a=1
    14        else
    15            w=$(($aa+$a))
    16            echo -n "$w "
    17            aa=$a
    18            a=$w
    19
    20        fi
    21        ((i++))
    22      done
    23      echo
    24  }
    25
    26  fibonacci $1
```

```
$ ./fibonacci_function.sh 20↵
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

จากตัวอย่างข้างบนมีการใช้คำสั่ง `local` เป็นคำสั่งที่ระบุตัวแปรที่ใช้ว่าเป็น*ตัวแปรเฉพาะที่ (local variable)* ใช้ในฟังก์ชันเท่านั้น. ตัวแปรที่ประกาศด้วยคำสั่ง `local` จะไม่สามารถอ้างอิง, ตั้งค่าใหม่นอกฟังก์ชัน. ถ้าไม่ใช้คำสั่ง `local` ช่วยนิยามตัวแปร, ตัวแปรนั้นจะเป็น*ตัวแปรส่วนกลาง (global variable)* คือใช้อ้างอิง, ตั้งค่าจากที่ไหนก็ได้ในสคริปต์.

นอกจากการนิยามฟังก์ชันในเชลล์สคริปต์แล้ว, เรายังสามารถนิยามฟังก์ชันในไฟล์ตั้งค่าเริ่มต้นของเชลล์เช่นในไฟล์ `.bash_profile` เพื่อสร้างคำสั่งเฉพาะตามที่ต้องการได้ด้วย.

4.7.8 เรียนรู้เชลล์สคริปต์จากตัวอย่าง

วิธีเรียนการเขียนเชลล์สคริปต์ที่ดีอย่างหนึ่งคืออ่านเชลล์สคริปต์ที่มีอยู่แล้วและทำความเข้าใจ. ตัวอย่างต่อไปนี้เป็นการรวมคำสั่งต่าง ๆ ที่ได้แนะนำไปแล้วมารวมกันเขียนเป็นเชลล์สคริปต์. เชลล์สคริปต์นี้จะแสดงโลโกของ X วินโดว์วินาทีละหนึ่งหน้าต่างด้วยการรันโปรแกรม `xlogo`. ถ้ามีหน้าต่าง `xlogo` มากกว่า 5 หน้าต่างโดยปริยายจะปิดหน้าต่างแรกที่แสดงตามลำดับแล้วสร้างหน้าต่าง `xlogo` ใหม่ทดแทนไปเรื่อย ๆ. จำนวนหน้าต่างที่ต้องการแสดงสามารถระบุได้จากอาร์กิวเมนต์ของสคริปต์ด้วย.

การแสดงหน้าต่าง `xlogo` จะใช้สีฉากหน้าและฉากหลังด้วยการสุ่มเลือกสี, และสุ่มเลือกตำแหน่งที่แสดงให้อยู่ในบริเวณหน้าจอ. รูปที่ 4.8 แสดงหน้าจอการทำงานของเชลล์สคริปต์.

ในเชลล์สคริปต์ที่จะอธิบายต่อไปนี้มีการเรียกใช้โปรแกรมที่ต้องแนะนำก่อนได้แก่โปรแกรม `showrgb`, `xdpyinfo`, `xlogo`, `pgrep` และ `kill`.

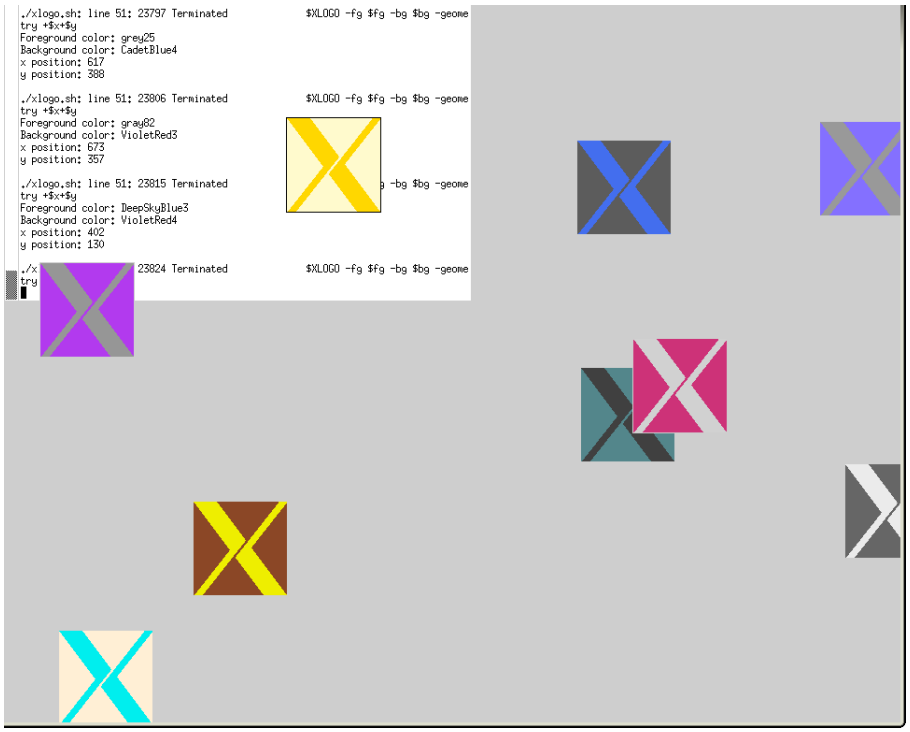
`showrgb`

โปรแกรม `showrgb` เป็นโปรแกรมคำสั่งที่แสดงค่า *RGB* ของสีแสดงและชื่อของสีนั้น ๆ. แม่สีที่ใช้ได้แก่แสงสีแดงจะมีค่าเป็น 255 0 0, แสงสีเขียวมีค่าเป็น 0 255 0 และแสงสีน้ำเงินมีค่าเป็น 0 0 255. ค่าเหล่านี้จะแสดงด้วยบิตแปดตัวซึ่งมีค่าต่ำสุดเป็น 0 และค่าสูงสุดเป็น 255. การสร้างสีอื่นโดยใช้แม่สีสามารถทำได้โดยนำแม่สีความเข้มต่าง ๆ มาผสมกัน. เช่นสีขาวจะมีค่าเป็น 255 255 255, สีดำจะมีค่าเป็น 0 0 0.

RGB ▶
เป็นวิธีการแสดงสีต่างด้วยแสงโดยใช้แม่สีได้แก่สีแดง (Red), สีเขียว (Green) และสีน้ำเงิน (Blue). จะใช้ค่าแปดบิตซึ่งมีค่าตั้งแต่ 0 ถึง 255 แทนความเข้มของแม่สี. เช่นสีแดงมีค่าเป็น 255 0 0, สีเขียวมีค่าเป็น 0 255 0, และสีน้ำเงินมีค่าเป็น 0 0 255. ในทฤษฎีแสง, ถ้านำแม่สีเหล่านี้มาผสมกันจะได้สีขาว.

ตัวอย่างที่ 4.96: แสดงค่าและชื่อสีต่าง ๆ ที่กำหนดไว้ใน X วินโดว์.

```
$ showrgb↵
255 250 250             snow
248 248 255             ghost white
248 248 255             GhostWhite
245 245 245             white smoke
245 245 245             WhiteSmoke
...
```



รูปที่ 4.8: ผลของเชลล์สคริปต์ในตัวอย่างที่ 4.98.

จริงๆแล้วคำสั่ง `showrgb` เป็นเพียงโปรแกรมที่แสดงข้อมูลที่อยู่ในไฟล์ `/usr/X11R6/lib/X11/rgb.txt` ซึ่งมีเนื้อหาเหมือนกับผลลัพธ์ของคำสั่ง `showrgb`. ในระบบ X วินโดว์จะรับรู้ชื่อสีกำหนดไว้ในไฟล์นี้. ให้สังเกตุว่าชื่อสีบางชื่อซ้ำกันเช่น “ghost white” ชื่อซ้ำกับ GhostWhite เป็นต้น.

xdpyinfo

`xdpyinfo` เป็นโปรแกรมคำสั่งที่แสดงข้อมูลของ display ของ X วินโดว์. คำสั่งนี้จะแสดงค่าต่างๆของ X เซิฟร์เวอร์เช่นความสามารถ, ขนาดของหน้าจอ, ความละเอียดของสี เป็นต้น.

ตัวอย่างที่ 4.97: สำรวจคุณสมบัติและค่าเฉพาะต่างของ X เซิฟร์เวอร์.

```
$ xdpyinfo↵
...
    XKEYBOARD
    XTEST
    XVideo
default screen number:    0
number of screens:    1

screen #0:
  dimensions:    1280x1024 pixels (382x313 millimeters)
  resolution:    85x83 dots per inch
  depths (7):    24, 1, 4, 8, 15, 16, 32
  root window id:    0x40
```

```
    depth of root window:    24 planes
...
```

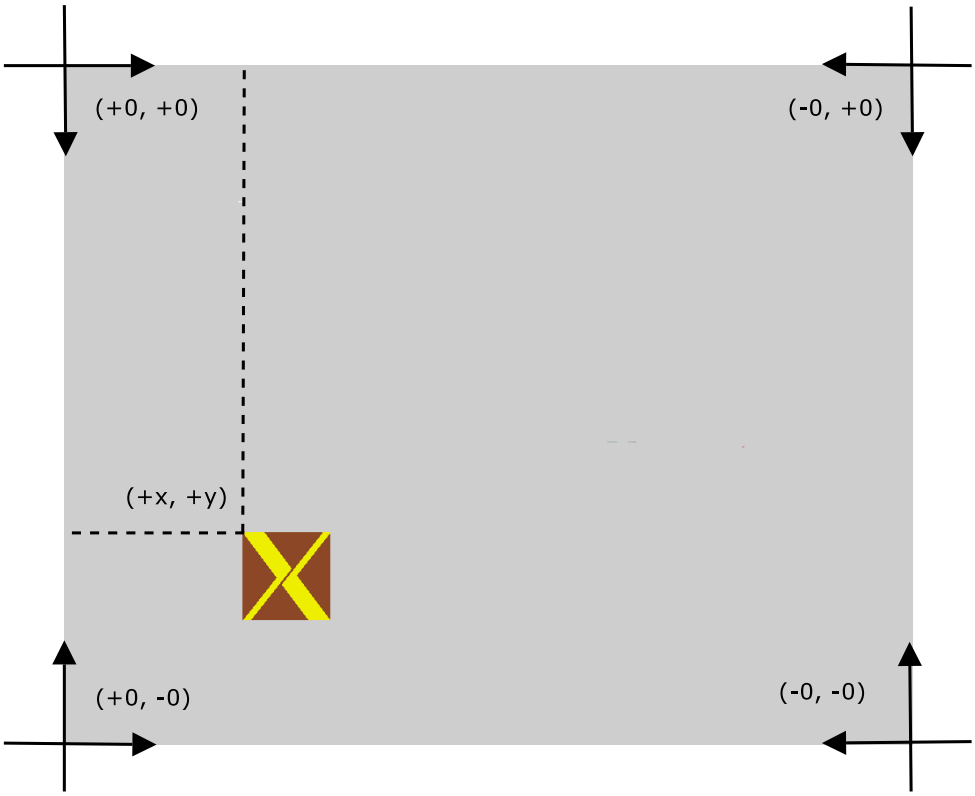
ในสคริปต์จะตรวจสอบขนาดของหน้าจอในหน่วย *พิกเซล (pixel)* เพื่อคำนวณตำแหน่งที่แสดงหน้าจอ `xlogo` ไม่ให้อยู่นอกหน้าจอ.

pixel ▶
หมายถึงจุดที่ประกอบกันเป็นรูปหรือหน่วยที่บอกขนาดของรูป. ขนาดของจุดไม่มีขนาดสัมบูรณ์ขึ้นอยู่กับอุปกรณ์ที่ใช้แสดง. เพราะฉะนั้นถ้าพูดถึงรูปขนาด 16x16 พิกเซลจะมีขนาดต่างกันเมื่อพิมพ์ออกทางเครื่องพิมพ์เทียบกับหน้าจอ.

xlogo

`xlogo` เป็นโปรแกรม X วินโดว์มาตรฐานที่แสดงโลโกบนหน้าจอ. โปรแกรม X วินโดว์มาตรฐานจะมีตัวเลือกเหมือน ๆกันเช่น

- `-fg` หรือ `--foreground` ระบุสีของฉากหน้า (foreground) ของรูปหรือวัตถุที่แสดง.
- `-bg` หรือ `--background` ระบุสีของฉากหลัง (background) ของรูปหรือวัตถุที่แสดง.
- `-geometry +`*x*`+`*y* ระบุตำแหน่งหน้าต่างพิกัด x และ y ในหน่วยพิกเซลโดยที่จุด (0, 0) อยู่ที่มุมซ้ายบนของหน้าจอ.



รูปที่ 4.9: ระบบพิกัดใน X วินโดว์.

คำสั่ง pgrep ไว้ใช้หาหมายเลขโปรเซสโดยระบุชื่อคำสั่งแบบ regular expression. และคำสั่ง kill ใช้สำหรับจบการทำงานของโปรเซสโดยระบุหมายเลขโปรเซสที่ต้องการ. รายละเอียดของคำสั่ง pgrep และ kill จะอยู่ในบทถัดไป.

ตอนนี้เรามาดูเชลล์สคริปต์ชื่อ xlogo.sh ในตัวอย่างที่ 4.98.

ตัวอย่างที่ 4.98: ตัวอย่างเชลล์สคริปต์แสดงและควบคุมจำนวน xlogo.

```
$ cat -n xlogo.sh↵
     1  #!/bin/sh
     2  #
     3  #  File name: xlogo.sh
     4  #  Description: Show one xlogo window per second. If there are more than
     5  #  5 windows (default), it will kill the first one and keep the number of
     6  #  window to 5.
     7  #
     8  #  Location and color of xlogo are random.
     9
    10  if [ -z $1 ]
    11  then
    12      N=5
    13  else
    14      N=$1
    15  fi
    16
    17  SHOWRGB=/usr/X11R6/bin/showrgb
    18  XDPYINFO=/usr/X11R6/bin/xdpyinfo
    19  XLOGO=/usr/X11R6/bin/xlogo
    20  PGREP=/usr/bin/pgrep
    21  KILL=/bin/kill
    22  HEAD=/bin/head
    23
    24  # get screen width and height
    25  width=‘$XDPYINFO | awk ’/dimensions/ {print $2}’ | awk -F x ’{print $1}’‘
    26  height=‘$XDPYINFO | awk ’/dimensions/ {print $2}’ | awk -F x ’{print $2}’‘
    27
    28  # put color names in array ’colors’
    29  colors=(‘$SHOWRGB | sed -r ’s/[[:digit:][:blank:]]+//’ | grep -v ’ ’‘)
    30  NC=${#colors[*]} # size of array
    31
    32
    33  while [ 1 ]
    34    do
    35    fg=${colors[$(($RANDOM % $NC))]}
    36    bg=${colors[$(($RANDOM % $NC))]}
    37    x=$(($RANDOM % $width))
    38    y=$(($RANDOM % $height))
    39    echo Foreground color: $fg
    40    echo Background color: $bg
    41    echo x position: $x
    42    echo y position: $y
    43    $XLOGO -fg $fg  -bg $bg -geometry +$x+$y &
    44    echo
    45    if ((‘$PGREP ’xlogo$’ | wc -l‘ > $N ))
    46        then
    47        # kill first xlogo process in the list
    48        $KILL ‘pgrep ’xlogo$’ | $HEAD -n 1‘
    49    fi
```

```
50    sleep 1
51  done
```

บรรทัดที่ 2 ถึง 8 เป็นหมายเหตุเขียนอธิบายการทำงานเกี่ยวกับเชลล์สคริปต์ว่าทำอะไรบ้าง. บรรทัดที่ 10 ถึง 15 ตรวจสอบดูว่ามีอาร์กิวเมนต์เวลารันเชลล์สคริปต์นี้หรือไม่. ถ้าไม่มีอาร์กิวเมนต์จะตั้งค่า N ซึ่งคือจำนวนหน้าต่าง xlogo ไว้ที่ 5 บาน. ถ้ามีอาร์กิวเมนต์จะสมมติว่าเป็นตัวเลขแล้วใช้ตัวเลขนั้นตั้งค่าจำนวนหน้าต่าง xlogo ที่จะแสดง.

บรรทัดที่ 17 ถึง 21 เป็นการบันทึกโปรแกรมหรือคำสั่งต่าง ๆเก็บไว้ในตัวแปรให้เป็นระเบียบเรียบร้อย. วิธีแบบนี้มีประโยชน์ตอนแก้ไขสคริปต์, คือถ้าในระบบมีโปรแกรมอยู่ในไดเรกทอรีที่ไม่เหมือนกันก็แก้ไขได้ในช่วงนี้. ไม่ต้องไปหาในสคริปต์ว่าเขียนไว้อยู่ตรงไหน.

บรรทัดที่ 25, 26 ใช้คำสั่ง xdpyinfo ร่วมกับคำสั่ง awk ตั้งค่าตัวแปร width และ height เก็บความกว้างและความสูงของหน้าจอเป็นหน่วยพิกเซล. คำสั่ง xdpyinfo จะแสดงข้อมูลต่าง ๆของ X เซิฟร์เวอร์และค่าต่าง ๆของหน้าจอ. เรารู้ว่าผลลัพธ์ของคำสั่ง xdpyinfo จะมีบรรทัดที่บอกข้อมูลเกี่ยวกับความกว้างและสูงของหน้าจอได้แก่

```
dimensions:    1280x1024 pixels (382x313 millimeters)
```

เพื่อที่จะสกัดเอาค่าความกว้าง (1280) และความสูง (1024) จากบรรทัดนี้, จึงใช้คำสั่ง awk สกัดคอลัมน์ที่ 2 ซึ่งได้แก่ 1280x1024 ออกมาด้วยคำสั่ง

```
awk '/dimensions/ {print $2}'
```

เมื่อได้สายอักขระ 1280x1024 มาแล้วต้องแยกให้ออกจากกันโดยใช้คำสั่ง awk อีกรอบโดยให้ตัวแบ่งคอลัมน์เป็น x และสกัดคอลัมน์ที่ 1 เก็บไว้ในตัวแปร width และสกัดคอลัมน์ที่ 2 เก็บไว้ในตัวแปร height.

```
awk -F x '{print ...}'
```

บรรทัดที่ 29 เป็นการสร้างตัวแปรแถวลำดับชื่อ colors ไว้เก็บชื่อสีต่าง ๆจากผลลัพธ์ของคำสั่ง showrgb. ผลลัพธ์ของคำสั่ง showrgb จะแสดงค่า RGB และชื่อของสีในตัวอย่างที่ 4.96. ในเชลล์สคริปต์นี้จะสกัดเอาส่วนที่เป็นชื่อสีโดยใช้คำสั่ง sed.

```
sed -r 's/[[:digit:][:blank:]]+//'
```

คำสั่งนี้ใช้จะลบส่วนที่เป็นตัวเลขหรือช่องว่าออกจากบรรทัดทุกบรรทัด. ผลของคำสั่งนี้จะได้ชื่อสีบรรทัดต่อบรรทัด. แต่เราทราบแล้วว่ามีกรณีชื่อสีซ้ำอยู่ด้วยเช่น “ghost white” จะเหมือนกับ GhostWhite. ในกรณีนี้จะใช้ grep คัดเอาคำที่มีช่องไฟออกด้วยคำสั่ง grep -v ' '.

บรรทัดที่ 30 เป็นการหาจำนวนสมาชิกที่อยู่ในตัวแปรแถวลำดับและเก็บไว้ในตัวแปร NC. บรรทัดที่ 33 เริ่มต้นวงวนไม่รู้จบ. ในแต่ละรอบของวงวนจะตั้งค่าตัวแปร fg และ

bg โดยใช้ชื่อสีที่บันทึกอยู่ในตัวแปรแถวลำดับ colors. การสุ่มดรรชนีเพื่อระบุชื่อสีจะใช้วิธีหาเศษการหารค่าตัวแปรพิเศษ $RANDOM กับ NC ทำให้ดรรชนีที่ได้อยู่ในช่วงจำนวนของสีทั้งหมดที่มีอยู่เสมอ (บรรทัดที่ 35, 36). ในทำนองเดียวกัน, บรรทัดที่ 37, 38 ตั้งค่าพิกัด x, y โดยสุ่มให้อยู่ในกรอบของหน้าจอ.

บรรทัดที่ 39 ถึง 42 แสดงค่าของตัวแปร fg, bg, x และ y ทางเทอร์มินอล. หลังจากนั้นจะรันโปรแกรม xlogo ด้วยตัวเลือกและค่าที่ตั้งไว้ในบรรทัดที่ 43. และบรรทัดที่ 44 แสดงบรรทัดว่างเปล่าหนึ่งบรรทัดเพื่อให้อ่านข้อมูลที่แสดงทางเทอร์มินอลได้สะดวกขึ้น.

บรรทัดที่ 45 จะใช้ if ตรวจสอบดูว่ามีจำนวนโปรเซสที่ชื่อ xlogo เท่าไร, แล้วเทียบกับค่า N ว่ามีมากกว่าจำนวนที่ตั้งไว้หรือไม่. อาร์กิวเมนต์ของคำสั่ง pgrep คือ xlogo$ ระบุด้วย regular expression. หมายถึงโปรเซสที่ชื่อ xlogo, จะไม่รวมถึง xlogo.sh เพราะมีเครื่องหมาย $ แสดงว่าไม่มีคำใด ๆต่อถัดจาก xlogo. แล้วใช้คำสั่ง wc -l เพื่อนับจำนวนโปรเซส xlogo. ถ้าจำนวนโปรเซสมากกว่าค่า N ก็จะสั่งคำสั่ง

```
$KILL ‘pgrep ’xlogo$’ | $HEAD -n 1‘
```

อาร์กิวเมนต์ของคำสั่ง kill เป็นหมายเลขโปรเซสที่ได้จากคำสั่ง pgrep ’xlogo$’ | $HEAD -n 1 ซึ่งคือหมายเลขโปรเซสตัวแรกของคำสั่ง pgrep อีกที.

บรรทัดที่ 50 จะหยุดการทำงาน 1 วินาทีเพื่อไม่ให้การวนรอบเร็วเกินไป, หลังจากนั้นก็เป็นการซ้ำวงวน while ไปเรื่อย ๆในบรรทัดสุดท้าย. ถ้าต้องการหยุดการทำงานของเชลล์สคริปต์นี้ให้กด C-c.

สำหรับผู้ที่สนใจการเขียนเชลล์สคริปต์อย่างจริงจัง, ควรจะอ่านหนังสือที่เกี่ยวกับการใช้เชลล์และเขียนเชลล์สคริปต์โดยเฉพาะ. หรืออ่านเอกสารจาก The Linux Document Project เช่น Bash Guide for Beginners [37] และ Advanced Bash-Scripting Guide [38].

4.8 สรุปท้ายบท

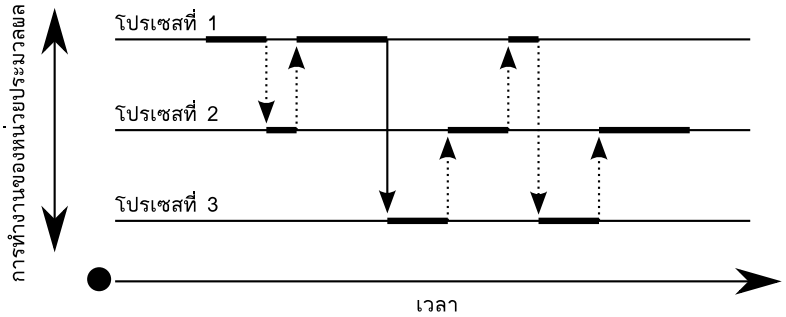
- คำสั่งพื้นฐานในลินุกซ์มักจะหมายถึงคำสั่งที่สืบทอดมาจากระบบปฏิบัติการยูนิกซ์ เช่น คำสั่งสำหรับจัดการระบบไฟล์, คำสั่งประมวลผลข้อมูลเท็กซ์ และคำสั่งที่ใช้ประกอบกับเชลล์ เป็นต้น.
- ในกรณีทั่วไป, อาร์กิวเมนต์ของคำสั่งมักจะเป็นชื่อไฟล์และรับข้อมูลเข้าจากไฟล์นั้นและแสดงผลทาง stdout. ถ้าไม่มีการระบุชื่อไฟล์ก็มักจะรับข้อมูลจาก stdin โดยปริยาย.
- ให้ใช้ไปป์และรีไดเรกชันช่วยสำหรับการใช้คำสั่งหลาย ๆคำสั่งด้วยกัน.
- สร้างและใช้เชลล์สคริปต์เมื่อต้องการทำงานตามขั้นตอนที่กำหนดไว้แล้วโดยอัตโนมัติ. หรือเมื่อคำสั่งที่มีอยู่ในระบบไม่สามารถแก้ไขปัญหางานที่ต้องการกระทำ.

บทที่ 5
โปรเซส

ในบทที่ผ่านมาเราได้เรียนรู้การใช้โปรแกรมคำสั่งพื้นฐานต่าง ๆที่ใช้ในลินุกซ์ไปแล้ว. ในบทนี้จะแนะนำคำสั่งที่เกี่ยวกับระบบปฏิบัติการโดยซึ่งได้แก่ลินุกซ์เคอร์เนล. เนื้อหาของบทนี้ส่วนหนึ่งจะเกี่ยวข้องกับการทำงานของเคอร์เนลโดยใช้มองผ่านการใช้งานคำสั่งที่เกี่ยวข้องเช่น `ps`, `kill` เป็นต้น, ช่วยให้ผู้อ่านเข้าใจการทำงานของระบบปฏิบัติการซึ่งได้แก่เคอร์เนลได้ดียิ่งขึ้น.

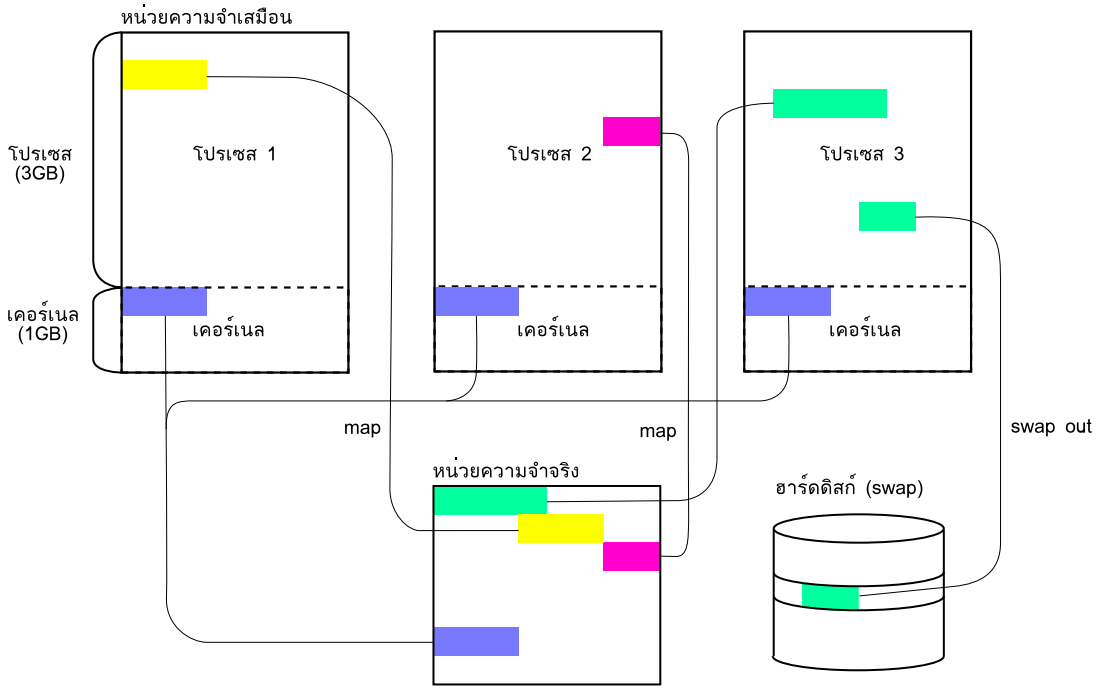
5.1 โปรเซส (process)

นิยามอย่างง่ายของโปรแกรมคือไฟล์ที่มีเนื้อหาเป็นภาษาเครื่องกลและหน่วยประมวลผลสามารถแปลความกระทำการได้. ส่วนนิยามอย่างง่ายของโปรเซสคือโปรแกรมที่กำลังทำงานอยู่. เวลารันโปรแกรม, เคอร์เนลจะอ่านข้อมูลที่อยู่ในไฟล์โปรแกรมเข้าไปไว้ในหน่วยความจำ, และหน่วยประมวลผลจะกระทำการตามคำสั่งต่อไป. สภาพที่ระบบปฏิบัติการโหลดภาษาเครื่องกล (โปรแกรม) นั้นสู่หน่วยความจำแล้วกระทำการอยู่เรียกว่า*โปรเซส (process)*. ในมุมมองของเคอร์เนลมักจะเรียกโปรเซสว่า*ทาส์ก (task)* ซึ่งมีความหมายเหมือนกัน.



รูปที่ 5.1: การสลับการทำงานของหน่วยประมวลผล.

ลินุกซ์เป็นระบบปฏิบัติการแบบระบบมัลติทาส์ก, ในระบบมีโปรเซสหลายโปรเซสทำงานหลายอย่างพร้อม ๆกัน. ในความเป็นจริงแล้วหน่วยประมวลผลซึ่งเป็นตัวกระทำการ



รูปที่ 5.2: ความสัมพันธ์ระหว่างโปรเซสและหน่วยความจำ.

ต่าง ๆสามารถทำงานได้ทีละหนึ่งอย่าง, หนึ่งคำสั่ง. แต่เนื่องจากการทำงานของหน่วยประมวลผลสั้นมากทำให้ผู้ใช้รู้สึกเหมือนกับโปรเซสหลาย ๆตัวทำงานได้พร้อม ๆกัน. การที่หน่วยประมวลเปลี่ยนการทำงานจากโปรเซสหนึ่งไปอีกโปรเซสหนึ่งเรียกว่า *context switching* คือเนื้อหาการทำงาน, ข้อมูลที่ต้องจัดการเปลี่ยนไป. หน่วยประมวลเป็นตัวที่คำนวณข้อมูล, เรียกอ่านข้อมูลจากหน่วยความจำ, จากฮาร์ดิกส์หรือฮาร์ดแวร์. การเรียกอ่านเขียนข้อมูลโดยเฉพาะฮาร์ดดิสก์นั้นจะใช้เวลามากกว่าการประมวลผล. ดังนั้นโปรเซสที่ต้องเขียนอ่านข้อมูลจะถูก*บล็อก (block)* และเปลี่ยนไปอยู่ในสภาพรอ (wait). กล่าวคือแทนที่หน่วยประมวลจะรอให้ข้อมูลที่ต้องการใช้มาถึงก็จะหยุดการทำงานของโปรเซสนั้นชั่วคราวไปใช้เวลาทำงานกับโปรเซสอื่นเพื่อไม่ให้เสียเวลา. พอได้รับ *interrupt* จากฮาร์ดแวร์นั้น ๆเช่นเรียกอ่านเขียนข้อมูลเสร็จแล้วก็จะจัดการกับโปรเซสนั้นอีกที.

address space ▶ หน่วยความจำเสมือน (virtual memory) ที่โปรเซสสามารถใช้ได้. เคอร์เนลจะแม็ป (map) address space เข้ากับหน่วยความจำจริง (physical memory) เวลาใช้งาน.

code ▶ ส่วนที่เป็นข้อมูลสำหรับหน่วยประมวลผลสั่งคำสั่ง.

data ▶ ส่วนที่เป็นข้อมูลตัวแปรส่วนกลาง (global variable) ของโปรเซส.

stack ▶ โครงสร้างข้อมูล (data structure) แบบเข้าก่อนออกหลัง (first-in last-out) สำหรับเก็บตัวแปรเฉพาะที่ (local variable) ในฟังก์ชัน ฯลฯ. พื้นที่สำหรับ stack จะอยู่ท้าย ๆของหน่วยความจำ.

ตามทฤษฎีแล้ว, โปรเซสแต่ละตัวจะมี *address space* เป็นของตัวเอง. Address space คือพื้นที่หน่วยความจำเสมือน (virtual memory) เป็นที่เก็บข้อมูลของโปรเซสสำคัญ ๆ เช่น *code*, *data*, *stack*. หน่วยความจำเสมือนสำหรับหน่วยประมวลผลสถาปัตยกรรมแบบ 32 บิตจะมีพื้นที่ 4GB (2^{32} ไบต์) และพื้นที่ 1GB ท้ายของหน่วยความจำจะใช้โดยเคอร์เนล. พื้นที่ที่ใช้โดยเคอร์เนลจะเป็นส่วนเดียวกันสำหรับทุกโปรเซส. เคอร์เนลเป็นตัวจัดการหน่วยความจำโดยแม็ป (map) หน่วยความจำเสมือนนี้เข้ากับหน่วยความจำจริง (physical memory) ซึ่งไม่จำเป็นต้องแม็ปหน่วยความจำทั้งหมดที่โปรเซสใช้เข้ากับหน่วยความจำจริง. ส่วนของหน่วยความจำที่ไม่จำเป็นต้องอยู่ในหน่วยความจำจริงจะถูกเคอร์เนล swap out ไปเก็บไว้ที่ฮาร์ดดิสก์ส่วนที่เป็น swap. เมื่อโปรเซสต้องพยายามเข้าถึง address ที่ข้อมูลไม่อยู่ในหน่วยความจำจริง, จะเกิด *page fault* คือมีข้อมูลใน address

ที่ระบุแต่ข้อมูลนั้นไม่ได้อยู่ในหน่วยความจำจริง, เคอร์เนลต้องไปดึงข้อมูลที่อยู่ใน swap กลับมาในหน่วยความจำจริง.

เวลาที่โปรเซสหนึ่งทำงาน, หน่วยประมวลผลจะทำงานและใช้เวลาในการแปลคำสั่งของโปรเซสนั้น ๆ. เวลาหน่วยประมวลผลทำงานจะมีสภาพแบบใดแบบหนึ่งได้แก่ *user mode* หรือ *kernel mode*. เวลาที่หน่วยประมวลทำงานอยู่ใน user mode, การเข้าถึงหน่วยความจำจะถูกจำกัดไว้ในส่วนที่อนุญาตให้เข้าถึงได้เท่านั้นและไม่สามารถใช้ฮาร์ดแวร์ต่าง ๆ ได้โดยตรง. การทำงานของโปรเซสโดยทั่วไปจะอยู่ใน user mode. เมื่อโปรเซสนั้นเรียกใช้ซิสเต็มคอล (system call), หน่วยประมวลจะเปลี่ยนสภาพการทำงานไปอยู่ใน kernel mode. ใน kernel mode, หน่วยประมวลผลสามารถเข้าถึงหน่วยความจำได้ทุกส่วนและติดต่อกับฮาร์ดแวร์ได้. เมื่อจบการทำงานใน kernel mode แล้วก็จะกลับไปเป็น user mode ใหม่. เป็นเช่นนี้ไปเรื่อย ๆ.

โปรเซส

5.2 สำรวจโปรเซส

คำสั่งที่ใช้แสดงรายการโปรเซสที่มีอยู่ระบบได้แก่ ps. คำสั่งจะแสดงรายการโปรเซส (ชื่อโปรแกรม) แบบ foreground และ background ที่เกิดจากเทอร์มินอลนั้นโดยปริยายถ้าไม่ระบุอาร์กิวเมนต์.

ps อ้างอิงหน้า 403

ตัวอย่างที่ 5.1: ผลลัพธ์ของคำสั่ง ps โดยไม่มีอาร์กิวเมนต์.

```
$ ps↵
  PID TTY          TIME CMD
10523 pts/1    00:00:00 bash
11556 pts/1    00:00:00 firefox
11595 pts/1    00:00:00 run-mozilla.sh
11600 pts/1    00:00:05 firefox-bin
11690 pts/1    00:00:00 ps
```

ข้อมูลที่คำสั่ง ps แสดงได้แก่

- PID (Process ID)
  โปรเซสทุกโปรเซสจะมีเลขประจำตัวเฉพาะที่เรียกว่า*โปรเซส ID (process ID)* เพื่อใช้อ้างอิง. หมายเลขโปรเซสจะเริ่มต้นด้วย 0 จนถึง 32767 [39]. ถ้ามีโปรเซสใหม่เกิดขึ้นก็จะใช้โปรเซส ID ที่มีค่าถัดจากโปรเซส ID ตัวสุดท้ายในระบบไปเรื่อย ๆ. ถ้าโปรเซส ID เกิน 32767 ก็จะใช้โปรเซส ID เริ่มตั้งแต่ 0 ใหม่โดยจะใช้โปรเซส ID ที่ยังไม่ซ้ำกับโปรเซสที่กำลังทำงานอยู่.

  ค่าโปรเซส ID สูงสุดสามารถดูได้จากไฟล์ /proc/sys/kernel/pid_max

- TTY (Teletype)
  โปรเซสแต่ละโปรเซสจะมีเทอร์มินอลควบคุมโปรเซสซึ่งโดยปรกติคือเทอร์มินอลที่รันโปรเซสนั้น. โปรเซสบางตัวที่ไม่ได้เกิดจากการสั่งคำสั่งทางเทอร์มินอลเช่นโปรเซสของเซิฟร์เวอร์ต่าง ๆ ในคอลัมน์ที่แสดงเทอร์มินอลควบคุมจะเป็นเครื่องหมาย ?.

- TIME
  ระยะเวลาที่หน่วยประมวลผลใช้ทำงานโดยโปรเซส. ค่า TIME ไม่ใช่ระยะเวลาที่รันโปรเซสนั้นจนถึงปัจจุบันแต่เป็นระยะเวลาจริงที่หน่วยประมวลผลใช้ไปกับโปรเซสนั้น. ระยะเวลานี้สั้นมากดังนั้นค่า TIME ของบางโปรเซสมีค่าประมาณเป็นศูนย์.

- CMD
  ชื่อโปรเซสแบบสั้นซึ่งได้แก่ชื่อโปรแกรมที่กระทำการ.

ถ้าต้องการแสดงโปรเซสทั้งหมดที่มีอยู่ในระบบให้ใช้ตัวเลือก `-e` (everything) หรือ `-A` (All).

ตัวอย่างที่ 5.2: แสดงโปรเซสทั้งหมดในระบบ.

```
$ ps -e↵                                                    ← หรือ ps -A
  PID TTY          TIME CMD
    1 ?        00:00:04 init
    2 ?        00:00:00 ksoftirqd/0
    3 ?        00:00:00 events/0
    4 ?        00:00:00 kblockd/0
--- แสดงผลต่อไปเรื่อยๆ ---
```

5.2.1 โปรเซสและเจ้าของ

ในระบบมัลติยูสเซอร์, โปรเซสทุกตัวจะมีเจ้าของซึ่งโดยปรกติแล้วจะเป็นผู้ที่สร้างโปรเซสนั้น. การควบคุมโปรเซสเช่นสั่งจบการทำงานจะสามารถทำได้กับโปรเซสที่เป็นเจ้าของเท่านั้น. แต่สำหรับผู้ใช้ root มีสิทธิ์พิเศษสามารถควบคุมโปรเซสได้ทุกตัวในระบบ. การควบคุมโปรเซสทำได้โดยการส่งสัญญาณ (signal) ให้โปรเซสรับรู้ซึ่งจะแนะนำในช่วงถัดไป (หน้า 216).

เราสามารถใช้ตัวเลือก `-f` เพื่อแสดงเจ้าของโปรเซสและข้อมูลเพิ่มเติมเกี่ยวกับโปรเซสได้ด้วย. ถ้าใช้ตัวเลือกนี้กับตัวเลือก `-e` ก็จะแสดงโปรเซสทั้งหมดในระบบและรายละเอียดของโปรเซส.

ตัวอย่างที่ 5.3: แสดงรายละเอียดของโปรเซสด้วยตัวเลือก `-f`

```
$ ps -ef↵
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 09:01 ?        00:00:04 init [3]
root         2     1  0 09:01 ?        00:00:00 [ksoftirqd/0]
root         3     1  0 09:01 ?        00:00:00 [events/0]
root         4     3  0 09:01 ?        00:00:00 [kblockd/0]
--- แสดงผลต่อไปเรื่อยๆ ---
```

รายเอียดของโปรเซสเพิ่มเติมจากค่าปริยายที่แสดงได้แก่

- UID (User ID)
  แสดงเจ้าของโปรเซสโดยใช้ชื่อล็อกอิน.

- PPID (Parent Process ID)

โปรเซสพ่อแม่, แสดงโปรเซส ID ที่เป็นตัวสร้างโปรเซสที่แสดงอยู่.

- C
  ข้อมูลดิบของระยะเวลาที่หน่วยประมวลผลใช้ไปกับโปรเซสในหน่วย clock tick.

- STIME (Start TIME)
  เวลาที่โปรเซสเริ่มทำงาน.

สำหรับผู้ดูแลระบบที่ต้องดูว่ายูสเซอร์รันโปรเซสอะไรอยู่, สามารถใช้ตัวเลือก `-u` *`user`* ดูโปรเซสโดยระบุยูสเซอร์ที่ต้องการได้.

ตัวอย่างที่ 5.4: ใช้คำสั่ง `ps` เลือกดูโปรเซสของผู้ใช้ที่ต้องการ.

```
$ ps -fu poonlap↵
  PID TTY          TIME CMD
 7099 ?        00:00:00 gam_server
10197 ?        00:00:00 esd
10291 ?        00:00:01 gnome-session
--- แสดงผลต่อไปเรื่อยๆ ---
10523 pts/1    00:00:00 bash
10642 pts/2    00:00:00 bash
10679 pts/2    00:00:01 ssh
11556 pts/1    00:00:00 firefox
11595 pts/1    00:00:00 run-mozilla.sh
11600 pts/1    00:00:17 firefox-bin
13715 pts/1    00:00:00 ps
```

อาร์กิวเมนต์ของตัวเลือก `-u` จะเป็นชื่อล็อกอินหรือ UID ก็ได้.

ในกรณีที่ยูสเซอร์ใช้เทอร์มินอลอยู่หลายตัวและเราต้องการเลือกดูโปรเซสที่อยู่ในเทอร์มินอลทีทต้องการ, ให้ใช้ตัวเลือก `-t` *`tty`*.

ตัวอย่างที่ 5.5: เลือกแสดงโปรเซสตามเทอร์มินอลที่ต้องการ.

```
$ ps -ft pts/2↵
UID        PID  PPID  C STIME TTY          TIME CMD
poonlap  10642 10499  0 10:26 pts/2    00:00:00 bash
poonlap  10679 10642  0 10:26 pts/2    00:00:01 ssh -X poonlap@10.0.0.1
```

จากตัวอย่างเป็นการแสดงโปรเซสที่ควบคุมโดยเทอร์มินอล pts/2 และใช้ตัวเลือก `-f` แสดงรายละเอียดประกอบ. ลำดับของตัวเลือกมีความสำคัญ, ไม่สามารถสลับกันได้เพราะ pts/2 เป็นอาร์กิวเมนต์ของ `-t` ไม่ใช่ `-f`.

5.2.2 ความสัมพันธ์ระหว่างโปรเซส

การสร้างโปรเซสคือการรันโปรแกรมซึ่งสามารถทำได้โดยสั่งคำสั่งในเชลล์พรอมต์, หรือเลือกโปรแกรมที่ต้องการใช้จากเมนูเป็นต้น. ในกรณีที่ใช้เชลล์สั่งคำสั่ง, โปรเซสใหม่จะสร้างโดยเชลล์และจะเกิดความสัมพันธ์ระหว่างโปรเซสที่เป็นผู้สร้างและโปรเซสที่ถูกสร้าง. เราจะเรียกโปรเซสที่เป็นผู้สร้างว่า*โปรเซสพ่อแม่ (parent process)* และโปรเซสที่ถูกสร้าง

ใหม่ว่า*โปรเซสลูก (child process)*. โปรเซสที่เกิดใหม่จะมีโปรเซสพ่อแม่เสมอซึ่งจะดูได้จากคอลัมน์ PPID ของคำสั่ง ps.

คำสั่ง pstree ช่วยแสดงโปรเซสโดยใช้แผนภาพต้นไม้ทำให้ดูความสัมพันธ์ระหว่างโปรเซสง่ายขึ้น. เราลองมาดูความสัมพันธ์ของโปรเซส bash (PID 10523) ที่แสดงในตัวอย่าง 5.4 ด้วยคำสั่ง pstree ดังต่อไปนี้.

ตัวอย่างที่ 5.6: แผนภาพต้นไม้ของโปรเซส.

```
$ pstree -pu 10523↵
bash(10523,poonlap)-+-firefox(11556)---run-mozilla.sh(11595)---firefox-bin(1160+
                    `-pstree(16346)
```

ตัวเลือก -p และ -u ใช้สำหรับแสดงโปรเซส ID และชื่อยูสเซอร์ในวงเล็บตามลำดับ. จากคำสั่ง pstree ทำให้เราเห็นภาพชัดเจนขึ้นว่า bash เป็นโปรเซสพ่อแม่ของ firefox. ต่อจากนั้น firefox สร้างโปรเซสตัวใหม่ชื่อ run-mozilla.sh, และสุดท้ายโปรเซส run-mozilla.sh สร้างโปรเซส firefox-bin ซึ่งเป็นไบนารีไฟล์จริง ๆของโปรแกรมเบราเซอร์ Firefox.

อาร์กิวเมนต์ของคำสั่ง pstree สามารถเป็นได้ทั้งโปรเซส ID หรือชื่อยูสเซอร์. ในกรณีที่ไม่ระบุโปรเซส ID หรือชื่อยูสเซอร์, จะแสดงแผนภาพโปรเซสของระบบ.

ตัวอย่างที่ 5.7: ความสัมพันธ์ระหว่างโปรเซสต่าง ๆในระบบ.

```
$ ps -cpun↵
init(1)-+-ksoftirqd/0(2)
        |-events/0(3)-+-khelper(4)
        |              |-kacpid(5)
        |              |-kblockd/0(24)
        |              |-pdflush(34)
        |              |-pdflush(35)
        |              `-aio/0(37)
        |-khubd(25)
--- แสดงผลต่อไปเรื่อยๆ ---
        |-mapping-daemon(10483,poonlap)
        |-gnome-terminal(10499,poonlap)-+-gnome-pty-helpe(10522)
        |                               |-bash(10523)-+-firefox(11556)---run-mo+
        |                               |             `-pstree(17205)
        |                               `-bash(10642)---ssh(10679)
        |-wnck-applet(10503,poonlap)
        |-multiload-apple(10507,poonlap)
        |-clock-applet(10509,poonlap)
        |-mixer_applet2(10512,poonlap)
        |-gnome-keyboard-(10515,poonlap)
        `-notification-ar(10518,poonlap)
```

ตัวอย่างข้างบนเป็นการแสดงผลของคำสั่ง pstree ตามลำดับการสร้างโปรเซสต่าง ๆในระบบ (ตัวเลือก -n), แสดงโปรเซส ID (ตัวเลือก -p), และแสดงชื่อล็อกอินที่เป็นเจ้าของโปรเซส (ตัวเลือก -u).

ในทางเทคนิค, การสร้างโปรเซสใหม่จะเรียกว่าการ fork. ในระบบปฏิบัติการลินุกซ์, โปรเซสตัวแรกที่เป็นโปรเซสเริ่มต้นของโปรเซสทั้งหมดคือ init. เนื่องจากเป็นโปรเซสตัว

fork เป็นคำกริยาแปลว่าการแยกออกเป็นกิ่งก้านสาขา.

แรก, จึงมีโปรเซส ID เป็นเลข 1 เสมอ. ถัดจากโปรเซส init จะเป็นโปรเซสที่เกิดจากเคอร์เนลเช่น ksoftirqd, eventd, khelper ฯลฯ. โปรเซสเหล่านี้เรียกว่า *kernel thread* เป็นโปรเซสที่ช่วยการทำงานของเคอร์เนล. ถ้าใช้คำสั่ง `ps -ef` จะแสดง kernel thread ในวงเล็บเหลี่ยม.

จากตัวอย่างจะเห็นว่า firefox เป็นโปรเซสลูกของโปรเซส bash. จะเกิดอะไรขึ้นถ้าโปรเซส bash ตายไป? ถ้าโปรเซส firefox เป็นโปรเซสแบบ foreground, โปรเซส firefox จะตายตามไปด้วย. แต่ถ้า firefox เป็นโปรเซส background, โปรเซส firefox จะกลายเป็น*โปรเซสกำพร้า (orphan process)* ไม่มีโปรเซสพ่อแม่. ในกรณีนี้โปรเซสพ่อของ firefox จะกลายเป็นโปรเซส init โดยปริยาย. จะสังเกตได้ว่าโปรเซสหลายตัวมีโปรเซสพ่อแม่เป็นโปรเซส init เช่น wnck-applet, multiload-apple ฯลฯ. โปรเซสเหล่านี้จะไม่มีเทอร์มินอลควบคุม.

โปรเซสอีกประเภทหนึ่งซึ่งไม่พบบ่อยนักได้แก่โปรเซส zombie. โปรเซส zombie คือโปรเซสที่ตายไปแต่โปรเซสพ่อแม่ไม่ได้รับรู้, ทำให้ชื่อโปรเซสยังอยู่ในตารางโปรเซสของเคอร์เนล. คำสั่ง `ps` จะแสดงชื่อโปรเซส zombie ตามด้วยคำว่า <defunct>. โปรเซส zombie เป็นโปรเซสที่ตายไปแล้วไม่ได้ใช้ทรัพยากรใด ๆในระบบ, ไม่มีอันตรายใด ๆ.

ตัวอย่างที่ 5.8: โปรเซส zombie.

```
$ ps -ly 21634↵
S   UID   PID  PPID  C PRI  NI  RSS    SZ WCHAN  TTY        TIME CMD
Z   500 21634 11600  0  77   0    0     0 exit   pts/1      0:00 [netstat] <defu
nct>
```

5.2.3 เกี่ยวกับคำสั่ง ps

คำสั่ง `ps` เป็นคำสั่งที่ใช้มานานตั้งแต่สมัยระบบปฏิบัติการยูนิกซ์. ตัวเลือกต่าง ๆของคำสั่งดั้งเดิมจะแตกต่างกันตามระบบปฏิบัติการยูนิกซ์ที่ใช้เช่น SysV หรือ BSD. สำหรับคนที่เคยใช้ยูนิกซ์มาก่อนอาจจะรู้มาว่าถ้าต้องการแสดงโปรเซสในระบบทั้งหมดให้ใช้คำสั่ง `ps aux`. การใช้คำสั่ง `ps` และตัวเลือกโดยไม่ใช้เครื่องหมาย `-` นำหน้าตัวเลือกเป็นการใช้คำสั่ง `ps` สไตล์ BSD. เพื่อความเป็นมาตรฐานและไม่สับสนควรใช้คำสั่ง `ps` แบบมาตรฐานตาม The Single UNIX Specification Version 3 (SusV3) [40] ซึ่งจะใช้เครื่องหมาย `-` นำหน้าตัวเลือกเสมอ. ในหนังสือเล่มนี้จะยึดตามหลัก SusV3. สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับการใช้คำสั่ง `ps` สไตล์ BSD สามารถอ่านได้จาก `man ps`.

รายละเอียดของตัวเลือก

คำสั่ง `ps` มีความยืดหยุ่นสูงสามารถแสดงข้อมูลต่าง ๆของโปรเซสที่เราต้องการได้. ตัวอย่างเช่นตัวเลือก `-f` (full-format) ใช้แสดงรายละเอียดของโปรเซสได้ในระดับที่พอควร. นอกจากนั้นยังมีตัวเลือก `-F` (extra full-format) ซึ่งจะแสดงรายละเอียดของโปรเซสให้ละเอียดขึ้นอีก, และตัวเลือก `-l` (long format) แสดงรายเอียดแบบยาว.

ตัวอย่างที่ 5.9: เปรียบเทียบตัวเลือกแสดงรายละเอียดของโปรเซส.

```
$ ps -f↵
UID        PID  PPID  C STIME TTY          TIME CMD
poonlap  18519  7868  0 12:07 pts/7    00:00:00 -bash
poonlap  22212 18519  0 14:00 pts/7    00:00:00 ps -f
$ ps -F↵
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
poonlap  18519  7868  0   561  1292   0 12:07 pts/7    00:00:00 -bash
poonlap  22215 18519  0   605   828   0 14:00 pts/7    00:00:00 ps -F
$ ps -l↵
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 18519  7868  0  75   0 -   561 wait4  pts/7    00:00:00 bash
0 R  1000 22218 18519  0  77   0 -   559 -      pts/7    00:00:00 ps
$ ps -ly↵
S   UID   PID  PPID  C PRI  NI   RSS    SZ WCHAN  TTY          TIME CMD
S  1000 18519  7868  0  75   0  1292   561 wait4  pts/7    00:00:00 bash
R  1000 22232 18519  0  77   0   668   559 -      pts/7    00:00:00 ps
```

รายละเอียดต่างของแต่คอลัมน์ได้แก่

- SZ
  ขนาดของหน่วยความจำที่โปรเซสใช้มีหน่วยเป็น *page* หน่วยความจำนี้คือเนื้อที่ที่ใช้สำหรับข้อมูล text, data และ stack ของโปรเซส. อีกความหมายหนึ่งคือขนาดของ*หน่วยความจำเสมือน (virtual memory)* ของโปรเซส.

- RSS (Resident Set Size)
  จำนวนหน่วยความจำที่โปรเซสใช้อยู่จริง. มีหน่วยเป็น kB. สมมติว่าโปรเซสหนึ่งต้องการใช้หน่วยความจำทั้งหมด x กิโลไบต์, โดยปรกติแล้วเคอร์เนลจะจัดการการใช้หน่วยความจำของระบบและจะจัด*หน่วยความจำจริง (physical memory)* ให้ซึ่งไม่จำเป็นต้องเท่ากับหน่วยความจำที่โปรเซสต้องการทั้งหมด.

- PSR (Processor)
  หน่วยประมวลที่ทำงานโปรเซสนั้น. มีความหมายสำหรับเครื่องคอมพิวเตอร์ที่มีหน่วยประมวลผลหลายตัว.

- F และ ADDR
  เป็นข้อมูลที่ไม่ค่อยใช้กันและมักจะใช้ตัวเลือก `-y` กับ `-l` เพื่อไม่แสดง flag และเปลี่ยน `ADDR` ให้เป็น `RSS` แทน.

- S (Status)
  สภาพของโปรเซสได้แก่

  — D Uninterruptible sleep

  — R Running หรือ runnable คือโปรเซสที่อยู่ใน*รันคิว (run queue)* หน่วยประมวลผลกำลังทำงานโปรเซสนั้นอยู่, หรือรอหน่วยประมวลอยู่.

  — S Interruptible sleep, กำลังรอเหตุการณ์ใดเหตุการณ์หนึ่งให้เสร็จ

page ▶
หน่วยของหน่วยความจำ. ในระบบยูนิกซ์รวมถึงลินุกซ์จะแบ่งหน่วยความจำเป็นกลุ่มขนาดเล็กเรียกว่า page. โดยทั่วไป page จะมีขนาด 4Kb (4096 ไบต์).

virtual memory ▶
หน่วยความจำเสมือน. พื้นที่หน่วยความจำ (address space) ที่โปรเซสมองเห็นซึ่งโดยทั่วไปจะมีขนาดไม่เท่ากับหน่วยความจำจริง (physical memory). ในระบบสถาปัตยกรรม 32 บิตพื้นที่ที่โปรเซสเห็นจะมีขนาด 4GB.

physical memory ▶
หน่วยความจำจริง. คือจำนวนหน่วยความจำที่มีจริงในระบบหมายถึงตัวฮาร์ดแวร์ (memory). เคอร์เนลจะใช้หน่วยความจำจริงโดยแบ่งเป็น page, มักจะเรียกว่า freame.

ข้อมูลของ flag ของโปรเซสอยู่ในไฟล์ `/usr/include/linux/sched.h`

  - T Stopped หมายถึงหยุดทำงานชั่วคราวไม่ใช่จบการทำงาน.
  - W paging (ยกเลิกหลังจากเคอร์เนล 2.6)
  - X dead คือโปรเซสที่ตายไปแล้ว. จริง ๆแล้วจะไม่เห็นในรายการโปรเซส.
  - Z Defunct process หรือเรียกอีกอย่างว่าโปรเซส *zombie* คือโปรเซสที่ตายไปแล้วแต่โปรเซสพ่อแม่ไม่ได้รับรู้ทำให้ยังแสดงอยู่ในรายการโปรเซส. จากตัวอย่างที่ 5.8 จะเห็นว่า SZ และ RSS มีค่าเป็น 0 คือไม่ได้ใช้ทรัพยากร.

- PRI (Priority)
  แสดง priority ของโปรเซสตัวเลขอยู่ระหว่าง 0 ถึง 99. สำหรับโปรเซสที่อยู่ในรันคิว, โปรเซสที่มีค่า PRI สูงกว่าจะทำงานก่อน.

- NI (Nice)
  ตัวเลขนี้ช่วยบอกให้เคอร์เนลรู้ว่าควรจะรันโปรเซสนี้บ่อยหรือให้เวลาหน่วยประมวลผลอย่างไร. ค่าโดยปริยายจะเป็น 0. NI จะมีค่าตั้งแต่ -20 ถึง 19. ตัวเลขยิ่งมากจะได้เวลาใช้หน่วยประมวลผลน้อย, ค่าน้อย ๆจะได้เวลาประมวลผลมาก. Nice ในที่นี้หมายถึงดี (nice) ต่อโปรเซสอื่น ๆ.

- WCHAN (Wait)
  โปรเซสที่อยู่ในระบบไม่จำเป็นต้องกำลังทำงานอยู่ทุกตัว. ส่วนมากจะ sleep รอเหตุการณ์ใดเหตุการณ์หนึ่งอยู่และคอลัมน์ WCHAN แสดงชื่อฟังก์ชันเคอร์เนลที่โปรเซสนั้นกำลังรออยู่. ตัวอย่างเช่นโปรเซส bash อยู่ในสภาพ sleep (S) โดยฟังก์ชัน wait4. เราพอจะเดาได้ว่าเชลล์กำลังรอการทำงานของโปรเซส ps ให้เสร็จอยู่.

wait4 เป็นซิสเต็มคอลล์. อ่านรายละเอียดได้จาก `man wait4`.

คำสั่ง `ps` มีตัวเลือกสามารถข้อมูลที่ต้องการเองได้โดยใช้ตัวเลือก `-o`. ตัวอย่างต่อไปนี้จะแสดงโปรเซสโดยเลือกยูสเซอร์, จัดคอลัมน์ใหม่โดยระบุค่าที่ต้องการดูและแสดงเป็นแผนภาพต้นไม้ความสัมพันธ์ระหว่างโปรเซสด้วยตัวเลือก `--forest`.

ตัวอย่างที่ 5.10: โปรเซสของยูสเซอร์ที่ระบุและความสัมพันธ์ระหว่างโปรเซส.

```
$  ps -u poonlap -o pid,ppid,tty,stime,comm --forest↵
  PID  PPID TT       STIME COMMAND
10291  3725 ?        10:25 gnome-session
10373 10291 ?        10:25  \_ ssh-agent
10518     1 ?        10:26 notification-ar
10515     1 ?        10:26 gnome-keyboard-
10512     1 ?        10:26 mixer_applet2
10509     1 ?        10:26 clock-applet
10507     1 ?        10:26 multiload-apple
10503     1 ?        10:26 wnck-applet
10499     1 ?        10:26 gnome-terminal
10522 10499 ?        10:26  \_ gnome-pty-helpe
10523 10499 pts/1    10:26  \_ bash
11556 10523 pts/1    10:34  |   \_ firefox
11595 11556 pts/1    10:34  |   |   \_ run-mozilla.sh
11600 11595 pts/1    10:34  |   |       \_ firefox-bin
21634 11600 pts/1    12:54  |   |           \_ netstat <defunct>
 9394 10523 pts/1    17:41  |   \_ ps
```

```
10642 10499 pts/2    10:26  \_ bash
10679 10642 pts/2    10:26  |   \_ ssh
 1733 10499 pts/3    15:53  \_ bash
10483     1 ?        10:26 mapping-daemon
10477     1 ?        10:25 gnome-vfs-daemo
10466     1 ?        10:25 eggcups
10464     1 ?        10:25 pam-panel-icon
10459     1 ?        10:25 gnome-volume-ma
10457     1 ?        10:25 nautilus
10455     1 ?        10:25 gnome-panel
10450     1 ?        10:25 metacity
10413     1 ?        10:25 xscreensaver
10398     1 ?        10:25 gnome-settings-
10393     1 ?        10:25 bonobo-activati
10391     1 ?        10:25 gnome-keyring-d
10382     1 ?        10:25 gconfd-2
10377     1 ?        10:25 dbus-daemon-1
10376     1 ?        10:25 dbus-launch
10197     1 ?        10:24 esd
 7099     1 ?        09:58 gam_server
```

คำสั่ง ps เป็นคำสั่งที่ค่อนข้างซับซ้อน. ผู้อ่านสามารถดูรายละเอียดเพิ่มเติมได้จาก man ps.

5.2.4 โปรเซส และ thread

thread ▶ สายงานที่อยู่โปรเซสที่สามารถทำงานได้พร้อม ๆกัน. โปรแกรมที่ใช้ thread ได้จะใช้ไลบรารี pthread.

โปรเซสหนึ่งโปรเซสสามารถทำงานเป็นขั้นตอนที่กำหนดไว้ตั้งแต่ต้นจนจบได้หนึ่งชุด. ถ้าเรามีสายงานอยู่สองชุดที่ต้องทำพร้อม ๆกันจะทำอย่างไร? วิธีหนึ่งคือโปรเซสที่ทำงานอาจจะสร้างโปรเซสเพิ่มโดยการ fork และให้โปรเซสลูกนั้นทำงานที่สองพร้อม ๆกัน. การใช้วิธี fork เป็นการใช้โปรเซสสองตัว. อีกวิธีหนึ่งคือการใช้ *thread*. Thread คือสายงานที่อยู่ในโปรเซส. โปรเซสหนึ่งอาจจะมีได้หลายสายงานคือมี thread หลายสายทำงานได้พร้อม ๆกันซึ่งแต่ละสายงานอาจจะเหมือนกันหรือต่างกันก็ได้. การใช้ thread มีข้อดีที่ว่าไม่มีการสลับเปลี่ยนการทำงานโปรเซสของหน่วยประมวลเพราะ thread เป็นสายงานที่อยู่ในโปรเซสและ thread สามารถใช้ข้อมูลบางอย่างที่อยู่ในหน่วยความจำร่วมกันได้. ส่วนการแบ่งเวลาการทำงานของ thread ในโปรเซสนั้นจะใช้ไลบรารีไลบรารี pthread (POSIX thread library).

ไลบรารี thread สำหรับลินุกซ์ในช่วงแรก ๆไม่สมบูรณ์และมีปัญหาหลายอย่าง [41]. ตัวอย่างที่เห็นได้ชัดเช่นโปรแกรมที่ใช้ไลบรารีรุ่นเก่าเวลาแสดงโปรเซสด้วยคำสั่ง ps จะเห็น thread เป็นโปรเซสหลายโปรเซสแยกแยะไม่ออกว่าจริง ๆแล้วโปรเซสไหนเป็น thread หรือโปรเซสจริง ๆ. ไลบรารี thread รุ่นใหม่ที่เรียกว่า *Native POSIX Thread Library (NPTL)* จะไม่มีปัญหาแบบนี้. เวลาใช้คำสั่ง ps แสดงโปรเซสจะเห็นเป็นโปรเซสเดียว.

คำสั่ง ps มีตัวเลือกที่ใช้แสดง thread ID (LWP) ได้แก่ -L. ถ้าใช้ตัวเลือกนี้ร่วมกับตัวเลือก -f จะแสดงจำนวน thread (NLWP) ของโปรเซสด้วย.

ตัวอย่างที่ 5.11: โปรเซสที่ใช้ thread.

```
$ ps -e | grep firefox-bin↵                    ← หาโปรเซส ID ของ firefox-bin
```

```
 7379 pts/0    00:17:03 firefox-bin
$ ps -fLp 7379↵
UID        PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
poonlap   7379  7369  7379  0    6 Nov15 pts/0    00:17:03 /usr/lib/MozillaFiref
ox/firefox-bin
poonlap   7379  7369  7389  0    6 Nov15 pts/0    00:00:04 /usr/lib/MozillaFiref
ox/firefox-bin
poonlap   7379  7369  7391  0    6 Nov15 pts/0    00:00:06 /usr/lib/MozillaFiref
ox/firefox-bin
poonlap   7379  7369  7781  0    6 Nov15 pts/0    00:00:00 /usr/lib/MozillaFiref
ox/firefox-bin
poonlap   7379  7369 12544  0    6 Nov16 pts/0    00:00:00 /usr/lib/MozillaFiref
ox/firefox-bin
poonlap   7379  7369 12545  0    6 Nov16 pts/0    00:00:00 /usr/lib/MozillaFiref
ox/firefox-bin
```

จากตัวอย่างจะเห็นว่าโปรเซส firefox-bin มีโปรเซส ID เป็น 7379 และมี thread อยู่ 6 ตัว. Thread แต่ละตัวจะมีหมายเลขเฉพาะแสดงในคอลัมน์ LWP ซึ่งย่อมาจากคำว่า *Light Weight Process* ซึ่งคือ thread นั่นเอง.

5.2.5 หาโปรเซส ID

คำสั่ง ps เป็นคำสั่งที่แสดงข้อมูลต่าง ๆเกี่ยวกับโปรเซส. ถ้าเรารู้ชื่อโปรเซสหรือชื่อโปรแกรมแล้วต้องการหาโปรเซส ID ของโปรเซสนั้น, อาจจะใช้คำสั่ง grep ช่วยในการกรองผลลัพธ์ของคำสั่ง ps อีกที. ตัวอย่างเช่นเราต้องการค้นหาโปรเซส ID ของโปรแกรม emacs ก็สามารถสั่งคำสั่งดังนี้

ตัวอย่างที่ 5.12: การใช้ ps ร่วมกับ grep เพื่อหาโปรเซส ID จากชื่อ

```
$ ps -ef | grep emacs↵
poonlap   4837  4831  0 14:51 pts/2    00:02:41 emacs
poonlap   6162  4811  0 22:01 pts/1    00:00:00 grep emacs
```

ในกรณีนี้เราก็จะรู้ว่าโปรเซส ID ของ emacs คือ 4837. หลังจากนั้นถ้าเราต้องการติดต่อกับโปรเซสนี้เช่น หยุดการทำงานของโปรเซสนี้ก็สามารถทำได้โดยอ้างอิงโปรเซส ID ที่ได้มา. การใช้ grep เข้าช่วยทำให้รู้โปรเซส ID ของโปรแกรมที่ต้องการแต่สุดท้ายเราต้องมาดูอยู่ดีว่าโปรเซส ID ของ emacs คือตัวเลขที่อยู่บรรทัดแรกคอลัมน์ที่สอง.

คำสั่ง pgrep เป็นคำสั่งที่รวมความสามารถของ ps และ grep เข้าด้วยกันใช้หาโปรเซส ID โดยระบุชื่อโปรเซสแบบ regular expression โดยปริยายถ้าไม่ระบุตัวเลือก.

pgrep อ้างอิงหน้า 402

ตัวอย่างที่ 5.13: การใช้โปรแกรม pgrep หาโปรเซส ID

```
$ pgrep -x emacs↵
4837
$ pgrep -lx emacs↵
4837 emacs
```

จากตัวอย่าง, เป็นการหาโปรเซส ID ของโปรเซสชื่อ `emacs` โดยใช้ตัวเลือก `-x` เพื่อให้คำสั่งแมชท์คำนั้นตรงตัว. ถ้าไม่ใช้ตัวเลือก `-x` ประกอบ, คำสั่ง `pgrep` อาจจะแสดงโปรเซส ID ของโปรเซสที่มีคำว่า `emacs` อยู่ซึ่งอาจจะเป็น `emacs`, `xemacs`, `emacsclient` ฯลฯ เพราะคำที่ระบุเป็น regular expression. ตัวเลือก `-l` ใช้แสดงชื่อโปรเซสร่วมกับโปรเซส ID.

5.3 สัญญาณ (signal)

สัญญาณ (signal) เป็นวิธีสื่อสารวิธีหนึ่งระหว่างโปรเซส. ระบบปฏิบัติการลินุกซ์ได้มีการกำหนดสัญญาณต่าง ๆซึ่งคล้ายเหมือนกับระบบปฏิบัติการยูนิกซ์. เมื่อโปรเซสใดโปรเซสหนึ่งได้รับสัญญาณหนึ่งจะมีการตอบสนองต่อสัญญาณนั้น ๆโดยปริยาย. ตารางที่ 5.1 แสดงรายการสัญญาณทั่ว ๆไปที่ใช้กัน.

ตารางที่ 5.1: สัญญาณต่าง ๆและหมายเลขที่กำหนดโดยระบบปฏิบัติการ

สัญญาณ	หมายเลข	การตอบสนอง	คำอธิบาย
SIGHUP	1	Term	รับรู้การ hangup จากเทอร์มินอลที่ควบคุมอยู่หรือสูญเสียการควบคุมโปรเซส
SIGINT	2	Term	ยับยั้งจากคีย์บอร์ด
SIGQUIT	3	Core	จบการทำงานจากคีย์บอร์ด
SIGILL	4	Core	คำสั่งปฏิบัติการที่ไม่ถูกต้อง
SIGABRT	6	Core	ทำให้ล้มเหลวจากซิสเต็มคอล abort
SIGFPE	8	Core	ข้อยกเว้น (exception) แบบ floating point
SIGKILL	9	Term	สัญญาณ kill
SIGSEGV	11	Core	การ อ้าง อิง หน่วย ความ จำ ที่ เป็น โมฆะ
SIGPIPE	13	Term	การส่งต่อข้อมูลให้ไปป์ที่ไม่มีตัวรับข้อมูลต่อ (broken pipe)
SIGALRM	14	Term	การปลุกจากซิสเต็มคอล alarm
SIGTERM	15	Term	สัญญาณสิ้นสุดการทำงาน
SIGUSR1	30,10,16	Term	สัญญาณกำหนดโดยผู้ใช้ 1
SIGUSR2	31,12,17	Term	สัญญาณกำหนดโดยผู้ใช้ 2
SIGCHLD	20,17,18	Ign	โปร เซส ลูก ถูก หยุด หรือ สิ้น สุด การ ทำงาน

ต่อหน้าถัดไป

ต่อจากหน้าที่แล้ว

สัญญาณ	หมายเลข	การตอบสนอง	คำอธิบาย
SIGCONT	19,18,25		กระทำการต่อไปถ้าหยุดอยู่
SIGSTOP	17,19,23	Stop	หยุดโปรเซส
SIGTSTP	18,20,24	Stop	หยุดการพิมพ์จากเทอร์มินอล

การตอบสนองโดยปริยายมีหลายประเภทเช่น

- **Term**
  สิ้นสุดโปรเซส. การสิ้นสุดโปรเซสนี้ไม่ได้หมายถึงโปรเซสจบการทำงานโดยสมบูรณ์แบบแต่หมายถึงโปรเซสที่ตอบสนองนั้นจะสิ้นสุดการกระทำการทันทีไม่ว่างานที่ทำอยู่นั้นจะเสร็จบริบูรณ์หรือไม่ก็ตาม.

การจบการทำโดยสมบูรณ์หมายถึงโปรเซสมีโอกาสสามารถกระทำการสิ่งต่าง ๆที่จำเป็นก่อนที่จะจบการทำงาน.

- **Ign**
  เป็นการตอบสนองโดยเพิกเฉยต่อสัญญาณที่ได้รับ.

- **Core**
  ตอบสนองโดยการสิ้นสุดโปรเซสและ*ดัมพ์ (dump)* เนื้อหาของหน่วยความจำที่โปรเซสนั้นใช้ลงไฟล์ชื่อว่า `core`. โดยทั่วไปเรียกว่า *core dump*.

- **Stop**
  หยุดการทำงานของโปรเซสชั่วคราว. โปรเซสที่หยุดการทำงานนี้สามารถทำให้ทำงานต่อได้.

ตัวอย่างเช่นเมื่อโปรเซสได้รับสัญญาณ SIGINT, โปรเซสนั้นก็จะยกเลิกการทำงานโดยปริยาย. ถ้าโปรเซสนั้นมีความสามารถดักจับรอสัญญาณที่ได้ก็จะสามารถกระทำการอื่น ๆที่ไม่ใช่การตอบสนองโดยปริยายก็ได้. สำหรับสัญญาณโดยทั่ว ๆไป, โปรเซสที่ได้รับสัญญาณนั้นสามารถดัก, บล็อกหรือเพิกเฉยการตอบสนองที่กำหนดไว้ได้. สัญญาณ SIGKILL และ SIGSTOP เป็นสัญญาณที่ไม่สามารถดัก, บล็อกหรือเพิกเฉยได้. กล่าวคือถ้าโปรเซสใดโปรเซสหนึ่งได้รับสัญญาณ SIGKILL ก็จะสิ้นสุดการทำงานทันทีเสมอโดยไม่มีข้อยกเว้น. ส่วนโปรเซสที่ได้รับสัญญาณ SIGSTOP ก็จะหยุดการทำงานชั่วคราวเสมอโดยไม่มีข้อยกเว้นเช่นกัน. ตัวอย่างการส่งสัญญาณ SIGSTOP จากเชลล์ (`C-z`) ได้แสดงไปแล้วในตอนต้น.

สัญญาณ SIGINT สามารถส่งได้โดยการกดคีย์ `C-c` จากเทอร์มินัล, ใช้ในการยกเลิกการทำงานของจ็อบแบบ foreground. ตัวอย่างเช่น

ตัวอย่างที่ 5.14: การส่งสัญญาณ SIGINT ให้โปรแกรมที่ใช้อยู่ด้วย `C-c`

```
$ cat↵
'cat' will repeat what you typed from standard input↵
'cat' will repeat what you typed from standard input
Now we are going to press C-c↵
Now we are going to press C-c
Ctrl+c
$
```

จากตัวอย่างดังกล่าวเป็นการยกเลิกการทำงาน, ไม่ใช่การจบการทำงานอย่างถูกต้อง. ในกรณีของโปรแกรม cat จะเรียกได้ว่าจบการทำงานบริบูรณ์ก็ต่อเมื่อไม่มีข้อมูลป้อนให้โปรแกรม cat อีกต่อไป. กล่าวคือเมื่อได้รับอักขระควบคุม EOT (End Of Transmission) คือกด C-d.

EOT บ้างก็เรียกว่า EOF (End Of File).

โดยทั่วไปผู้ใช้สามารถใช้คีย์ C-c ส่งสัญญาณ SIGINT เพื่อยกเลิกการทำงานของโปรแกรม foreground ต่าง ๆได้. แต่โปรแกรมก็มีสิทธิ์ที่จะเพิกเฉยการรบกวนของสัญญาณ SIGINT ได้โดยการดักสัญญาณ SIGINT และกระทำสิ่งที่โปรแกรมกำหนดไว้. ตัวอย่างเช่นถ้าผู้ใช้ใช้โปรแกรมเครื่องคิดเลข bc แล้วกด C-c เพื่อยกเลิกการใช้การใช้งาน, ตัวโปรแกรม bc จะดักสัญญาณ SIGINT แล้วแสดงข้อความการใช้งานที่ถูกต้องทางหน้าจอแทนที่จะจบกระทำการกลางครร.

bc อ้างอิงหน้า 409

ในโปรแกรม bc ผู้ใช้สามารถสร้างตัวแปรและใช้ฟังชันคณิตศาสตร์ต่าง ๆ ได้เช่น sin, log, ยกกำลังเป็นต้น. ในตัวอย่างมีการกำหนดค่า π โดยอาศัยฟังชัน $arctan(1)$ เข้าช่วย, $\pi = 4arctan(1)$.

ตัวอย่างที่ 5.15: โปรแกรมที่ดักสัญญาณ SIGINT

```
$ bc -l↵
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type ‘warranty’.
pi = 4*a(1)↵
r = 2.5↵
area = pi * r^2↵
area↵
19.63495408493620774025
4*a(1) * 2.5^2↵
19.63495408493620774025
Ctrl+C
(interrupt) use quit to exit.
quit↵
$
```

ตามตัวอย่างที่แสดงไปขั้นต้นจะเห็นได้ว่าสัญญาณบางอย่างผู้ใช้สามารถส่งได้โดยใช้ key binding เช่น C-c, C-z. แต่สำหรับการส่งสัญญาณโดยทั่วไปแล้วผู้ใช้ทำได้โดยใช้คำสั่ง kill โดยส่งโปรเซส ID เป็นอาร์กิวเมนต์ของคำสั่ง. ผู้ใช้สามารถหาโปรเซส ID ได้จากคำสั่ง ps หรือ pgrep ที่ได้แนะนำไปแล้ว (5.1).

kill อ้างอิงหน้า 402

โดยปรกติคำสั่ง kill จะเป็นการทำให้โปรเซสหรือจ็อบที่ระบุสิ้นสุดการทำงาน. ในหลักการแล้ว kill จะเป็นตัวส่งสัญญาณ SIGTERM เพื่อให้โปรเซสหรือจ็อบที่ต้องการหยุดการทำงาน. ตัวอย่างเช่น

ตัวอย่างที่ 5.16: การใช้คำสั่ง kill โดยระบุหมายเลขโปรเซส

```
$ emacs &↵
[1] 4362
$ kill 4362↵
```

แทนที่จะระบุหมายเลขโปรเซสเราอาจจะระบุหมายเลขจ็อบแทนก็ได้เช่น

ตัวอย่างที่ 5.17: การใช้คำสั่ง `kill` โดยระบุหมายเลขจ็อบ

```
$ emacs &↵
[1] 4410
$ kill %1↵
```

นอกจากสัญญาณ SIGTERM แล้ว, โปรแกรม `kill` ยังสามารถส่งสัญญาณอื่น ๆให้โปรเซสได้ด้วย. รายการสัญญาณที่โปรแกรม `kill` ส่งได้ใช้ตัวเลือก `-l` ในการแสดง.

ตัวอย่างที่ 5.18: ใช้คำสั่ง `kill` แสดงชื่อสัญญาณต่าง ๆ

```
$ kill -l↵
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR      31) SIGSYS      32) SIGRTMIN    33) SIGRTMIN+1
34) SIGRTMIN+2  35) SIGRTMIN+3  36) SIGRTMIN+4  37) SIGRTMIN+5
38) SIGRTMIN+6  39) SIGRTMIN+7  40) SIGRTMIN+8  41) SIGRTMIN+9
42) SIGRTMIN+10 43) SIGRTMIN+11 44) SIGRTMIN+12 45) SIGRTMIN+13
46) SIGRTMIN+14 47) SIGRTMIN+15 48) SIGRTMAX-15 49) SIGRTMAX-14
50) SIGRTMAX-13 51) SIGRTMAX-12 52) SIGRTMAX-11 53) SIGRTMAX-10
54) SIGRTMAX-9  55) SIGRTMAX-8  56) SIGRTMAX-7  57) SIGRTMAX-6
58) SIGRTMAX-5  59) SIGRTMAX-4  60) SIGRTMAX-3  61) SIGRTMAX-2
62) SIGRTMAX-1  63) SIGRTMAX
```

ในกรณีที่ `kill` ไม่สามารถสิ้นสุดการทำงานของโปรเซสที่ต้องการได้, ผู้ใช้สามารถส่งสัญญาณ SIGKILL ด้วยอาร์กิวเมนต์ `-9` หรือ `-SIGKILL` ซึ่งมีความหมายเหมือนกัน. สัญญาณนี้ใช้สำหรับฆ่าโปรเซสที่ผิดปรกติเช่นโปรเซสที่ไม่ตอบสนองการทำงาน. เนื่องจาก SIGKILL เป็นสัญญาณที่โปรเซสไม่สามารถปฏิเสธได้, เมื่อโปรเซสได้รับสัญญาณ SIGKILL แล้ว, ตัวโปรเซสนั้นจะสิ้นสุดการทำงานแน่นอน.

การใช้คำสั่ง `kill` ฆ่าโปรเซสต้องรู้โปรเซส ID ของโปรเซสที่ต้องการก่อนจึงจะใช้ส่งสัญญาณหาโปรเซสนั้น ๆได้. ในความเป็นจริง, ผู้ใช้ต้องค้นหาโปรเซส ID ด้วยคำสั่ง `ps` หรือ `pgrep` เอง. เมื่อรู้โปรเซส ID แล้วจึงสามารถใช้คำสั่ง `kill` ส่งสัญญาณ. บางดิสทริบิวชันอาจจะมีคำสั่ง `pkill` ซึ่งรวมการทำงานของคำสั่ง `kill` กับ `grep` เข้าด้วยกันให้ส่งสัญญาณให้โปรเซสที่ต้องการโดยระบุชื่อหรือ regular expression. คำสั่ง `pkill` จะมีตัวเลือกส่วนใหญ่เหมือนกับคำสั่ง `pgrep`.

`pkill` อ้างอิงหน้า 402

คำสั่งที่เกี่ยวกับการส่งสัญญาณอีกตัวหนึ่งคือ `killall` ใช้ส่งสัญญาณให้โปรเซสทุกตัวที่ต้องการโดยการระบุชื่อโปรเซส. จะต่างกับคำสั่ง `pkill` ที่ไม่สามารถใช้ regular expression. คำสั่ง `killall` สำหรับระบบปฏิบัติการยูนิกซ์อื่น ๆที่ไม่ใช้ลินุกซ์อาจจะมีพฤติกรรมที่แตกต่างจากคำสั่ง `killall` ในลินุกซ์. เช่นในระบบปฏิบัติการ Solaris, ถ้าสั่ง `killall` ด้วย root จะทำลายโปรเซสทุกตัวในระบบตามคำแปลของชื่อคำสั่ง.

`killall` อ้างอิงหน้า 402

5.4 ทรัพยากร

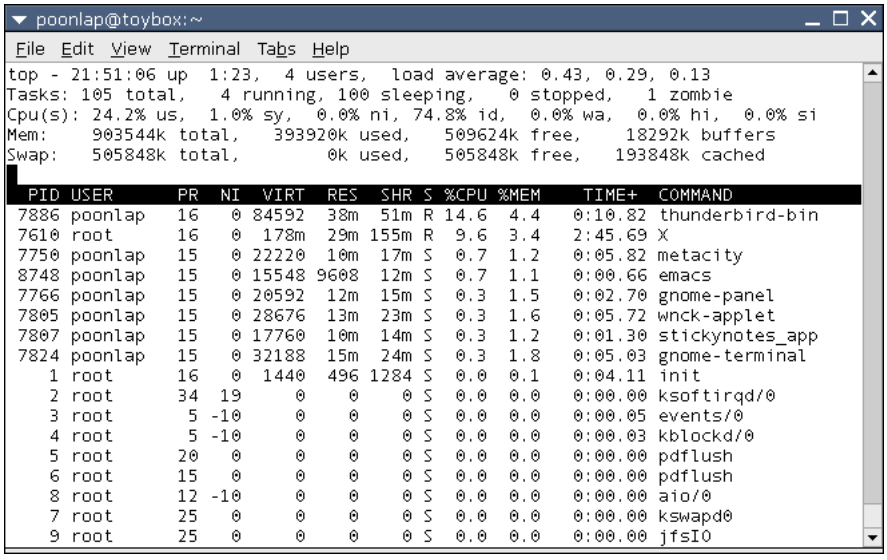
คำว่า “ทรัพยากร” ในภาษาไทยอาจจะฟังดูแปลก ๆสำหรับหนังสือคอมพิวเตอร์. ถ้าจะเขียนเป็นภาษาอังกฤษจะใช้คำว่า resource ซึ่งหมายถึงทรัพยากรของเครื่องคอมพิวเตอร์ได้เช่น เวลาที่หน่วยประมวลผลใช้ไปกับโปรเซสหนึ่ง ๆ, หน่วยความจำจริงที่ใช้ไป, พื้นที่ของฮาร์ดดิสก์ เป็นต้น. เคอร์เนลเป็นตัวที่ควบคุมการใช้ทรัพยากรต่าง ๆของโปรเซสเพื่อให้ระบบทั้งระบบทำงานได้อย่างมีประสิทธิภาพ.

ลินุกซ์เป็นระบบปฏิบัติการแบบมัลติยูสเซอร์, มัลติทาส์ก สามารถมีโปรเซสหลาย ๆตัวทำงานในเวลาเดียวกัน. บางโปรเซสใช้ทรัพยากรมาก, บางโปรเซสใช้ทรัพยากรน้อยแตกต่างกันไป. คำสั่ง ps สามารถดูทรัพยากรที่โปรเซสหนึ่ง ๆใช้ไปได้เช่น เปอร์เซนต์การใช้งานหน่วยประมวลผลในช่วงเวลาสั้น ๆ (CPU), หน่วยความจำที่ใช้ไปกับโปรเซส ฯลฯ. แต่สำหรับการดูการใช้ทรัพยากรต่าง ๆในระบบแบบทันต่อเวลา (real time) มักจะใช้คำสั่ง top เพราะจะแสดงการใช้ทรัพยากรต่าง ๆที่สำคัญ ๆของโปรเซสตามลำดับเปอร์เซ็นต์การใช้งานหน่วยประมวลผล.

ใช้วิธีการเลือกคอลัมน์ -o pcpu.

top อ้างอิงหน้า 404

5.4.1 คำสั่ง top



รูปที่ 5.3: คำสั่ง top แสดงโปรเซสต่าง ๆและทรัพยากรที่ใช้.

คำสั่ง top เป็นโปรแกรมที่ใช้เทอร์มินอลชั่วแสดงผลจะอัปเดทข้อมูลทางหน้าจอทุก ๆ 3 วินาทีโดยปริยาย. ข้อมูลในบรรทัดแรกเป็นข้อมูลโหลดโดยเฉลี่ย (load average) ของระบบซึ่งเหมือนกับผลลัพธ์ของคำสั่ง uptime. บรรทัดที่ 2 แสดงจำนวนโปรเซสและสถานะต่างของโปรเซสได้แก่

- จำนวนโปรเซสทั้งหมดในระบบ (Tasks)

- จำนวนโปรเซที่อยู่ในรันคิว (running). ถ้ามีจำนวนโปรเซสอยู่ในรันคิวมากจำทำให้ค่าโหลดโดยเฉลี่ยสูง.
- จำนวนโปรเซสที่กำลังรอทำงานต่อไป (sleeping)
- จำนวนโปรเซสที่หยุดชั่วคราว (stopped) คือโปรเซสที่ได้รับสัญญาณ SIGSTOP.
- จำนวนโปรเซส zombie.

บรรทัดที่ 3 แสดงเปอร์เซ็นต์การทำงานของหน่วยประมวลผลในช่วงที่ผ่านมาได้แก่

- user (us) เปอร์เซ็นต์การใช้หน่วยประมวลผลไปกับโปรเซสธรรมดาที่ทำงานอยู่ใน user space.
- system (sy) เปอร์เซ็นต์การใช้หน่วยประมวลผลไปกับโปรเซสที่ทำงานอยู่ใน kernel space.
- nice (ni) เปอร์เซ็นต์การใช้หน่วยประมวลผลไปกับโปรเซสที่มีค่า nice ทำงานอยู่ใน user space.
- idle (id) เปอร์เซ็นต์ที่หน่วยประมวลผลไม่ได้ทำอะไร.
- iowait (wa) เปอร์เซ็นต์การใช้หน่วยประมวลผลรอให้ I/O เสร็จเรียบร้อย.
- irq (hi) เปอร์เซ็นต์การใช้หน่วยประมวลผลที่ใช้ไปกับการ interrupt ของฮาร์ดแวร์.
- softirq (si) เปอร์เซ็นต์การใช้หน่วยประมวลผลที่ใช้ไปกับการ interrupt แบบซอฟต์แวร์.

บรรทัดที่ 4 เป็นข้อมูลเกี่ยวกับหน่วยความจำในระบบในหน่วยกิโลไบต์ได้แก่

- จำนวนหน่วยความจำจริงทั้งหมด (total).
- จำนวนหน่วยความจำที่ใช้อยู่ (used).
- จำนวนหน่วยความจำที่ยังเหลืออยู่ (free).
- จำนวนหน่วยความจำที่ใช้ไปกับ *buffer cache* (buffers). หน่วยความจำส่วนนี้ใช้สำหรับเก็บข้อมูลที่จะบันทึกลงในฮาร์ดดิสก์, หรือข้อมูลที่อ่านมาจากฮาร์ดดิสก์. การอ่านข้อมูลจากฮาร์ดดิสก์จะใช้เวลานานเมื่อเทียบกับการอ่านข้อมูลจากหน่วยความจำ. ดังนั้นเคอร์เนลจึงใช้เนื้อที่หน่วยความจำส่วนนี้ช่วยทุ่นเวลา I/O ที่จะเกิดกับฮาร์ดดิสก์.
- จำนวนหน่วยความจำที่ใช้ไปกับ *page cache* (cached) คือ cache ของ page ที่แม็ปไว้กับไฟล์. ข้อมูลนี้อยู่ในบรรทัดที่ห้าแต่เป็นข้อมูลการใช้งานหน่วยความจำ.

cache ▶
การเก็บข้อมูลบางอย่างไว้ช่วยคราวเพื่อความรวดเร็วในการเข้าถึงข้อมูลนั้นภายหลัง. ถ้าใช้คำว่า cache อย่างเดียวไม่สามารถบอกได้ว่าเป็น cache ของอะไร. ลินุกซ์เคอร์เนลใช้หน่วยความจำในการ cache ข้อมูลหลายอย่างเช่น buffer cache, page cache, inode cache, directory cache และ swap cache.

บรรทัดที่ 5 แสดงข้อมูลเกี่ยวกับ *swap* ในระบบ. Swap เป็นพื้นที่ในฮาร์ดดิสก์ใช้เก็บข้อมูลแทนหน่วยความจำ. ตัวอย่างการใช้ swap ของเคอร์เนลเช่นในกรณีที่มีข้อมูลบางอย่างในโปรเซสที่ไม่มีความจำเป็นต้องอยู่ในหน่วยความจำจริงก็จะเก็บไว้ใน swap (ฮาร์ดดิสก์) แทนไม่ให้เปลืองที่โดยใช่เหตุเป็นต้น. หน่วยความจำที่ไปอยู่ใน swap นี้เรียกว่าถูก *swap out* ถ้ามีการเรียกข้อมูลจาก swap เข้าไปในหน่วยความจำใหม่ก็เรียกว่า *swap in*

swap ทำงานกลับกัน
cache.

- จำนวน swap ที่มีอยู่ในระบบ (total).
- จำนวน swap ที่ใช้ไป (used).
- จำนวน swap ที่ยังเหลืออยู่ (free).

ถัดจากสถิติโดยรวมของระบบแล้วก็จะเป็นรายงานการใช้ทรัพยากรต่าง ๆของโปรเซสโดยเรียงตามลำดับโปรเซสที่ใช้งานหน่วยความจำมาก. ถ้าต้องการจะออกจากโปรแกรม `top` ให้กดคีย์ “q” หมายถึง quit ก็จะจบการทำงานแสดงเชลล์พรอมต์ต่อไป.

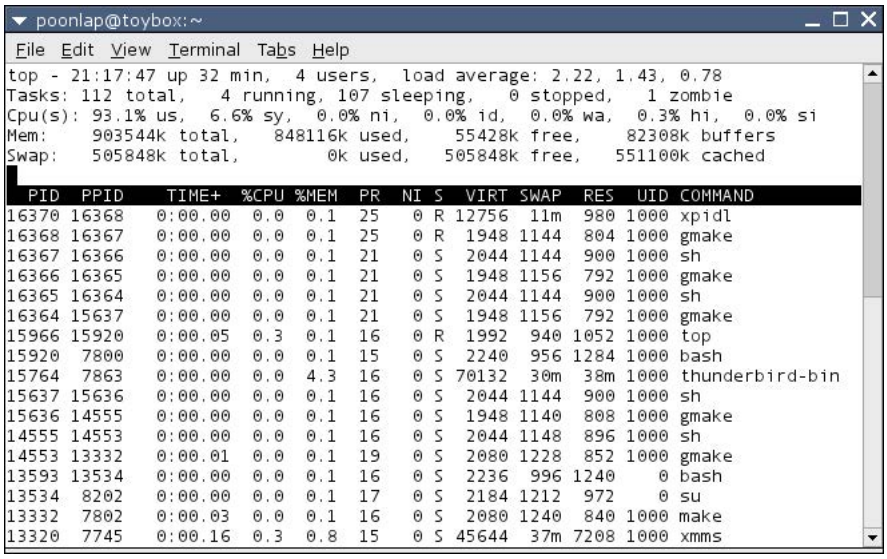
รายละเอียดการใช้ทรัพยากรของโปรเซสจะคล้ายเมหือนกับผลลัพธ์ของคำสั่ง `ps` เช่น PID, USER, PR, NI เป็นต้น. ส่วนที่ต้องอธิบายเพิ่มเติมได้แก่

- **VIRT**
  หน่วยความจำเสมือนที่โปรเซสใช้ในหน่วยของ kB ประกอบด้วยจำนวนหน่วยความจำที่ใช้ไปกับ code, data, shared library และ page ที่ถูก swap out ออกไป. VIRT = SWAP + RES.
- **RES**
  ได้แก่หน่วยความจำ Resident Size ในหน่วย kB ได้แก่หน่วยความจำจริงที่ไม่ได้ swap out. RES = CODE + DATA.
- **SHR**
  Shared memory ได้แก่หน่วยความจำที่ใช้ร่วมกันระหว่างโปรเซสในหน่วย kB.

5.4.2 กลุ่มการแสดงผลของคำสั่ง `top`

คำสั่ง `top` จะแสดงผลโดยใช้หน้าต่างเทอร์มินอลและหน้าต่างที่แสดงในรูปที่ 5.3 เป็นเพียง 1 ใน 4 ของกลุ่มแสดงผลที่เตรียมและคอลัมน์ต่าง ๆที่แสดงในแต่ละกลุ่มจะแยกแยะไว้ไม่เหมือนกันแล้วแต่จุดประสงค์ของกลุ่มแสดงผล. กลุ่มแสดงผลมีดังนี้

1. **Def** (Default) กลุ่มนี้เป็นหน้าต่างโดยปริยายใช้แสดงรายละเอียดโปรเซสต่าง ๆ ตามลำดับโดยเน้นเปอร์เซ็นต์การใช้งานหน่วยประมวลผลเป็นหลัก.
2. **Job** สำหรับแสดงผลเน้นจ็อบ. จะแสดงโปรเซสตามลำดับโปรเซส ID จากมากไปหาน้อยโดยปริยาย. คอลัมน์ต่าง ๆที่แสดงไม่ต่างจากกลุ่ม Def มากนัก. ข้อมูลที่แสดงแทนหรือเพิ่มเข้ามาได้แก่ PPID, UID และ SWAP.



รูปที่ 5.4: กลุ่มการแสดงผล Job ของคำสั่ง `top`.

- **PPID** โปรเซส ID ของโปรเซสพ่อแม่.
- **UID** ยูสเซอร์ ID.
- **SWAP** แสดงหน่วยความจำของโปรเซสที่ถูก swap out จากหน่วยความจำเสมือนในหน่วย kB.

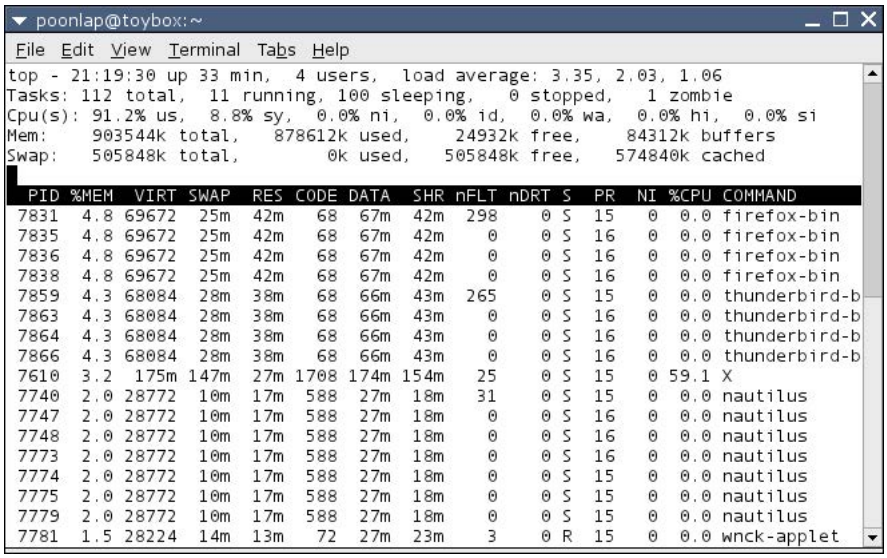
3. **Mem** (Memory) กลุ่มแสดงผลที่เน้นสำหรับข้อมูลเกี่ยวกับหน่วยความจำโดยเฉพาะ. รายการโปรเซสจะเรียงลำดับตามค่า %MEM ซึ่งได้แก่จำนวนหน่วยความจำจริงที่ใช้. รายละเอียดที่แสดงเพิ่มเติมในกลุ่มนี้ได้แก่

   - **CODE** หน่วยความจำจริงที่ใช้สำหรับส่วนที่เป็น code. รู้จักกันในอีกชื่อว่า *Text Resident Set (TRS)*.
   - **nFLT** จำนวน major page fault ที่เกิดขึ้นเนื่องจากการเขียนหรืออ่าน address ที่ไม่ได้อยู่หน่วยความจำแต่อยู่ในฮาร์ดดิสก์.
   - **nDRT** จำนวน dirty page ได้แก่หน่วยความจำที่ข้อมูลมีการเปลี่ยนแปลงต้องบันทึกลงในฮาร์ดดิสก์.

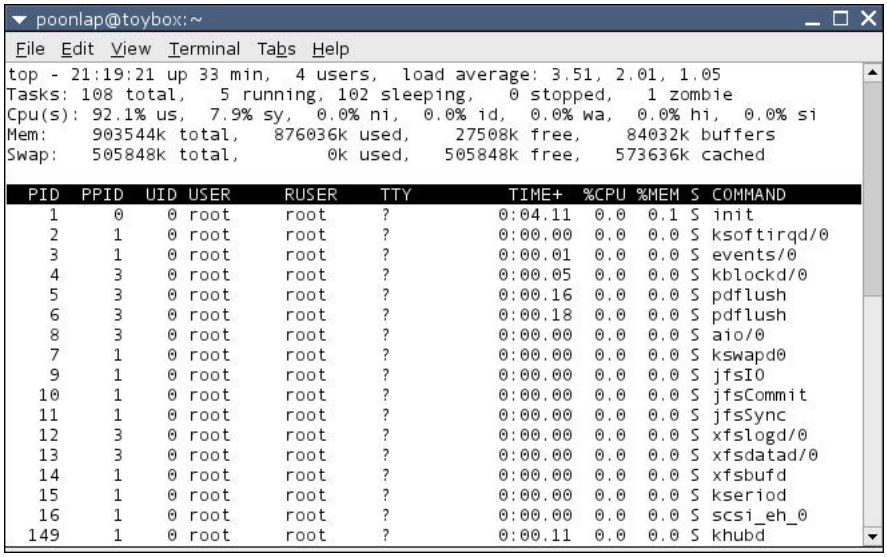
4. **Usr** (User) กลุ่มแสดงผลเน้นข้อมูลที่เกี่ยวกับยูสเซอร์โดยเรียงลำดับโปรเซสตามชื่อยูสเซอร์. รายละเอียดที่เพิ่มเข้ามาได้แก่

   - **RUSER** ยูสเซอร์จริงที่เป็นเจ้าของโปรเซส.
   - **TTY** ชื่อเทอร์มินอลควบคุมโปรเซส.

การเปลี่ยนกลุ่มการแสดงผลทำได้โดยการกดคีย์ “G” แล้วกดเลข 1 ถึง 4 ตามกลุ่มที่ต้องการแสดง. ถ้าต้องการแสดงหน้าต่างทั้ง 4 แบบพร้อม ๆกันให้กดคีย์ “A”. มุมบนซ้าย



รูปที่ 5.5: กลุ่มการแสดงผล Mem ของคำสั่ง top.

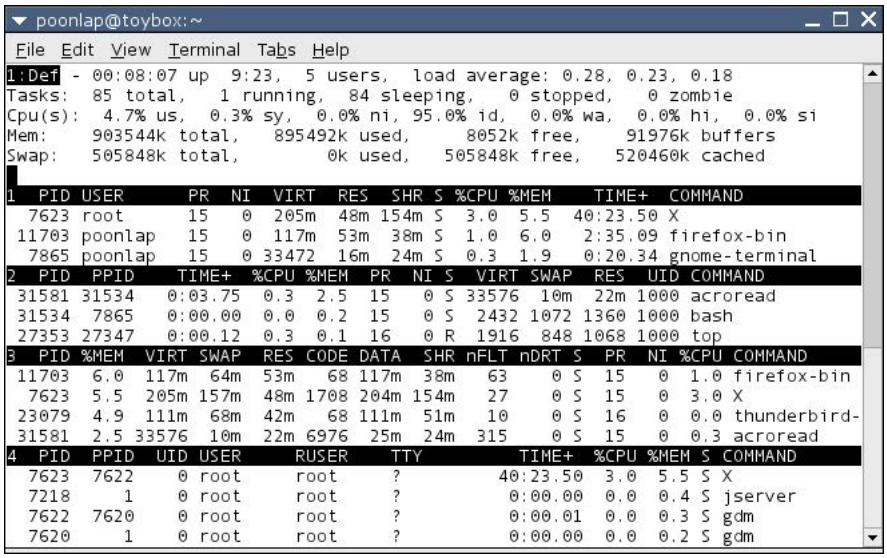


รูปที่ 5.6: กลุ่มการแสดงผล Usr ของคำสั่ง top.

ของหน้าจอจะแสดงหน้าต่างปัจจุบันที่เลือกอยู่. ถ้ามีการโต้ตอบกับโปรแกรม top จะถือว่าเป็นการกระทำกับหน้านั้น. ถ้าต้องการเปลี่ยนหน้าต่างที่เลือกอยู่เป็นหน้าต่างอื่นให้กดคีย์ “a”.

5.4.3 คำสั่งในโปรแกรม top

โปรแกรม top เป็นโปรแกรมแบบโต้ตอบ, ผู้ใช้สามารถกดคีย์ต่อไปนี้กระทำการต่าง ๆ ได้. คำสั่งเหล่านี้ใช้ได้กับหน้าต่างกลุ่มการแสดงแสดงผลทุกแบบ.



รูปที่ 5.7: หน้าจอ `top` เมื่อแสดงหน้าต่าง 4 แบบพร้อมๆกัน.

- “q” จบการทำงาน.
- “h” แสดงหน้าจอช่วยเหลือ. การออกจากหน้าจอช่วยเหลือให้กดคีย์ใดๆ.
- “z” ใช้สีในการแสดงผล. ถ้ากด “Z” จะแสดงคำอธิบายสั้นๆก่อนแล้วใช้สีตามภายหลัง. ถ้าต้องการเลิกใช้สีให้กด “z” อีกที.
- “b” แสดงบรรทัดโปรเซสที่อยู่ในรันคิว (สถานะ R) ให้มีสีพื้นหลังกับสีตัวอักษรกลับกัน. ในกรณีจะทำให้เห็นโปรเซสสถานะ R เด่นชัดขึ้น.
- “B” แสดงบรรทัดโปรเซสที่สถานะ R และค่าสถิติโดยรวมต่างๆด้วยตัวหนา.
- “l” ซ่อน/แสดง บรรทัดแรกที่เป็นผลลัพธ์ของ `uptime`.
- “t” ซ่อน/แสดง บรรทัดสถิติจำนวนโปรเซสแบบต่างๆ.
- “m” ซ่อน/แสดง บรรทัดที่เกี่ยวกับหน่วยความจำและ swap.
- “u” แสดงโปรเซสของยูสเซอร์ที่ต้องการ. ให้ผลเหมือนกับการสั่งคำสั่งด้วยตัวเลือก `-u` *`user`*.
- “n” หรือ “#” ระบุจำนวนโปรเซสที่ต้องการดู.
- “d” หรือ “s” ตั้งช่วงเวลาการแสดงผลในแต่ละครั้งในหน่วยวินาที. คำสั่ง `top` จะแสดงผลทุกๆ 3 วินาทีโดยปริยาย. การตั้งช่วงเวลาการแสดงผลสามารถทำได้ด้วยตัวเลือก `-d` *`seconds`* เช่นกัน.
- “W” เขียนไฟล์ตั้งค่าเริ่มต้นในไฟล์. การปรับแต่งโปรแกรมต่างๆเช่น สี ฯลฯ จะเขียนเก็บไว้ในไฟล์ `~/.toprc` และใช้ได้ใหม่ครั้งหน้า.

- “R” เรียงลำดับจากน้อยไปหามาก. คำสั่ง top จะเรียงลำดับโปรเซสตามคอลัมน์ที่จัดลำดับอยู่จากค่ามากไปหาน้อยโดยปริยาย.
- “f” เพิ่ม/ลด คอลัมน์ข้อมูลที่ต้องการแสดง. หลังจากที่กดคีย์ “f” แล้วจะเปลี่ยนเป็นหน้าจอให้เลือกคอลัมน์ที่ต้องแสดง.
- “o” จัดลำดับคอลัมน์ที่ต้องการแสดงก่อนหลังตามต้องการ.
- “F” เลือกคอลัมน์ที่ต้องการให้เรียงลำดับรายการโปรเซส. เช่นถ้าคอลัมน์ %CPU เป็นคอลัมน์สำหรับเรียงลำดับโปรเซสก็จะเรียงลำดับโปรเซสที่มีเปอร์เซ็นต์การใช้หน่วยประมวลผลจากมากไปหาน้อยโดยปริยาย.
- “k” (kill) ส่งสัญญาณให้กับโปรเซสเหมือนกับคำสั่ง kill. โปรแกรม top จะถาม PID ของโปรเซสที่ต้องการส่งสัญญาณไปให้และถามสัญญาณที่ต้องการส่งซึ่งจะส่งสัญญาณ SIGTERM โดยปริยายถ้าไม่ระบุ.
- “r” (renice) ให้ผลเหมือนกับคำสั่ง renice ใช้เปลี่ยนค่า nice ของโปรเซสที่ต้องการ.

5.4.4 ทรัพยากรหน่วยความจำ

free อ้างอิงหน้า 400

ในลินุกซ์จะมีคำสั่ง free ใช้สำหรับแสดงการใช้งานหน่วยความจำโดยรวมของระบบ ซึ่งจะให้ผลคล้ายกับคำสั่ง top ในช่วงสรุปการใช้งานหน่วยความจำ. สิ่งที่แตกต่างจากคำสั่ง top คือจะแสดงจำนวนหน่วยความที่หักลบส่วนที่ใช้ไปกับ buffer และ cached แสดงในตัวอย่างต่อไปนี้.

ตัวอย่างที่ 5.19: คำสั่ง free แสดงการใช้งานหน่วยความจำในระบบ.

```
$ free↵
             total       used       free     shared    buffers     cached
Mem:        903544     655784     247760          0      22348     436352
-/+ buffers/cache:     197084     706460
Swap:       505848          0     505848
```

ถ้าเป็นการสั่งคำสั่งโดยไม่มีอาร์กิวเมนต์, คำสั่ง free จะแสดงจำนวนหน่วยความจำในหน่วย kB โดยปริยาย.

vmstat อ้างอิงหน้า 401

คำสั่งที่เกี่ยวกับการตรวจสอบการใช้งานหน่วยความจำอีกตัวได้แก่ vmstat. คำสั่ง vmstat คล้ายกับ free ที่รายงานการใช้งานหน่วยความจำโดยรวมแต่จะให้ข้อมูลอื่น ๆ ที่เกี่ยวข้องในระบบด้วย.

ตัวอย่างที่ 5.20: ใช้ vmstat ตรวจสอบการทำงานของระบบ.

```
$ vmstat 10 6
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
 2  0      0 239848  23404 437448    0    0    26    80 1101   853 11  1 87  1
 1  0      0 239920  23440 437580    0    0    16     4 1021  1485 10  2 86  2
```

```
0  1      0 230328  25452 438844    0    0   326     8 1071  1504 13  2 42 44
0  1      0 224952  28108 440980    0    0   476     9 1124  1055  5  2  0 94
0  1      0 220216  30396 441796    0    0   308    48 1069   851  3  0  0 97
0  1      0 216184  32752 441796    0    0   235    14 1064   787  3  1  0 96
```

รายละเอียดของคำสั่ง `vmstat` ได้แก่.

- procs
  ข้อมูลเกี่ยวกับโปรเซสในระบบว่ามีโปรเซสกี่ตัวที่อยู่ในรันคิว (r), โปรเซสที่ถูกบล็อก (b) หรืออยู่ในสภาพ sleep แบบรบกวนไม่ได้ (uninterruptible).

- memory
  ข้อมูลเกี่ยวกับหน่วยความจำเหมือนคำสั่ง `free` และ `top`.

- swap
  จำนวนหน่วยความจำที่ swap in (si) เข้ามาจากฮาร์ดดิสก์และ swap out (so) จากหน่วยความจำจริง.หน่วยเป็น kB ต่อวินาที.

- IO
  ความเร็วการเขียน (bo) อ่าน (bi) ข้อมูลหน่วยเป็น block ต่อวินาที่.

โดยปรกติ 1 block มีค่า 512 ไบต์.

- system
  จำนวน interrupt ต่อวินาทีที่เกิดขึ้น (in). จำนวน context switch (cs) ต่อวินาที.

- CPU
  เปอร์เซ็นการทำงานของหน่วยประมวลผลที่อยู่ใน user mode (us). เปอร์เซ็นต์การทำงานของหน่วยประมวลใน kernel mode (sys). เปอร์เซ็นต์ที่หน่วยประมวลผลว่าง (id) และเปอร์เซ็นต์ที่หน่วยประมวลผลใช้รอ IO (wa).

นอกจากคำสั่ง `vmstat` ยังมีคำสั่งรายงานสถิติต่าง ๆที่เกี่ยวกับระบบในทำนองเดียวกันได้แก่ `iostat` ใช้แสดงสถิติการถ่ายโอนข้อมูลของดิสก์และ `mpstat` ใช้แสดงสถิติเกี่ยวกับการใช้หน่วยประมวลผลซึ่งจะไม่กล่าวถึงในที่นี้.

5.5 จับเวลาการทำงานของโปรเซส

การจับเวลาการทำงานของโปรเซสเป็นเรื่องที่เกิดขึ้นบ่อยครั้งโดยเฉพาะเวลาที่ต้องการสำรวจเวลาที่ใช้ในการรันโปรแกรมที่สร้างขึ้นเอง. คำสั่งที่ใช้จับเวลาการทำงานของโปรเซสคือ `time`.

time อ้างอิงหน้า 404

ตัวอย่างที่ 5.21: จับเวลาการทำงานของคำสั่ง.

```
$ time cp aowthai-5.5.92-i386-cd1.iso /mnt/usb↵

real    0m45.845s
user    0m0.033s
sys     0m2.040s
```

เวลาที่ใช้ทำงานจริงตั้งแต่เริ่มทำงานจนจบ (real) ได้แก่ค่าที่แสดงในบรรทัดแรก. เวลาที่ใช้ไปจะช้าหรือเร็วขึ้นกับระบบตอนนั้นว่าทำงานหนักอยู่หรือไม่. ส่วนเวลาที่แสดงในบรรทัดที่สอง (user) บอกเวลาที่หน่วยประมวลผลใช้ไปกับโปรเซสใน user mode. และบรรทัดสุดท้าย (sys) แสดงเวลาที่หน่วยประมวลผลใช้ไปกับโปรเซสใน kernel mode. จากตัวอย่างข้างบนเป็นการก็อปปี้ไฟล์ใหญ่ ๆ. ในกรณีนี้ทำให้รู้ว่าหน่วยประมวลใช้เวลาไปกับระบบ (kernel) มากกว่าการทำงานของโปรเซสจริง ๆ (user).

5.6 ไฟล์และโปรเซส

โปรเซสจะมีความสัมพันธ์กับไฟล์หรือไดเรกทอรีเสมอ. เมื่อโปรเซสเริ่มกระทำการ, โปรเซสจะรับรู้สภาพแวดล้อมที่ทำงานอยู่เช่นไดเรกทอรีที่ทำงานอยู่ (current working directory). , ไฟล์ที่เปิดใช้อยู่ เป็นต้น.

ไดเรกทอรีที่โปรเซสทำงานอยู่นี้สามารถตรวจสอบได้จากไฟล์ `/proc/PID/cwd`. cwd ย่อมาจาก current working directory.

fuser อ้างอิงหน้า 401

5.6.1 หาโปรเซสที่ใช้ไฟล์

ถ้าต้องการตรวจไฟล์ฉบับหนึ่งว่ามีโปรเซสอะไรบ้างที่ใช้ไฟล์นั้นอยู่, ให้ใช้คำสั่ง `fuser` โดยระบุชื่อไฟล์.

ตัวอย่างที่ 5.22: ตรวจดูโปรเซสที่ใช้ไฟล์ `/bin/bash`.

```
$ fuser -uv /bin/bash↵

                     USER        PID ACCESS COMMAND
/bin/bash            poonlap    7820 ...e.  bash
                     poonlap    7826 ...e.  firefox
                     poonlap    7848 ...e.  thunderbird
                     poonlap    8213 ...e.  xdvi
```

ในกรณีนี้มีการใช้ตัวเลือก `-u` เพื่อให้คำสั่ง `fuser` (user) แสดงชื่อผู้ใช้, และตัวเลือก `-v` (verbose) เพื่อแสดงรายละเอียดความสัมพันธ์ระหว่างไฟล์กับโปรเซสที่ใช้ไฟล์นั้น. ในคอลัมน์ ACCESS เป็นรายละเอียดการใช้งานของโปรเซสต่อไฟล์ที่เกี่ยวข้องมีความหมายดังนี้.

- c: ไดเรกทอรที่ตรวจสอบเป็นไดเรกทอรีปัจจุบันของโปรเซสนั้น ๆ.
- e: ไฟล์นั้นกำลังถูกรันอยู่.
- f: ไฟล์ถูกเปิดใช้อยู่.
- r: รูทไดเรกทอรี.
- m: ไฟล์ถูกแม็ปในหน่วยความจำหรือเป็น shared library.

จากตัวอย่างทำให้เรารู้ว่าโปรเซสชื่อ bash, firefox, thunderbird และ xdvi กำลังรันไฟล์ `/bin/bash` อยู่. นี่เป็นวิธีหนึ่งที่สามารถตรวจเวลาสงสัยว่ามีโปรเซสอะไรบ้างใช้

ในกรณีนี้ `firefox`, `thunderbird` และ `xdvi` เป็นเชลล์สคริปต์, ไม่ใช่ไฟล์ไบนารี. ไฟล์ไบนารีที่แท้จริงได้แก่ `firefox-bin`, `thunderbird-bin` และ `xdvi.bin`.

ไฟล์ที่ระบุ. มีประโยชน์ใช้หาโปรเซสแปลกปลอมที่กำลังจะเข้าถึงไฟล์สำคัญ ๆในระบบได้.

บางครั้งเราต้องการจะ unmount พาร์ทิชันออกจากระบบไฟล์แต่ไม่สามารถ umount ได้และเกิด error ว่า "device is busy". สาเหตุที่ไม่สามารถ unmount อาจเกิดจากมีโปรเซสใช้ไฟล์ที่อยู่ในพาร์ทิชันนั้น, หรือไดเรกทอรีที่อยู่ในพาร์ทิชันนั้นเป็นไดเรกทอรีที่ทำงานอยู่ของโปรเซสบางตัว. ในกรณีนี้ก็ใช้คำสั่ง `fuser` ตรวจสอบกับตัวเลือก `-m` หมายถึงต้องการตรวจสอบทั้งระบบไฟล์, ไม่ใช้ไฟล์เดี่ยว ๆ.

ตัวอย่างที่ 5.23: หาโปรเซสที่ใช้ระบบไฟล์.

```
$ eject
umount: /mnt/cdrom: device is busy
umount: /mnt/cdrom: device is busy
eject: unmount of '/dev/cdrom' failed
$ fuser -muv /mnt/cdrom↵

                     USER        PID ACCESS COMMAND
/mnt/cdrom/          poonlap    7941 ..c..  bash
                     poonlap   27237 f.c.m  mpg321
```

ตัวอย่างข้างบนเป็นการใช้คำสั่ง `eject` เพื่อเอา CD ออกจากไดรว์ได้. คำสั่ง `eject` จะพยายาม unmount ก่อนแล้วดีด CD ออกมา. แต่ในกรณีนี้มีโปรเซสที่ใช้ไฟล์หรือไดเรกทอรี `/mnt/cdrom` อยู่จึงต้องใช้คำสั่ง `fuser` ตรวจสอบโปรเซส.

eject อ้างอิงหน้า 395

จากผลลัพธ์ของคำสั่งทำให้เรารู้ว่าไดเรกทอรี `/mnt/cdrom` ไม่สามารถ unmount ได้เพราะมีโปรเซส bash และ mpg321 ทำงานเกี่ยวข้องกับไดเรกทอรีนั้นอยู่. ถ้าต้องการ unmount ไดเรกทอรี `/mnt/cdrom` ต้องฆ่าโปรเซสที่เกี่ยวข้องการกับไดเรกทอรีนั้นก่อน, ซึ่งทำได้โดยการใช้คำสั่ง `kill` โดยให้โปรเซส ID ทีแสดงในผลลัพธ์ของคำสั่ง `fuser`. หรือถ้าไม่สนใจโปรเซสที่ใช้ไดเรกทอรี `/mnt/cdrom` เป็นโปรเซสอะไรแต่ต้องการจะฆ่าโปรเซสทุกโปรเซสที่ใช้หรือเกี่ยวข้องกับไดเรกทอรีนั้นอยู่, ให้ใช้ตัวเลือก `-k` ประกอบกับคำสั่ง `fuser`.

ตัวอย่างที่ 5.24: ใช้ `fuser` ฆ่าโปรเซสที่ใช้ไฟล์หรือไดเรกทอรีที่ระบุ.

```
$ fuser -k /mnt/cdrom/↵
/mnt/cdrom/:          7941c 27237c
kill 27237: No such process
No automatic removal. Please use  umount /mnt/cdrom
```

ในกรณีนี้คำสั่ง `fuser` จะฆ่าโปรเซส 7941 และ 27237 ตามลำดับ. แต่มี error ว่าหาโปรเซส 27237 ไม่เจอเพราะในความเป็นจริงแล้วโปรเซส 27237 เป็นโปรแกรม `mpg321` ที่รันอยู่ในเชลล์ (โปรเซส 7941) เมื่อโปรเซส 7941 ตายไปแล้วทำให้โปรเซส 27237 ตายตามไปด้วย.

โปรแกรม `mpg321` เป็นโปรแกรมคำสั่งสำหรับเล่นไฟล์ mp3, ogg ฯลฯ.

ตัวเลือก `-m` ไม่ได้ใช้สำหรับหาโปรเซสที่ใช้ไฟล์ใต้ไดเรกทอรีที่ระบุ, แต่เป็นการหาทั้งระบบไฟล์ของไฟล์ที่ระบุ. ตัวอย่างเช่นถ้าใช้ตัวเลือก `-m` และระบุไดเรกทอรีจะได้ผลดังต่อไปนี้.

ตัวอย่างที่ 5.25: ผลของตัวเลือก -m ต่อชื่อไดเรกทอรีหรือไฟล์.

```
$ fuser -muv /tmp↵

                     USER        PID ACCESS COMMAND
/tmp                 root          1 .rce.  init
                     root          2 .rc..  ksoftirqd/0
                     root          3 .rc..  events/0
                     root          4 .rc..  kblockd/0
--- แลดงผลต่อไปเรื่อยๆ ---
$ mount↵
/dev/hdb1 on / type ext3 (rw,noatime)
devfs on /dev type devfs (rw)
none on /proc type proc (rw)
none on /sys type sysfs (rw)
none on /dev/pts type devpts (rw)
/dev/hdb2 on /home type ext3 (rw,noatime)
/dev/hda3 on /mnt/hda type ext3 (rw,noatime)
none on /dev/shm type tmpfs (rw)
none on /proc/bus/usb type usbfs (rw)
/dev/cdrom on /mnt/cdrom type iso9660 (ro,noexec,nosuid,nodev,user=poonlap)
```

คำสั่ง `fuser` จะแสดงโปรเซสที่จริงๆแล้วไม่ได้ใช้ไฟล์ที่อยู่ใต้ไดเรกทอรี `tmp` เพราะตัวเลือก `-m` หมายถึงระบบไฟล์ (พาร์ทิชัน). ในกรณีนี้, ไดเรกทอรี `tmp` อยู่ในพาร์ทิชัน `/dev/hdb1`, ดังนั้นคำสั่ง `fuser` จึงแสดงโปรเซสทั้งหมดที่เกี่ยวข้องกับพาร์ทิชันนั้น.

lsof อ้างอิงหน้า ??

คำสั่งที่คล้ายกับ `fuser` แต่สามารถแสดงรายละเอียดได้มากกว่าได้แก่คำสั่ง `lsof`. ถ้าต้องการหาว่ามีโปรเซสอะไรบ้างที่ใช้ไฟล์ใต้ไดเรกทอรี `/tmp` รวมถึงไดเรกทอรีย่อย, จะใช้ตัวเลือก `+D`.

ตัวอย่างที่ 5.26: ใช้ `lsof` หาโปรเซสที่ใช้ไฟล์ในไดเรกทอรีที่ระบุ.

```
# lsof +D /tmp↵
COMMAND     PID     USER   FD   TYPE     DEVICE SIZE   NODE NAME
cannaserv  5914     root    0u  unix 0xecc81380        9957 /tmp/.iroha_unix/IROHA
cannaserv  5914     root    3u  unix 0xf7311380        6687 /tmp/.iroha_unix/IROHA
famd       7216 poonlap    4u  unix 0xf254e200        9279 /tmp/.fampnWIDy
famd       7216 poonlap    5u  unix 0xf0e58800        9437 /tmp/.famL21ZzK
--- แลดงผลต่อไปเรื่อยๆ ---
```

คำสั่ง `lsof` จะแสดงรายละเอียดต่างๆในแต่ละคอลัมน์ได้แก่ ชื่อคำสั่ง (COMMAND), โปรเซส ID (PID), ชื่อผู้ใช้ (USER), file descriptor (FD), ประเภทของไฟล์ (TYPE), ประเภทดีไวซ์ (DEVICE), ขนาด (SIZE), เลข i-node (NODE) และชื่อไฟล์ (NAME).

คอลัมน์ที่น่าสนใจที่ควรรู้จักในที่นี้ได้แก่คอลัมน์ FD ซึ่งจะบอกรายเอียดเกี่ยวกับ file descriptor แบ่งเป็น

- **cwd** (current working directory) ไดเรกทอรีที่โปรเซสทำงานอยู่.
- **ltx** (library text) ข้อมูลของ shared library ได้แก่ส่วนที่เป็น code และ data.
- **mem** (memory-mapped file) ไฟล์ที่แม็ปไว้ในหน่วยความจำ.

- **mmap** (memory-mapped device) ไฟล์ดีไวส์ที่แม็ปไว้ในหน่วยความจำ.
- **pd** (parent directory) ไดเรกทอรีพ่อแม่.
- **rtd** (root directory) ไดเรกทอรีรูท.
- **txt** (program text) ข้อมูลของโปรแกรมในส่วนของ code และ data.
- ***N*** ตัวเลขเฉพาะของ file descriptor. ตามด้วยอักษรต่อไปนี้บ่งบอกถึงการเข้าถึงของไฟล์นั้นได้แก่
  - **r** สิทธิ์การอ่าน
  - **w** สิทธิ์การเขียน
  - **u** สิทธิ์การอ่านและเขียน

คอลัมน์ TYPE แสดงประเภทของไฟล์เช่น

- **DIR** (directory) ไดเรกทอรี
- **REG** (regular file) ไฟล์ธรรมดา
- **CHR** (character device) ไฟล์ดีไวซ์แบบ character.
- **IPv4** (IPv4 socket) เน็ตเวิร์ก socket สื่อสารประเภท IP version 4.
- **FIFO** (first in first out) ไปป์
- **unix** (Unix domain socket) ไฟล์ socket

5.6.2 หาไฟล์ที่โปรเซสใช้

ข้อมูลเกี่ยวกับโปรเซสต่างๆจะอยู่ในไดเรกทอรีชื่อเป็นโปรเซส ID ใต้ไดเรกทอรี `proc`. เราสามารถใช้คำสั่ง `cat` ดูว่าโปรเซสเหล่านั้นกำลังใช้ไฟล์อะไรอยู่ได้. แต่วิธีนี้ไม่สะดวกมากนักเพราะเป็นการดูข้อมูลดิบโดยตรง. คำสั่งที่อำนวยความสะดวกใช้ตรวจสอบดูโปรเซสว่ากำลังใช้ไฟล์อะไรอยู่บ้างได้แก่ `lsof` แสดงไฟล์ทั้งหมดที่เปิดใช้อยู่และโปรเซสที่เกี่ยวข้องกับไฟล์นั้น.

`lsof` อ้างอิงหน้า ??

ตัวอย่างต่อไปนี้เป็นการใช้คำสั่ง `lsof` ตรวจสอบดูว่าโปรเซส xdvi.bin กำลังเปิดใช้ไฟล์อะไรบ้าง. ตัวเลือก `-c` ใช้สำหรับเลือกโปรเซสที่ขึ้นต้นด้วยอักษรที่กำหนดเป็นอาร์กิวเมนต์ของตัวเลือก.

ตัวอย่างที่ 5.27: ดูไฟล์ที่เปิดใช้โดยโปรเซส xdvi.bin ทั้งหมด.

```
$ /usr/sbin/lsof -c xdvi.bin↵
COMMAND   PID    USER   FD   TYPE     DEVICE    SIZE    NODE NAME
xdvi.bin 8677 poonlap  cwd    DIR       3,66    4096  587529 /home/poonlap/BOOK/
linuxbook
xdvi.bin 8677 poonlap  rtd    DIR       3,65    4096       2 /
```

```
xdvi.bin 8677 poonlap  txt    REG       3,65  344180  725577 /usr/bin/xdvi.bin
xdvi.bin 8677 poonlap  mem    REG       3,65   90796 1313292 /lib/ld-2.3.4.so
xdvi.bin 8677 poonlap  mem    REG       3,65   10572  823279 /usr/X11R6/lib/X11/
locale/lib/common/xlcDef.so.2
xdvi.bin 8677 poonlap  mem    REG       3,65   15480  724869 /usr/lib/libwwwsql.
so.0.1.0
xdvi.bin 8677 poonlap  mem    REG       3,65    8472  724849 /usr/lib/libwwwinit
.so.0.1.0
--- แลดงผลต่อไปเรื่อยๆ ---
xdvi.bin 8677 poonlap  mem    REG       3,65   40759  826417 /usr/X11R6/lib/libX
cursor.so.1.0
xdvi.bin 8677 poonlap  mem    REG       3,65   32191  826396 /usr/X11R6/lib/libX
render.so.1.2
xdvi.bin 8677 poonlap    0u   CHR      136,0               2 /dev/pts/0
xdvi.bin 8677 poonlap    1u   CHR      136,0               2 /dev/pts/0
xdvi.bin 8677 poonlap    2w   CHR        1,3               6 /dev/null
xdvi.bin 8677 poonlap    3u  unix 0xe8cdf980           10718 socket
xdvi.bin 8677 poonlap    4r   REG       3,66    2361  575351 /home/poonlap/.mail
cap
xdvi.bin 8677 poonlap    5r   REG       3,66 1964924  604123 /home/poonlap/BOOK/
linuxbook/linux.dvi
xdvi.bin 8677 poonlap    6r   REG       3,65   14312 1204286 /var/cache/fonts/pk
/ljfour/jknappen/tc/tctt1440.600pk
xdvi.bin 8677 poonlap    7r   REG       3,65    7572  923858 /var/cache/fonts/pk
/ljfour/jknappen/tc/tctt0800.600pk
xdvi.bin 8677 poonlap    8r   REG       3,65   11524  923854 /var/cache/fonts/pk
/ljfour/jknappen/tc/tctt1200.600pk
xdvi.bin 8677 poonlap    9r   REG       3,65   14220  923849 /var/cache/fonts/pk
/ljfour/jknappen/tc/tcrm1200.600pk
xdvi.bin 8677 poonlap   11w  FIFO        0,7           10725 pipe
xdvi.bin 8677 poonlap   12r  FIFO        0,7           10726 pipe
```

จะเห็นว่าโปรเซส ๆหนึ่งเปิดใช้ไฟล์หลายไฟล์ในเวลาเดียวกัน, ไดเรกทอรีที่ทำงานอยู่, ไฟล์โปรแกรมของโปรเซส, ไปป์, ไฟล์ไลบรารีที่แม็ปไว้ในหน่วยความจำ ฯลฯ.

คำสั่ง `lsof` มีประโยชน์อย่างยิ่งต่อการดูแลระบบ, สามารถหาโปรเซสที่กำลังเปิดใช้ไฟล์ได้และหาไฟล์ที่ใช้โดยโปรเซสได้เช่นกัน. ข้อมูลเหล่านี้บางทีสามารถบอกได้ว่าโปรเซสนั้น ๆเป็นโปรเซสแปลกปลอมหรือไม่. ตัวเลือกที่น่าสนใจอื่น ๆเช่น `-u` ***user*** สามารถใช้ดูไฟล์และโปรเซสที่รันอยู่โดยผู้ใช้ที่ระบุได้. คำสั่ง `lsof` เป็นคำสั่งที่ค่อนข้างซับซ้อนอีกคำสั่งและมีตัวเลือกหลายตัว. สำหรับรายละเอียดของตัวเลือกให้อ่านจาก `man lsof`.

5.7 ดูการทำงานของโปรเซส

การดูการทำงานของโปรเซสทำได้หลายวิธีเช่น ดูรหัสต้นฉบับของซอฟต์แวร์นั้น ๆ, ถ้าเป็นโปรแกรมที่คอมไพล์เองก็อาจจะใช้ตัวเลือกเวลาคอมไพล์โปรแกรมให้ debug ดูการทำงานได้ภายหลัง. วิธีการเหล่านี้มีข้อจำกัดคือต้องมีต้นฉบับของซอฟต์แวร์จึงจะใช้วิธีนี้ได้.

ทางเลือกอีกทางเวลาต้องการดูการทำงานของโปรเซสแต่ไม่มีรหัสต้นฉบับให้ศึกษา, อาจทำได้โดยการดูว่าโปรเซสนั้น ๆเรียกใช้ซิสเต็มคอลล์อะไร. เราเรียกวิธีนี้ว่าการ trace จะทำให้เรารู้คร่าว ๆว่าโปรเซสนั้นทำงานอย่างไร.

คำสั่งที่ใช้ดูซิสเต็มคอลล์ที่โปรเซสเรียกใช้คือ `strace`. คำสั่ง `strace` จะแสดงซิสเต็มคอลล์ต่าง ๆที่โปรเซสเรียกใช้และสัญญาณที่โปรเซสได้รับ. โปรเซสที่ต้องการ trace เป็นได้ทั้งโปรเซสที่รันอยู่แล้ว, หรือกำลังจะรันจากบรรทัดคำสั่งก็ได้. ตัวอย่างต่อไปนี้จะแสดงการ trace การทำงานของโปรแกรม `cat` ซึ่งมีอาร์กิวเมนต์เป็นไฟล์ชื่อ `hello.txt`.

strace อ้างอิงหน้า 403

ตัวอย่างที่ 5.28: การใช้ `strace` ดูการทำงานของโปรเซส.

```
$ cat hello.txt↵
hello
$ strace cat hello.txt↵
execve("/bin/cat", ["cat", "hello.txt"], [/* 57 vars */]) = 0
uname(sys="Linux", node="toybox", ...) = 0
brk(0)                                  = 0x804d000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, st_mode=S_IFREG|0644, st_size=150692, ...) = 0
mmap2(NULL, 150692, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40017000
close(3)                                = 0
open("/lib/tls/libc.so.6", O_RDONLY)    = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\275P\1"..., 512) = 512
fstat64(3, st_mode=S_IFREG|0755, st_size=1174064, ...) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40
03c000
mmap2(NULL, 1105132, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x4
003d000
mmap2(0x40145000, 16384, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRI
TE, 3, 0x107) = 0x40145000
mmap2(0x40149000, 7404, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOU
S, -1, 0) = 0x40149000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40
14b000
mprotect(0x40145000, 4096, PROT_READ)   = 0
mprotect(0x40015000, 4096, PROT_READ)   = 0
set_thread_area({entry_number:-1 -> 6, base_addr:0x4014b6c0, limit:1048575, seg_
32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, usea
ble:1}) = 0
munmap(0x40017000, 150692)              = 0
open("/dev/urandom", O_RDONLY)          = 3
read(3, "\214;^\222", 4)                = 4
close(3)                                = 0
brk(0)                                  = 0x804d000
brk(0x806e000)                          = 0x806e000
fstat64(1, st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...) = 0
open("hello.txt", O_RDONLY|O_LARGEFILE) = 3
fstat64(3, st_mode=S_IFREG|0644, st_size=6, ...) = 0
read(3, "hello\n", 4096)                = 6
write(1, "hello\n", 6hello           ← ``hello'' แสดงทาง stdout แทรกใน stderr
)                 = 6
```

```
read(3, "", 4096)                       = 0
close(3)                                = 0
close(1)                                = 0
exit_group(0)                           = ?
```

จากผลลัพธ์ของคำสั่ง strace ทำให้เรารู้ว่าคำสั่ง cat hello.txt ซึ่งใช้อ่านเนื้อหาที่อยู่ในไฟล์ hello.txt และแสดงผลออกทางหน้าจอเรียกใช้ซิสเต็มคอลล์ถึง 35 อย่าง. ผลลัพธ์ของคำสั่งจะแสดงทาง stderr จึงไม่สามารถไปป์หรือรีไดเรกบันทึกในไฟล์ได้. ถ้าต้องการเก็บผลลัพธ์ของคำสั่งลงในไฟล์, ให้ใช้ตัวเลือก -o *file*.

ข้อมูลที่แสดงในแต่ละบรรทัดได้แก่ซิสเต็มคอลล์ซึ่งก็คือฟังก์ชันในภาษา C, อาร์กิวเมนต์ของฟังก์ชัน, และค่าตอบกลับของฟังก์ชันนั้น. บรรทัดแรกได้แก่ซิสเต็มคอลล์ execve. ถ้าต้องการรู้ความหมายของซิสเต็มคอลล์นี้ก็ให้อ่าน man 2 execve ก็จะรู้เป็นการรันโปรแกรม /bin/cat มีอาร์กิวเมนต์เป็นสายอักขระได้แก่ cat และ hello.txt. และมีตัวแปรสภาพแวดล้อมอีก 57 ตัวซึ่งไม่ได้แสดงในที่นี้เป็นอาร์กิวเมนต์ของซิสเต็มคอลล์. ตัวเลขที่อยู่หลังเครื่องหมาย = คือค่าตอบกลับของซิสเต็มคอลล์ในกรณีนี้มีค่าเป็น 0.

ซิสเต็มคอลล์ execve

การดูซิสเต็มยังทำให้รู้อีกว่าโปรเซสนั้นพยายามจะเข้าถึงไฟล์อะไร, อย่างไร, และเกิดอะไรขึ้นได้ด้วย. เช่นซิสเต็มคอลล์ access ซึ่งเป็นการตรวจสอบสิทธิ์การใช้ไฟล์ของไฟล์ /etc/ld.so.preload แต่มีค่าตอบกลับเป็น -1 มีรหัสเป็น ENOENT หมายถึงเกิดข้อผิดพลาดและมีรายละเอียดบอกให้รู้ว่า “No such file or directory”. ทำให้รู้ว่าโปรเซสพยายามจะตรวจสอบไฟล์นี้แต่หาไม่เจอ. บรรทัดถัดมาเป็นซิสเต็มที่ใช้บ่อยคือ open. โปรเซสพยายามจะเปิดไฟล์ /etc/ld.so.cache แบบ O_RDONLY คืออ่านอย่างเดียว. ค่าตอบกลับของซิสเต็มคอลล์จะเป็น file descriptor ซึ่งในกรณีนี้มีค่าเป็น 3. อย่าลืมว่าโปรเซสทุกตัวจะมี file descriptor 0 (stdin), 1 (stdout), และ 2 (stderr) โดยปริยาย. ดังนั้น file descriptor ที่ได้จากการเปิดไฟล์ใหม่จึงเป็น 3.

ซิสเต็มคอลล์ access, open

ต่อจากนั้นใช้ซิสเต็มคอลล์ fstat64 ดึงข้อมูลเกี่ยวกับไฟล์ที่เปิด. เรียกใช้ซิสเต็มคอลล์ mmap2 เพื่อแม็ปข้อมูลจากไฟล์เข้าสู่หน่วยความจำ. และปิดไฟล์ที่เปิดไว้ด้วยซิสเต็มคอลล์ close. เปิดไฟล์อื่นๆและกระทำการต่อไปเรื่อยๆ.

ซิสเต็มคอลล์ fstat, mmap2, close

คำสั่ง cat ในตัวอย่างจะเปิดไฟล์ hello.txt หลังจากที่จัดการไฟล์ไลบรารีต่างๆ เรียบร้อยแล้ว. ซิสเต็มคอลล์ที่ใช้เปิดไฟล์ไฟล์คือ open และซิสเต็มคอลล์ที่ใช้อ่านไฟล์คือ read. หลังจากนั้นเขียนข้อมูลออกทาง stdout ซึ่งมี file descriptor เป็น 1 ด้วยซิสเต็มคอลล์ write. และสุดท้ายปิดไฟล์ด้วย close และจบการทำงาน.

ซิสเต็มคอลล์ open, read, write, close

สำหรับโปรเซสที่ทำงานอยู่แล้วและต้องการ trace โปรเซสนั้นให้ใช้ตัวเลือก -p *pid*. บางครั้งโปรแกรมบางตัวทำงานอยู่แต่ดูเหมือนไม่ทำงานหรือทำงานผิดปรกติ, เราสามารถ trace ด้วยคำสั่ง strace ดูว่าโปรเซสทำงานอยู่จริงหรือไม่. หรือถ้าไม่ทำงาน, โปรเซสนั้นไปค้างอยู่ตรงไหนสามารถรู้ได้จากผลลัพธ์ของคำสั่ง. โปรเซสบางตัวอาจจะ fork โปรเซสลูก. ในกรณีนี้คำสั่ง strace จะไม่ trace ดูไปถึงโปรเซสลูกว่าทำอะไรบ้างต้องใช้ตัวเลือก -f มิฉะนั้นคำสั่งจะ trace เฉพาะโปรเซสแม่ที่ดูเท่านั้น.

ตัวเลือกที่มีประโยชน์อีกตัวได้แก่ -e ใช้เลือกดูซิสเต็มคอลล์หรือเหตุการณ์ที่ต้องการ

ดู. อาร์กิวเมนต์ของตัวเลือก `-e` เป็นได้ทั้งชื่อซิสเต็มคอลล์ที่ต้องการจับตาดูหรือชื่อเหตุการณ์ที่เตรียมไว้อยู่แล้วในตารางที่ 5.2.

ตารางที่ 5.2: ตัวเลือกของ `strace` สำหรับเลือกเหตุการณ์ที่ต้องการ.

ตัวเลือก	คำอธิบาย
`-e trace=file`	สำรวจซิสเต็มคอลล์ที่เกี่ยวกับไฟล์เช่น `open`, `stat`, `chmod` ฯลฯ.
`-e trace=process`	สำรวจซิสเต็มคอลล์ที่เกี่ยวกับการควบคุมโปรเซสเช่น `fork`, `wait`, `exec` ฯลฯ.
`-e trace=network`	สำรวจซิสเต็มคอลล์ที่เกี่ยวกับเน็ตเวิร์ก.
`-e trace=signal`	สำรวจซิสเต็มคอลล์ที่เกี่ยวกับสัญญาณ.
`-e trace=ipc`	สำรวจซิสเต็มคอลล์ที่เกี่ยวกับ IPC (Inter Process Communication)
`-e trace=`*`syscall`*	เลือกดูซิสเต็มคอลล์ที่ระบุ. เช่นถ้าต้องการดูว่าโปรเซสเปิดไฟล์อะไรบ้างให้ใช้ `-e trace=open`.

การใช้คำสั่ง `strace` นอกจากจะเป็นดูการทำงานของโปรเซสโดยอาศัยการดูซิสเต็มคอลล์ที่โปรเซสใช้แล้วยังมีประโยชน์ในการ debug โปรแกรมในกรณีที่เกิด error โดยไม่รู้สาเหตุอีกด้วย.

5.8 สรุปท้ายบท

- โปรเซสคือโปรแกรมที่โหลดเข้าสู่หน่วยความจำเพื่อกระทำการโดยที่เคอร์เนลเป็นตัวควบคุมโปรเซสในระบบ.
- โปรเซสแต่ละตัวจะมีพื้นที่หน่วยความจำเสมือนของตัวเอง. ที่อยู่ของหน่วยความจำเสมือนจะถูกแปลงไปเป็นที่อยู่หน่วยความจำจริงเมื่อใช้. ที่อยู่หน่วยความจำเสมือนไม่จำเป็นต้องอยู่ในหน่วยความจำจริงทุกส่วน. ที่อยู่ของหน่วยความจำเสมือนบางส่วยอาจจะอยู่ในพื้นที่ swap ในหน่วยความจำถาวร.
- สัญญาณเป็นวิธีการสื่อสารระหว่างโปรเซส. เมื่อโปรเซสได้รับสัญญาณหนึ่ง ๆจะมีการตอบสนองต่อสัญญาณที่ได้รับ. บางสัญญาณที่ได้รับอาจปฏิบัติเพิกเฉยการตอบสนองได้โดยการสร้าง signal trap. สัญญาณที่ไม่สามารถเพิกเฉยได้แก่ SIGSTOP, SIGKILL.
- คำสั่งสำคัญที่เกี่ยวกับโปรเซสได้แก่ `ps`, `kill`, `top`
- คำสั่งที่ใช้ตรวจสอบความสัมพันธ์ระหว่างโปรเซสและไฟล์ที่โปรเซสใช้ได้แก่ `fuser` และ `lsof`.

- คำสั่ง `strace` ใช้สำหรับดูซิสเต็มคอลล์ที่โปรเซสเรียกใช้. การดูซิสเต็มคอลล์ที่โปรเซสใช้ช่วยให้เข้าใจการทำงานของโปรเซสได้คร่าว ๆ. สามารถใช้ในการแก้ปัญหาการทำงานของโปรเซสได้ด้วยเช่นการ debug โปรแกรมเมื่อเกิดข้อผิดพลาดหรือโปรเซสทำงานผิดปรกติ.

บทที่ 6
ระบบ X วินโดว์

ยูสเซอร์อินเทอร์เฟสแบบกราฟิกหรือที่เรียกกันว่า GUI (Graphical User Interface) มีบทบาทอย่างยิ่งต่อผู้ใช้คอมพิวเตอร์ในปัจจุบัน. ระบบ GUI อำนวยความสะดวกต่าง ๆ ด้วยการแสดงผลนอกเหนือจากอักษรได้แก่รูปภาพสองมิติหรือสามมิติ, แสดงโปรแกรมต่าง ๆด้วยระบบหน้าต่าง (window) และโต้ตอบกับผู้ใช้โดยใช้อุปกรณ์นอกเหนือจากแป้นพิมพ์ซึ่งได้แก่เมาส์. เมื่อเทียบกับระบบปฏิบัติการยูนิกซ์ในยุคแรก, ระบบปฏิบัติการยูนิกซ์จะใช้แค่แป้นพิมพ์และหน้าจอมีคุณสมบัติแสดงตัวอักษรได้เป็นอินเทอร์เฟสติดต่อกับผู้ใช้ผ่านทางเชลล์. ระบบปฏิบัติการยูนิกซ์รวมถึงลินุกซ์ไม่มีระบบการแสดงผลแบบกราฟิกเป็นอินเทอร์เฟสมาตรฐาน. อินเทอร์เฟสมาตรฐานระหว่างผู้ใช้กับเคอร์เนลคือเชลล์. ระบบ GUI จะเป็นส่วนเพิ่มเติมของระบบในรูปของโปรแกรมซึ่งไม่ได้รวมอยู่ในเคอร์เนล. จุดนี้จะต่างกับระบบปฏิบัติการวินโดวส์ (Microsoft Windows) ซึ่งมีระบบ GUI รวมอยู่ในตัวระบบปฏิบัติการ.

ระบบแสดงผลแบบกราฟิกที่เป็นที่ยอมรับและใช้กับระบบปฏิบัติยูนิกซ์มานานได้แก่ระบบ X วินโดว์ (X Window system). ในบทนี้จะอธิบายเกี่ยวกับระบบ X วินโดว์และนำเสนอการใช้งานแอพพลิเคชัน (application) ต่าง ๆ.

คำว่า Window ไม่มี s ต่อท้าย.

6.1 ประวัติระบบ X วินโดว์

ระบบ X วินโดว์เริ่มสร้างและพัฒนาที่ Massachusetts Institute of Tecnology (MIT) ในปีค.ศ. 1984 ในโครงการที่เรียกว่า Athena. ระบบ X วินโดว์รับแนวคิดต้นแบบจากระบบหน้าต่างที่ชื่อว่า “W” ที่พัฒนาโดยมหาวิทยาลัย Stanford. ดังนั้นชื่อตัวอักษร “X” ยังมีความหมายถึงระบบวินโดว์ที่สร้างต่อมาจากระบบวินโดว์ “W” ด้วย. โครงการพัฒนาระบบ X วินโดว์นี้ได้รับทุนวิจัยสนับสนุนจากบริษัทเอกชนได้แก่ Digital Wquipment Corporation (DEC) และ IBM.

ในปีค.ศ. 1985 ทางโครงการเปิดตัวระบบ X วินโดว์รุ่นที่ 10 สู่สาธารณะ, และในปีถัดไปทาง DEC ได้นำระบบ X วินโดว์ไปใช้ในเครื่องคอมพิวเตอร์ VAXstation II/GPX เชิงพาณิชย์ [42]. หลังจากนั้นบริษัทผลิตคอมพิวเตอร์ตระกูลยูนิกซ์ทั้งหลายเช่น Hewlett-Packard, Sun, IBM ได้นำระบบ X วินโดว์ไปใช้กับผลิตภัณฑ์ของตัวเองอย่างแพร่หลาย.

ด้วยเหตุนี้เองระบบ X วินโดว์จึงเป็นเสมือนระบบกราฟิกมาตรฐานสำหรับยูนิกซ์ในเวลาต่อมา. ในปีค.ศ. 1987 ระบบ X วินโดว์ออกรุ่น 11 ซึ่งรู้จักกันในชื่อ X11. จากนั้นบริษัทคอมพิวเตอร์ต่าง ๆเห็นความสำคัญของระบบ X วินโดว์นี้จึงได้ร่วมกันจัดตั้ง X consortium เป็นองค์กรกลางที่ได้รับการสนับสนุนจากบริษัทผู้ผลิตคอมพิวเตอร์ต่าง ๆร่วมกับ MIT. แต่หลังจากนั้น X consortium แยกตัวออกมาจาก MIT เป็นองค์กรอิสระควบคุมมาตรฐานที่เกี่ยวข้องกับระบบ X วินโดว์. ในขณะเดียวกันมีการออกรุ่นย่อย ๆของซอฟต์แวร์ X11 เรียกว่า release และรู้จักกันในชื่อ X11R2, X11R3 ไปเรื่อย ๆ. X11R6 เป็นรุ่นที่เสถียรมากและมีคุณสมบัติต่าง ๆที่สมบูรณ์และเป็นที่นิยมใช้กัน. ปัจจุบันองค์กรที่ควบคุมมาตรฐานของระบบ X วินโดว์ได้แก่ X.org foundation ซึ่งเป็นหน่วยงานย่อยของ The Open Group อีกทีหนึ่ง. และรุ่นของระบบ X วินโดว์ล่าสุดได้แก่ X11R6.8.1 ชื่อเรียกเป็นทางการคือ X Window System Version 11 Release 6.8.1.

ข้อมูลสำรวจเดือนธันวาคม ค.ศ. 2004

6.1.1 XFree86

ซอฟต์แวร์ระบบ X วินโดว์ที่ใช้กันในลินุกซ์ตั้งแต่เริ่มแรกได้แก่ซอฟต์แวร์จากโครงการ XFree86. โครงการนี้มีจุดมุ่งหมายสร้างระบบ X วินโดว์ที่สามารถจากจ่ายไปโดยเสรีและเป็นแบบโอเพนซอร์ส. ซอฟต์แวร์ของระบบ X วินโดว์ดังกล่าวพยายามที่จะคลอบคลุมฮาร์ดแวร์และซอฟต์แวร์ต่าง ๆที่มีอยู่, รวมถึงสถาปัตยกรรมคอมพิวเตอร์ต่าง ๆที่มีอยู่หลากหลายด้วย. ดังนั้นซอฟต์แวร์ระบบ X วินโดว์สามารถใช้ทั้งบนระบบปฏิบัติการลินุกซ์, ไมโครซอฟต์วินโดวส์ ฯลฯ, รองรับกราฟฟิกการ์ดต่าง ๆ และใช้ได้กับเครื่องเวิร์กสเตชันของบริษัทคอมพิวเตอร์แบบยูนิกซ์ต่าง ๆด้วย. ซอฟต์แวร์ระบบ X วินโดว์ที่เผยแพร่โดยโครงการ XFree86 รุ่นล่าสุดได้แก่รุ่น 4.4.0 ออกเมื่อเดือนกุมภาพันธ์ค.ศ. 2004.

ข้อมูลสำรวจเดือนธันวาคม ค.ศ. 2004.

ความนิยมของดิสทริบิวชันทั่วไปเริ่มหันไปใช้ซอฟต์แวร์ระบบ X วินโดว์ที่สร้างโดย X.org เพิ่มขึ้นแทน XFree86 แล้วเนื่องจาก XFree86 ประกาศเปลี่ยนหนังสืออนุญาตการใช้งานต้นปีค.ศ. 2004.

6.2 พื้นความรู้ระบบ X วินโดว์

ระบบ X วินโดว์เป็นระบบประกอบด้วยซอฟต์แวร์ที่ทำงานแบบ*ไคล์เอ็นต์เซิร์ฟเวอร์ (client server)*. เราสามารถแยกการทำงานของระบบได้เป็น 2 ส่วนคือโปรแกรมเซิร์ฟเวอร์และโปรแกรมไคลเอ็นต์.

โปรแกรมเซิร์ฟเวอร์ที่เรียกว่า *X เซิร์ฟเวอร์ (X server)* เป็นโปรเซสในระบบทำหน้าที่ต่อไปนี้

- รอรับคำขอ (request) จากไคลเอ็นต์และปฏิบัติการแสดงผลขั้นพื้นฐานเท่านั้นเช่น วาดเส้นตรง, เขียนอักษรบนหน้าจอ ฯลฯ.
- สร้างหน้าต่าง, จัดตำแหน่งหน้าต่าง, ลบหน้าต่าง.
- จัดการ*อีเวนท์ (event)* ต่าง ๆจากแป้นพิมพ์และเมาส์และส่งข้อมูลเกี่ยวกับอีเวนท์นั้นให้ไคลเอ็นต์รับรู้.

event ▶ *อีเวนท์*. เหตุการณ์ต่าง ๆที่เกิดขึ้น. เป็นคำศัพท์ที่ใช้ในระบบ GUI. ตัวอย่างเช่นการคลิ๊กเมาส์ทำให้เกิดอีเวนท์ (เหตุการณ์) การคลิ๊ก. ตัว X เซิร์ฟเวอร์จะส่งข้อมูลเกี่ยวกับเหตุการณ์นั้นไปให้ไคลเอ็นต์.

โปรแกรมไคล์เอ็นต์ซึ่งหมายถึงแอพพลิเคชันต่าง ๆที่แสดงผลแบบ GUI จะเป็นโปรเซสที่ติดต่อกับเซิร์ฟเวอร์ผ่านทางเน็ตเวิร์กหรือยูนิกซ์ซ็อกเก็ตโดยใช้ข้อตกลงที่เรียกว่า*เอ็กซ์โปรโตคอล (X protocol)*. ไคลเอ็นต์จะทำหน้าที่สำคัญ ๆต่อไปนี้ได้แก่

- ส่งคำร้องขอให้แสดงหรือวาดรูปทรงต่าง ๆทางหน้าจอ.
- รับข้อมูลอีเวนท์ที่เกิดขึ้นจากเซิร์ฟเวอร์และตัดสินใจกระทำการต่าง ๆ.
- ทำงานต่าง ๆเฉพาะโปรแกรม. เช่นถ้าเป็นโปรแกรมเบราว์เซอร์ก็จะติดใช้เน็ตเวิร์กดึงข้อมูลมาแสดงผลเป็นต้น. การใช้งานเน็ตเวิร์กในลักษณะนี้ไม่เป็นการทำงานเฉพาะโปรแกรมไม่เกี่ยวกับ X วินโดว์.

เนื่องจากโปรเซสที่จัดการเรื่องกราฟิกแสดงผลรวมถึงการรับอีเวนท์ที่เกิดขึ้นจากอุปกรณ์ต่าง ๆคือเซิร์ฟเวอร์, ดังนันจึงเป็นข้อดีต่อไคลเอ็นต์ที่ไม่จำเป็นต้องรู้เรื่องเกี่ยวกับฮาร์ดแวร์ที่แสดงผลเลย. สิ่งที่ไคลเอ็นต์ต้องรับรู้คือวิธีที่จะติดต่อกับเซิร์ฟเวอร์. โปรแกรมไคลเอ็นต์ต่าง ๆจะใช้ไลบรารีพื้นฐานได้แก่ *Xlib (X library)*. ตัวอย่างไฟล์ไลบรารีเช่น `/usr/X11R6/lib/libX11.a`. โปรแกรมแบบ GUI จะใช้ไลบรารีนี้เสมอ. และเนื่องจากการติดต่อระหว่างไคลเอ็นต์และเซิร์ฟเวอร์อาศัยเน็ตเวิร์กทำให้สามารถแสดงผลข้ามเครื่องผ่านทางเน็ตเวิร์กได้. เช่นรันแอพพริเคชันที่เครื่อง ก. แต่สั่งให้แสดงผลบนหน้าจอของคอมพิวเตอร์เครื่อง ข. ผ่านทางเน็ตเวิร์กได้ถ้าอนุญาตให้แสดงผล.

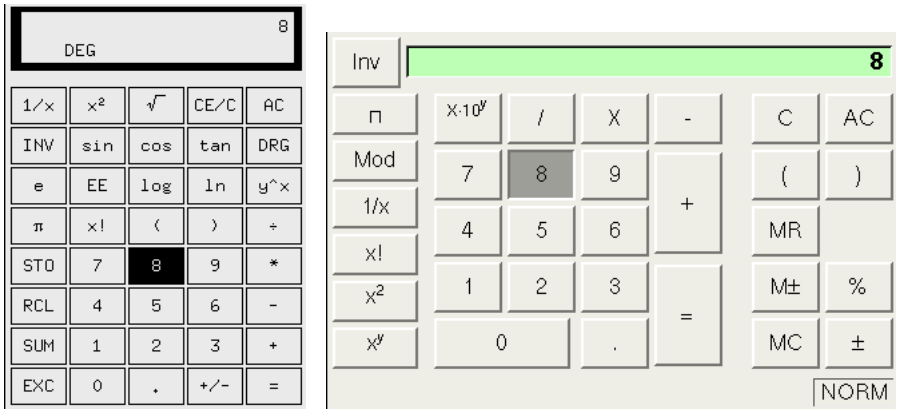
6.2.1 ทูลล์คิทและไลบรารี

การเขียนโปรแกรมภาษาซีแบบ GUI จะฟังก์ชันที่เตรียมไว้สำหรับไลบรารี X โดยตรงก็ได้แต่ไม่สะดวกนักเพราะฟังก์ชันเหล่านี้อำนวยความสะดวกขั้นพื้นฐานเท่านั้นเช่นความสามารถการวาดเส้นตรง, วงกลม, เขียนอักษร ฯลฯ. ดังนั้นจึงมีการสร้าง*ทูลล์คิต (toolkit)* ซึ่งเป็นไลบรารีอำนวยความสะดวกสำหรับเขียนโปรแกรมแบบ GUI ขั้นหนึ่ง. ไลบรารีทูลล์คิตช่วงแรกได้แก่ *Xt (X Toolkit Intrinsics)* เป็นไลบรารีช่วยให้สร้างโปรแกรม GUI ใน*เชิงอ็อบเจ็ค (object oriented)* ในภาษาซีได้ง่ายขึ้น. ทำให้การจัดการส่วนประกอบของ GUI ที่เรียกว่า*วิดเจ็ต (widget)* สะดวกขึ้น. ลำพังไลบรารี Xt อย่างเดียวยังไม่มีประโยชน์ในการสร้างโปรแกรมแบบ GUI. นอกจากนี้ยังต้องอาศัยไลบรารีที่เตรียมส่วนประกอบของ GUI เช่นปุ่มกด, ช่องเติมข้อความ เป็นต้น.

ทูลล์คิตที่ใช้กันในช่วงแรก ๆได้แก่ทูลล์คิต Athena และ Motif. ทูลล์คิต Athena หรือที่เรียกว่า *Xaw (X Athena Widget)* เป็นวิดเจ็ตแสดงผลเรียบ ๆธรรมดาเมื่อเทียบกับวิดเจ็ตของ Motif ซึ่งจะมีเงานูนเว้าทำให้ดูสวยน่าใช้กว่า. ปัจจุบันวิดเจ็ด Xaw ยังคงใช้อยู่กับแอพพลิเคชัน X ดั้งเดิมเช่น `xcalc`, `xlogo` เป็นต้น. ส่วนวิดเจ็ตแบบ Motif นั้นเดิมทีเป็นไลบรารีเชิงพาณิชย์. ในลินุกซ์จะใช้ไลบรารี Open Motif หรือ LessTif ซึ่งเป็นทูลล์คิต Motif แบบโอเพนซอร์ส.

`xcalc` โปรแกรมเครื่องคิดเลข.

โปรแกรมที่ใช้ไลบรารี Xt จะมีตัวเลือกร่วมที่เหมือนกันเช่น `-display`, `-geometry` ฯลฯ ตัวเลือกเหล่านี้ถือเป็นทรัพยากร (resource) อย่างหนึ่งในระบบ X วินโดว์. ค่าต่าง ๆเหล่านี้สามารถเขียนไว้ในไฟล์เริ่มต้นหรือระบุผ่านทางตัวเลือกก็ได้.



รูปที่ 6.1: ทูลล์คิตแบบ Athena และ Motif

ปัจจุบันทูลล์คิตที่นิยมใช้และมีบทบาทอย่างมากได้แก่ Gtk+ (The Gimp Toolkit) และ Qt. ทูลล์คิตสมัยใหม่เหล่านี้ใช้เขียนโปรแกรมแบบ GUI ง่ายกว่าทูลล์คิตสมัยแรกมากและยังใช้ในการสร้าง*สภาพแวดล้อมเดสก์ท็อป (desktop environment)* ที่นิยมใช้กันได้แก่ Gnome และ KDE ด้วย. โปรแกรมที่ใช้ทูลล์คิตสมัยใหม่เหล่านี้จะไม่ใช้ไลบรารี Xt, จะมีชุดตัวเลือกร่วมของทูลล์คิตตัวเองเช่น `--display=`*`name`*, `--screen=`*`number`* ฯลฯ. ตัวเลือกเหล่านี้จะแตกต่างกันไปตามแต่ละทูลล์คิต. ดังนั้นเวลาใช้ตัวเลือกควรจะระวังด้วยว่าโปรแกรมที่ต้องการใช้ใช้ทูลล์คิตอะไร. ในความเป็นจริงแล้วไม่ต้องกังวลเกี่ยวกับตัวเลือกมากเพราะค่าต่าง ๆมักปรับได้โดยใช้เมนูของโปรแกรมนั้น ๆ.

ตารางที่ 6.1: ไลบรารีทูลล์คิตสำคัญต่าง ๆและโปรแกรมที่ใช้ไลบรารีนั้น ๆ.

ไลบรารี Xt	ไลบรารี Gtk+	ไลบรารี Qt
`xterm`, `emacs`, `xdvi`, `xlogo`, `xman`, `xdpyinfo`, `xclock`, `bitmap`, `xfd`, `xfontsel`, `xcalc`, `xmag` ฯลฯ	`gnome-terminal`, `gimp`, `inkscape`, `ggv`, `firefox`, `mozilla`, `gedit`, `abiword`, `gthumb`, `evolution` ฯลฯ	`konsole`, `konqueror`, `kedit`, `kcalc`,`kword`, `kmail`, `kdevelop`, `kopete`, `kppp` ฯลฯ

6.2.2 หน้าจอและสกรีน

คำว่า*หน้าจอ (display)* และ*สกรีน (screen)* ในระบบ X วินโดว์มีความหมายต่างกัน. เซิร์ฟเวอร์หนึ่งโปรเซสจะมีหน้าจอเป็นคู่อยู่ด้วยเสมอ. หน้าจอจะมีความหมายถึงเซิร์ฟเวอร์เช่นการที่ไคลเอ็นต์ติดต่อกับเซิร์ฟเวอร์คือไคลเอ็นต์ติดต่อกับหน้าจอที่ใช้แสดง

ผล. ส่วนสกรีนจะหมายถึงฮาร์ดแวร์ดีไวซ์ที่ใช้แสดงผล. หน้าจอหนึ่งอาจมีได้หลายสกรีน. ชื่อของหน้าจอมีรูปแบบดังนี้

```
[hostname]:display[.screen]
```

- hostname คือชื่อโฮสเช่น localhost, mycomputer.net เป็นต้น. ถ้าละเว้นจะถือว่าเป็น localhost.
- display คือตัวเลขกำกับหน้าจอ. ตัวเลขนี้มีค่ามากกว่าหรือเท่ากับ 0. โดยปรกติเซิร์ฟเวอร์ตัวแรกจะมีหมายเลขหน้าจอเป็น 0. ถ้ามี X เซิร์ฟเวอร์รันอยู่อีกโปรเซสต้องใช้หมายเลขหน้าจออื่นที่ไม่ซ้ำกัน.
- screen คือตัวเลขสกรีนสามารถละเว้นไม่ระบุได้. ปรกติแล้วจะมีค่าเป็น 0.

ชื่อของหน้าจอนี้จะเก็บไว้ในตัวแปรสภาพแวดล้อมชื่อ DISPLAY. โดยปรกติแล้วระบบจะตั้งชื่อหน้าจอให้โดยอัตโนมัติ.

ตัวอย่างที่ 6.1: ชื่อหน้าจอ.

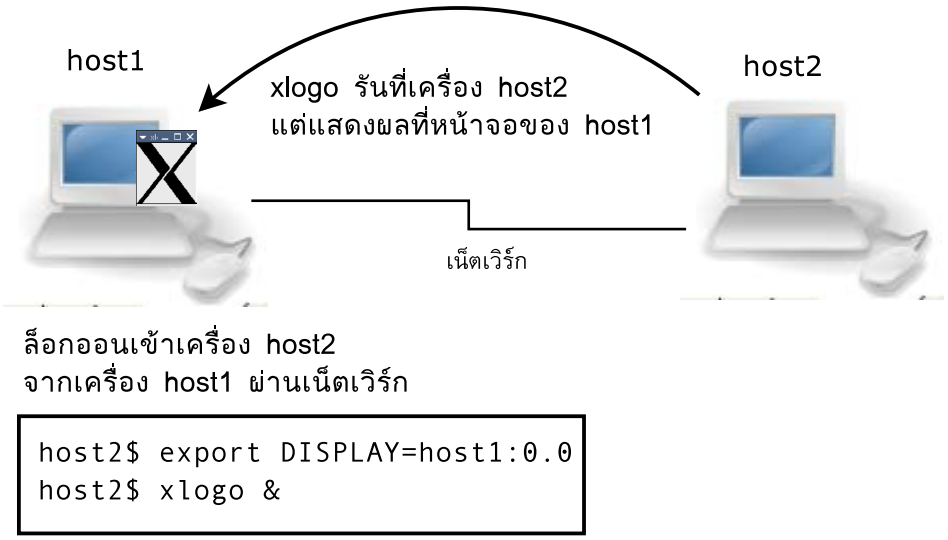
```
$ printenv DISPLAY↵                          ← แสดงค่าตัวแปรสภาพแวดล้อม DISPLAY
:0.0
$ xlogo &↵                                              ← รันโปรแกรม xlogo
$ unset DISPLAY↵
$ xlogo↵
Error: Can't open display                  ← ไม่สามารถติดต่อกับ X เซิร์ฟเวอร์ได้
```

ในตัวอย่างชื่อหน้าจอคือ :0.0. ถ้าไม่กำหนดตัวแปรสภาพแวดล้อมนี้ตอนรัน X ไคลเอ็นต์เช่นโปรแกรม xlogo ก็จะเกิดข้อผิดพลาดไม่สามารถติดต่อกับหน้าจอ (เซิร์ฟเวอร์) ได้.

นอกจากการระบุหน้าจอของ X เซิร์ฟเวอร์ด้วยตัวแปรสภาพแวดล้อมแล้วเรายังสามารถระบุหน้าผ่านทางตัวเลือกของโปรแกรมที่ใช้ไลบรารี Xt ได้ด้วย. ตัวเลือกนี้ได้แก่ -display *displayname*.

การระบุเซิร์ฟเวอร์ด้วยชื่อหน้าจอมีประโยชน์เวลารันโปรแกรมจากเครื่องคอมพิวเตอร์เครื่องหนึ่งแต่ต้องการแสดงผลบน X เซิร์ฟเวอร์ที่เครื่องอื่น. ตัวอย่างเช่นในรูปที่ 6.2 เป็นการใช้เทอร์มินัลเอมิวเลเตอร์ล็อกอินผ่านทางเน็ตเวิร์กจากเครื่อง host1 ไปที่เครื่อง host2 ด้วยคำสั่ง telnet. หลังจากที่ล็อกอินเรียบร้อยแล้วตั้งค่าตัวแปรสภาพแวดล้อม DISPLAY ให้เป็นหน้าจอของเครื่อง host1. เวลารันโปรแกรม GUI ก็จะแสดงผลบนหน้าจอ host1.

จากรูป ?? ไคล์เอ็นต์ได้แก่โปรแกรม xlogo รันในเครื่อง host2. ส่วนเซิร์ฟเวอร์ได้แก่ X เซิร์ฟเวอร์ที่รันอยู่ในเครื่อง host1. การติดต่อระหว่างไคลเอ็นต์และเซิร์ฟเวอร์ใช้โปรโตคอล X และอาศัยเน็ตเวิร์กเป็นตัวกลาง, ดังนั้นถ้ามีการติดตั้งไฟร์วอลล์ระหว่างเครื่องทั้งสองก็จะไม่สามารถแสดงหน้าต่างข้ามเครื่องได้.



รูปที่ 6.2: การแสดงหน้าต่างแอพพลิเคชันผ่านทางเน็ตเวิร์ก.

ถึงแม้ว่าจะไม่มีไฟร์วอลล์ระหว่างเครื่องทั้งสอง. ในบางกรณีอาจจะไม่สามารถแสดงหน้าต่างข้ามเครื่องได้เพราะ X เซิร์ฟเวอร์ไม่อนุญาตให้แสดงผล. วิธีแก้ไขคือเปิดเทอร์มินอลในเครื่อง host1 แล้วสั่งคำสั่ง `xhost` ซึ่งเป็นโปรแกรมควบคุมการเข้าถึงเซิร์ฟเวอร์ของเครื่องที่ใช้.

xhost อ้างอิงหน้า 428

ตัวอย่างที่ 6.2: ควบคุมการเข้าถึง X เซิร์ฟเวอร์.

```
$ xhost +↵                                    ← สั่งคำสั่งที่เครื่อง host1
access control disabled, clients can connect from any host
```

เครื่องหมาย + หมายถึงอนุญาตให้เครื่องคอมพิวเตอร์ใดๆก็ได้เข้าถึง X เซิร์ฟเวอร์ที่เครื่องตัวเอง (host1). ถ้าต้องการอนุญาตเฉพาะเครื่องก็ให้ระบุชื่อโฮสหรือ IP แอดเดรสตามหลังเครื่องหมาย +. ถ้าไม่อนุญาตการแสดงหน้าต่างจากเครื่องอื่นๆให้ใช้เครื่องหมาย −.

ssh ได้แก่การล็อกอินโดยโดยใช้โปรโตคอล secure shell ข้อมูลสื่อสารระหว่างเครื่องจะเข้ารหัสปลอดภัยกว่า telnet.

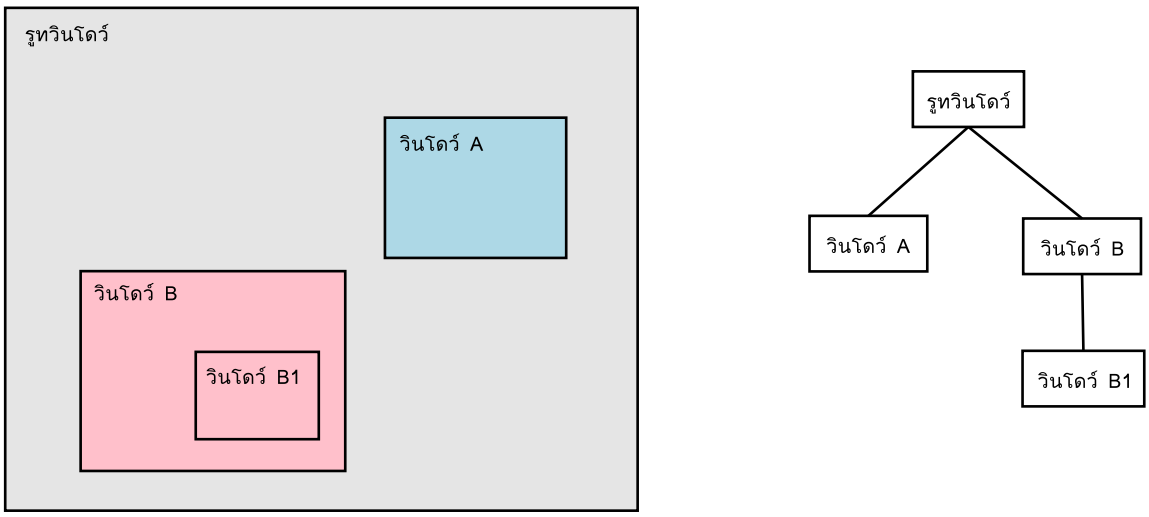
ถ้าเป็นการล็อกอินผ่าน `ssh` จะมีคุณสมบัติ X forwarding ให้. ถ้าเซิร์ฟเวอร์และไคล์เอ็นต์ของ `ssh` อนุญาตให้ใช้คุณสมบัตินี้ก็จะสามารถส่งผ่านหน้าต่างได้ผ่านโปรโตคอล ssh ไม่ผ่านโปรโตคอล X เหมือนในกรณี `telnet`. ถ้าใช้คุณสมบัติ X forwarding, โปรแกรม `ssh` จะจัดการค่าแปรสภาพแวดล้อม `DISPLAY` ให้โดยอัตโนมัติ. สำหรับระบบที่มีไฟร์วอลล์แต่อนุญาตให้ใช้ `ssh` วิธีนี้จะสะดวกที่สุด.

6.2.3 ระบบหน้าต่าง

โดยปรกติ, โปรแกรมแบบ GUI ที่เรียกใช้จะแสดงผลบนหน้าจอเป็นหน้าต่าง (วินโดว์). จะมีบางกรณีเท่านั้นที่ไม่แสดงผลเป็นหน้าต่างเช่น `xhost` จัดเป็นโปรแกรมใน

ระบบ X วินโดว์, มีการติดต่อกับ X เซิร์ฟเวอร์แต่ไม่มีการแสดงผลเป็นหน้าต่าง.

เมื่อ X เซิฟร์เวอร์ทำงาน, ในหน้าจอจะมีหน้าต่างพิเศษปรากฏอยู่เสมอเรียกว่า*รูทวินโดว์ (root window)*. รูทวินโดว์จะมีขนาดเท่าสกรีนที่ใช้. เมื่อมีการสร้างหน้าต่างสำหรับโปรแกรมจะเกิดความสัมพันธ์หน้าต่างขึ้น. เช่นในรูปที่ 6.3 วินโดว์ A และ B เป็นลูกของรูทวินโดว์. ในวินโดว์ B มีหน้าต่างย่อยลงไปอีก, ก็จะได้ความสัมพันธ์ว่าวินโดว์ B1 เป็นลูกของวินโดว์ B อีกทอดหนึ่ง. ระบบ X วินโดว์จะจัดการหน้าต่างในลักษณะนี้.



รูปที่ 6.3: ความสัมพันธ์ระหว่างวินโดว์ต่าง ๆ.

วินโดว์ที่แสดงในหน้าจอไม่จำเป็นต้องมองเห็นได้ตลอดเวลา. ในบางกรณีไคลเอ็นต์สามารถขอให้เซิร์ฟเวอร์ซ่อนหน้าต่างที่ต้องการและแสดงเมื่อขอร้องเซิร์ฟเวอร์ให้แสดงผล. หรือบางกรณีหน้าต่างอาจจะถูกทับด้วยหน้าต่างอื่น ๆ, เซิร์ฟเวอร์จะจัดการการแสดง/ไม่แสดงหน้าต่างโดยอัตโนมัติ. ถ้าหน้าต่างที่ทับหายไป, เซิร์ฟเวอร์ก็จะจัดการวาดหน้าต่างที่ถูกทับให้ปรากฏอีกที.

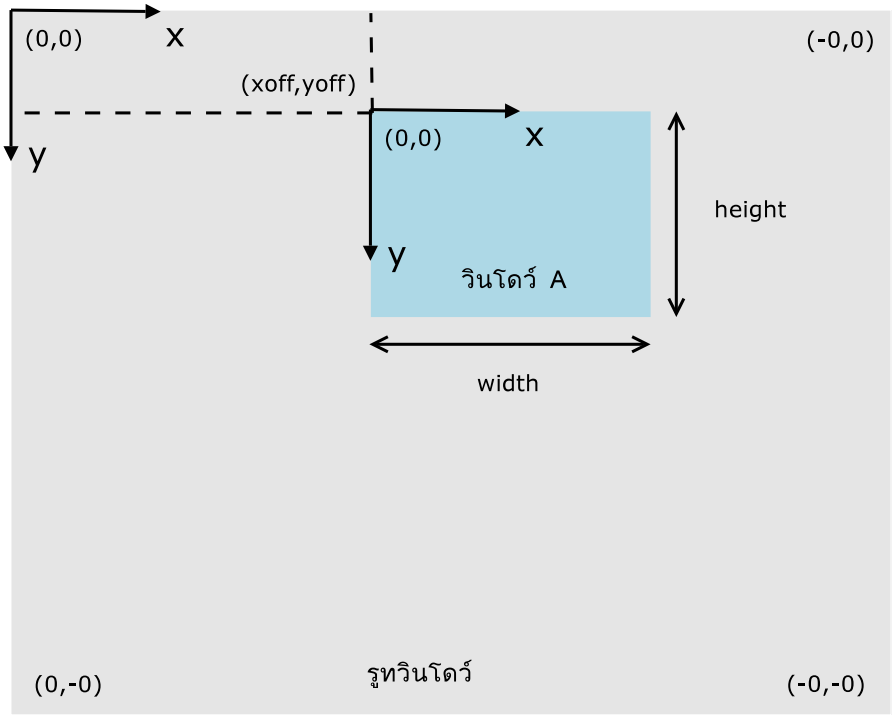
6.2.4 ตำแหน่งหน้าต่าง

สิ่งที่แสดงบน X วินโดว์จะเป็นกราฟิกแบบ*บิตแมป (bitmap)* คือเป็น*จุดพิกเซล (pixel)*. ความสามารถในการแสดงผลนี้ขึ้นอยู่กับอุปกรณ์เช่นหน้าจอ, กราฟิกการ์ดที่ใช้. ขนาดของหน้าจอมีหน่วยเป็นพิกเซลและระบบพิกัดที่ใช้อ้างอิงตำแหน่งจะมีจุดเริ่มต้นอยู่ที่มุมซ้ายบนของหน้าจอ. แกน x จะเริ่มจากซ้ายไปขวา, และแกน y จะเริ่มจากบนไปล่างตามรูปที่ 6.4.

สำหรับโปรแกรมที่ใช้ไลบรารี Xt สามารถระบุตำแหน่งของหน้าต่างได้ด้วยตัวเลือก `-geometry` มีรูปแบบดังนี้

```
program -geometry widthxheight+xoff+yoff
```

- width คือความกว้างของหน้าต่างโปรแกรมที่เรียกใช้.



รูปที่ 6.4: ตำแหน่งพิกัดในระบบ X วินโดว์.

- height คือความสูงของหน้าต่างโปรแกรมที่เรียกใช้.
- xoff คือตำแหน่งจากมุมซ้ายบนของรูทวินโดว์ตามแนวแกน x.
- yoff คือตำแหน่งจากมุมซ้ายบนของรูทวินโดว์ตามแนวแกน y.

ค่าตำแหน่งและขนาดนี้สามารถละเว้นบางส่วนได้เช่นระบุขนาดอย่างเดียวแต่ไม่ระบุตำแหน่งเป็นต้น. ตำแหน่งของหน้าต่างโดยปรกติจะอ้างอิงจากมุมซ้ายบนของรูทวินโดว์. ในกรณีที่ต้องการอ้างอิงจากมุมขวาบนหรือมุมขวาล่างของรูทวินโดว์ตามแนวแกน x เปลี่ยนเครื่องหมาย + เป็น -. ในทำนองเดียวกันถ้าต้องการอ้างอิงจากมุมซ้ายล่างหรือมุมขวาล่างของรูทวินโดว์ตามแนวแกน y ให้เปลี่ยนเครื่องหมาย + เป็น -. นอกจากนี้ในระบบ X วินโดว์รับรู้พิกัดพิเศษดังต่อไปนี้ด้วย

- -0+0 หมายถึงตำแหน่งมุมขวาบนของรูทวินโดว์.
- +0-0 หมายถึงตำแหน่งมุมซ้ายล่างของรูทวินโดว์.
- -0-0 หมายถึงตำแหน่งมุมขวาล่างของรูทวินโดว์.

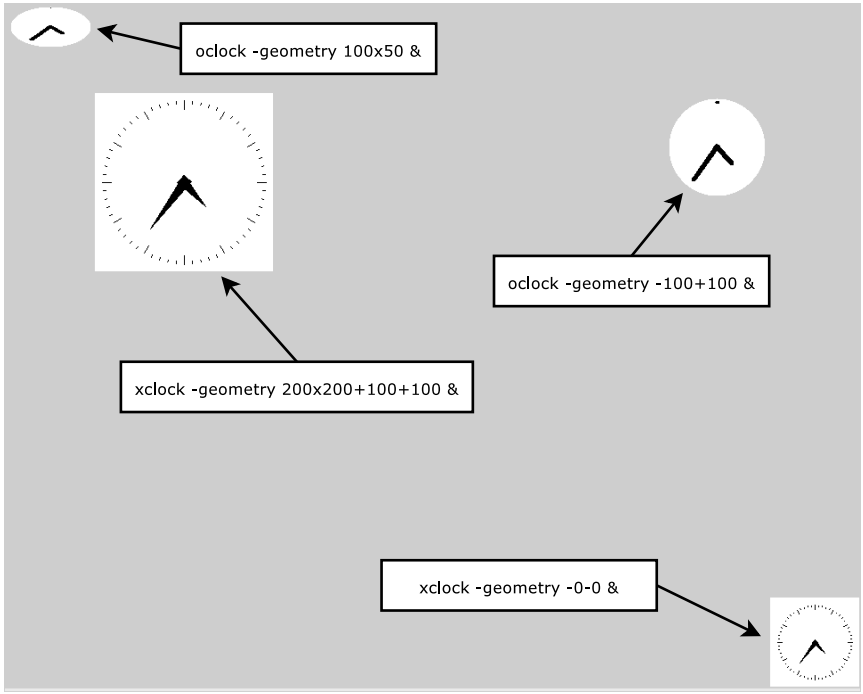
ตัวอย่างต่อไปนี้เป็นการแสดง `xclock` และ `oclock` โดยใช้ตัวเลือก `-geometry` ระบุขนาดและตำแหน่งต่าง ๆ.

`xclock` อ้างอิงหน้า 427
`oclock` อ้างอิงหน้า 427

`xclock` ใช้แสดงนาฬิกาทรงเหลี่ยม. `oclock` ใช้แสดงนาฬิกาทรงกลม.

ตัวอย่างที่ 6.3: ใช้ตัวเลือก -geometry ระบุขนาดและตำแหน่ง.

```
$ xclock -geometry 200x200+100+100 &↵
$ oclock -geometry -100+100 &↵                    ← ไม่ระบุขนาดหน้าต่าง
$ xclock -geometry 100x100-0-0 &↵                 ← อ้างอิงตำแหน่งมุมขวาล่าง
$ oclock -geometry 100x50 &↵                      ← ไม่ระบุตำแหน่ง
```



รูปที่ 6.5: ตัวอย่าง xclock และ oclock ในตำแหน่งและขนาดต่าง ๆ.

6.2.5 ตัวเลือกร่วมของไลบรารี Xt

โปรแกรม GUI ที่ใช้ทูลล์คิตของ Athena และ Motif จะมีตัวเลือกร่วมที่เหมือนกัน เพราะใช้ไลบรารีเหมือนกันคือ Xt. ตัวเลือกที่สำคัญ ๆของทูลล์คิตได้แก่

- `-display [host]:display[.screen]`
  ระบุหน้าจอ (เซิร์ฟเวอร์) ที่ต้องการแสดงผล.
- `-geometry geometry`
  ระบุตำแหน่งและขนาดของหน้าต่างโปรแกรม.
- `-fg color` หรือ `--foreground color`
  ระบุสีของฉากหน้าเช่นส่วนที่เป็นรูปทรงหลักของโปรแกรมหรือตัวอักษร (หน้า 247).
- `-bg color` หรือ `--background color`
  ระบุสีของฉากหลัง (พื้นหลัง) ของโปรแกรม.

- `-fn` *`font`* หรือ `-font` *`font`*
  ระบุชื่อฟอนต์ที่ต้องการใช้ (หน้า 290).

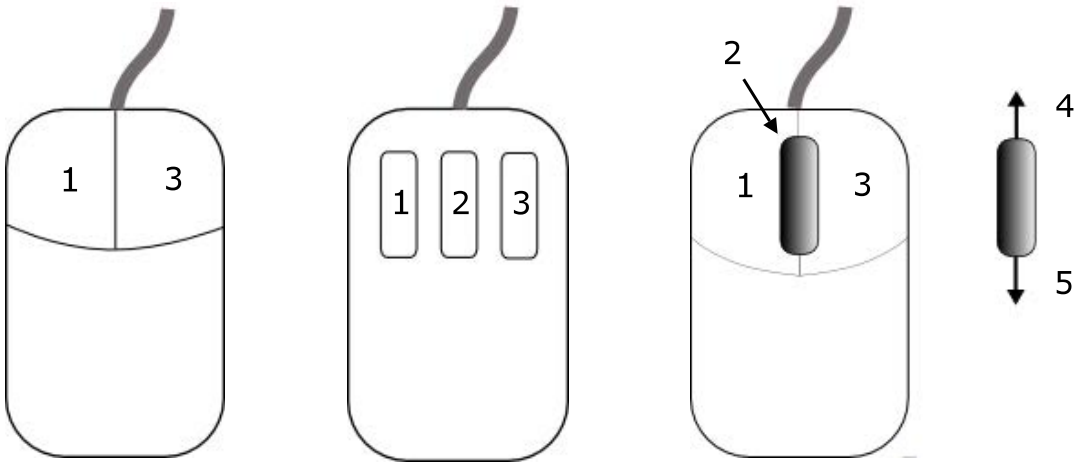
- `-title` *`title`*
  ชื่อหน้าต่างส่งต่อให้วินโดว์แมนเนเจอร์แสดงเป็นชื่อของหน้าต่าง.

- `-name` *`instance-name`*
  ระบุชื่อของโปรแกรมที่รัน. ถ้ามีการกำหนดทรัพยากรที่สอดคล้องกับชื่อนั้นก็จะใช้ค่าทรัพยากรตามที่ระบุซึ่งอาจจะแตกต่างจากการไม่ระบุชื่อได้. (หน้า 253).

- `-iconic`
  เริ่มต้นการทำงานโดยย่อให้เป็น*ไอคอน (icon)*.

- `-xrm` *`resource`*
  ระบุชื่อและค่าทรัพยากรที่ต้องการใช้ใ (หน้า 253).

6.2.6 เมาส์ในระบบ X วินโดว์



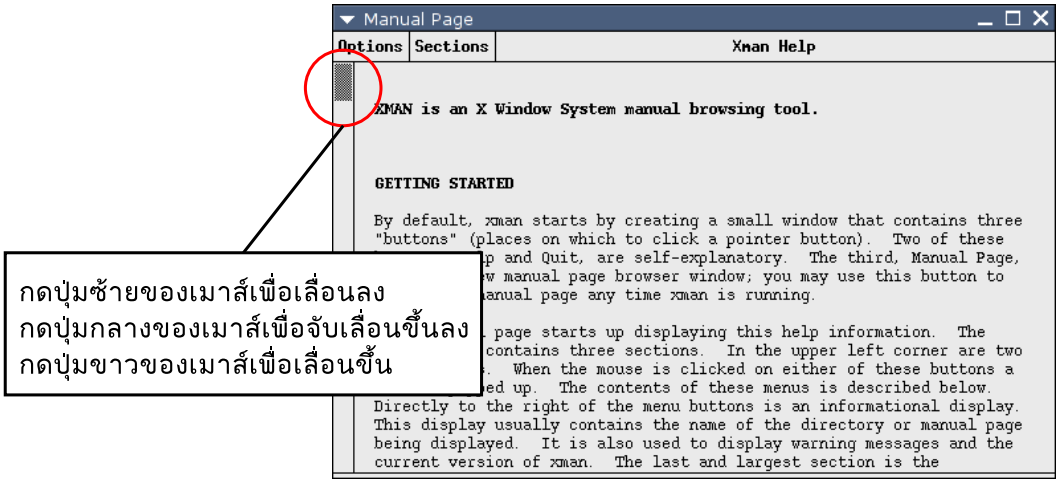
รูปที่ 6.6: เมาส์ทั่วไปที่ใช้ในระบบ X วินโดว์.

เมาส์ที่ใช้ยูนิกซ์เวิร์กสเตชันเดิมทีมีสามปุ่มคล้ายกับเมาส์ของเครื่องคอมพิวเตอร์ส่วนบุคคลในปัจจุบันซึ่งมักจะมีสามปุ่มและปุ่มตรงการเป็นลูกล้อ (wheel) เลื่อนขึ้นลงได้. การใช้เมาส์ในระบบ X วินโดว์เหมือนกับระบบ GUI อื่นๆคือปุ่มหลักที่ใช้ได้แก่ปุ่มซ้ายซึ่งแสดงด้วยหมายเลข 1 จากรูป 6.6. ปุ่มที่ 1 ใช้เลือกหน้าต่าง, กดลากและปล่อย (drag and drop) ฯลฯ. ถ้ามีการใช้เมาส์เลือกข้อความในโปรแกรม, สายอักขระที่เลือกจะถูกก็อปปี้ไว้ทันที. ถ้ากดปุ่มที่ 2 (ปุ่มกลาง) ก็จะแปะสายอักษรที่เลือกไว้ลงในหน้าต่างที่ต้องการ. วิธีการตัดแปะสายอักขระแบบนี้เป็นคุณสมบัติของ X วินโดว์ซึ่งแตกต่างจากระบบปฏิบัติการวินโดว์ที่ต้องเลือกสายอักษรที่ต้องการคัดลอกก่อน, คัดลอด (copy) ด้วยคีย์ `Ctrl+c` หรือจากเมนู, และทำการแปะ (paste) ด้วยคีย์ `Ctrl+v` หรือจากเมนู. ในสภาพแวดล้อม

เดสก์ท็อปเช่น GNOME หรือ KDE ใช้ได้ทั้งการตัดแปะสายอักขระแบบ X วินโดว์และแบบระบบปฏิบัติการอื่น ๆ.

การกระทำของปุ่มเมาส์ที่ 3 แล้วแต่แอพพลิเคชันที่ใช้. ในสภาพแวดล้อมเดสก์ท็อปอาจจะแสดงเมนูเมื่อกดปุ่มนี้. ส่วนลูกกลิ้งซึ่งจะถือว่าเป็นปุ่มที่ 4 และ 5 สามารถใช้เลื่อนเนื้อหาที่แสดงในหน้าต่างขึ้นลงได้แล้วแต่โปรแกรม. เมาส์รุ่นเก่าที่มีสองปุ่มสามารถจำลองการกดเมาส์ปุ่มที่สองได้ด้วยการกดเมาส์ปุ่มที่ 1 และ 3 พร้อม ๆกัน.

สำหรับโปรแกรมที่ใช้ทูลล์คิต Athena และมี*สกรอลล์บาร์ (scroll bar)* สำหรับเลื่อนหน้าต่างแนวตั้งแนวนอน, วิธีการใช้เมาส์จะไม่เหมือนกับอินเทอร์เฟสสมัยใหม่ที่ใช้ทูลลคิต Gtk+ หรือ Qt ที่สามารถใช้เมาส์ปุ่มที่ 1 จับสกรอลล์บาร์แล้วลากบังคับ. ถ้าต้องการเลื่อนหน้าต่างแสดงผลลงหรือเลื่อนไปทางขวาโดยสกรอลล์บาร์ของทูลล์คิต Athena, ให้คลิ้กเมาส์ปุ่มที่ 1. ในทางกลับกันถ้าต้องการเลื่อนขึ้นหรือเลื่อนไปทางซ้าย, ให้คลิ้กเมาส์ปุ่มที่ 3. เมาส์ปุ่มที่ 2 ใช้จับสกรอลล์บาร์แล้วลากบังคับได้.



รูปที่ 6.7: การใช้เมาส์ควบคุมสกรอลล์บาร์ของโปรแกรมที่ใช้ทูลล์คิต Athena.

6.2.7 สีในระบบ X วินโดว์

การระบุสีในระบบ X วินโดว์ใช้หลักการผสมสีของแสงจากแม่สีสีแดง (Red), สีเขียว (Green) และสีน้ำเงิน (Blue). วิธีแสดงชี่สีเช่นสำหรับให้เป็นอาร์กิวเมนต์ของตัวเลือก `-fg` หรือ `-bg` มี 2 วิธีได้แก่

- **เลขฐานสิบหก**
  แสดงเป็นเลขฐานสิบหกในรูปของ *#RRGGBB* โดยที่ *RR*, *GG* และ *BB* คือตัวเลขฐานสิบหกมีค่าตั้งแต่ 00 ถึง FF. ตัวอย่างเช่น #FF0000 แทนสีแดง, #191970 แทนสีที่มีชื่อว่า Midnight Blue เป็นต้น. ในกรณีจะสมมติว่าความละเอียดของแม่สีแต่ละสีเป็น 8 บิต.

- **ชื่อสี**

ในระบบ X วินโดว์มีการกำหนดชื่อสีไว้ในไฟล์ /usr/X11R6/lib/X11/rgb.txt. ข้อมูลในไฟล์เป็นข้อมูลเท็กซ์แสดงชื่อสีบรรทัดต่อบรรทัด. ในหนึ่งบรรทัดประกอบด้วยตัวเลขฐานสิบ 3 ตัวแทนค่าของสีแดง, สีเขียว, และสีน้ำเงินตามลำดับ. ตามด้วยชื่อของสีเป็นภาษาอังกฤษ. ในระบบ X วินโดว์จะมีคำสั่งมาตรฐานชื่อ showrgb แสดงเนื้อหาของไฟล์นี้โดยที่ไม่ต้องรู้ว่าไฟล์นี้อยู่ที่ไหน. นอกจากคำสั่งมาตรฐาน

ตัวอย่างที่ 6.4: ไฟล์ /usr/X11R6/lib/X11/rgb.txt.

```
$ cat /usr/X11R6/lib/X11/rgb.txt↵
! $Xorg: rgb.txt,v 1.3 2000/08/17 19:54:00 cpqbld Exp $
255 250 250             snow
248 248 255             ghost white
248 248 255             GhostWhite
245 245 245             white smoke
245 245 245             WhiteSmoke
--- แสดงผลต่อไปเรื่อยๆ ---
```

ชื่อของสีบางสีในไฟล์อาจจะมีช่องไฟระหว่างชื่อ. เวลาระบุชื่อสีในบรรทัดคำสั่งจะใช้แบบไหนก็ได้. ถ้าเป็นชื่อสีที่มีช่องไฟระหว่างกลางต้องเขียนในเครื่องหมายคำพูด. ชื่อที่ระบุจะใช้ตัวอักษรตัวเล็กหรือตัวใหญ่ก็ได้. ถ้าระบุชื่อสีโดยใช้เลขฐานสิบหกก็ต้องใช้เครื่องหมายคำพูดคล่อมเช่นกันเพราะมีการใช้เครื่องหมาย #.

xeyes เป็นโปรแกรมมาตรฐานแสดงลูกตามองตามตำแหน่งของเมาส์ในหน้าจอ.

ตัวอย่างที่ 6.5: การระบุชื่อสีในบรรทัดคำสั่ง.

```
$ xeyes  xeyes -bg "ghost white" -fg "#b0e0e6"↵
```

- **ชื่อสีในรูปทั่วไป**
  วิธีสองแบบที่ผ่านมาเป็นวิธีดั้งเดิมที่ใช้ในการระบุสีในระบบ X วินโดว์. ปัจจุบันมีวิธีระบุสีในรูปแบบต่อไปนี้

```
<color space name>:<value>/.../<value>
```

<*color space name*> คือชื่อของ color space ทำให้สามารถระบุสีได้หลายวิธี. การระบุสีโดยใช้แมสีแดง, เขียว, และน้ำเงินเป็นวิธีหนึ่งในหลาย ๆวิธี. ถ้าจะเขียนในรูปแบบนี้จะเป็น

```
rgb:<red_value>/<gree_value>/<gree_value>
```

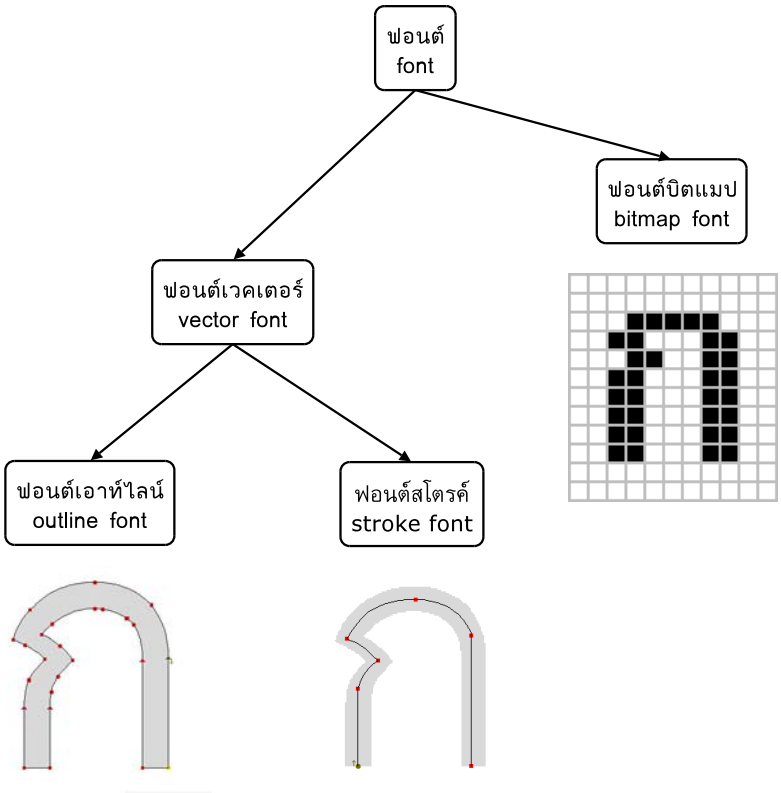
ค่าของแม่สีต่าง ๆเขียนอยู่ในรูปของเลขฐานสิบหก 1 หลัก (4 บิต), 2 หลัก (8 บิต), 3 หลัก (12 บิต), และ 4 หลัก (16 บิต) แล้วแต่ความละเอียด. color space นอกเหนือจาก rgb แล้วได้แก่ rgbi, CIEXYZ, CIEuvY, CIExyY, CIELab, CIELuv และ TekHVC.

CIE ย่อมาจาก Commission Internationale de l'Éclairage เป็นองค์กรรับผิดชอบเกี่ยวกับมาตรฐานเรื่องสีและแสง. หารายละเอียดเพิ่มเติมได้จากเรื่องเกี่ยวกับทฤษฎีสี.

6.2.8 ฟอนต์

ฟอนต์ (font) หรือภาษาไทยเรียกว่าแบบอักษรคือกลุ่มของรูปทรงตัวอักษร, อักขระต่าง ๆรวมไว้ในไฟล์. ไฟล์ฟอนต์ที่อยู่ใน*ตระกูล (family)* เดียวกันจะมีรูปร่างคล้ายเหมือนกัน, จะแตกต่างกันที่รูปทรงเช่นทรงตั้งตรง, ทรงหนา, ทรงเอียง เป็นต้น. ฟอนต์ที่ใช้ได้กับ X เซิร์ฟเวอร์แบ่งกว้าง ๆได้เป็น 2 แบบคือ*ฟอนต์บิตแมป (bitmap)* และ*ฟอนต์เวกเตอร์ (vector)*. ฟอนต์เวกเตอร์ที่ใช้ใน X วินโดว์ได้แก่*ฟอนต์เอาท์ไลน์ (outline font)*เช่น*ฟอนต์ทรูไทป์ (truetype)* หรือ*ฟอนต์ไทป์ 1 (Type 1)* เป็นต้น.

ในหนังสือเล่มนี้ขอใช้คำสั่งว่าฟอนต์แทนคำว่าแบบอักษร.



รูปที่ 6.8: ฟอนต์ประเภทต่าง ๆ.

ฟอนต์บิตแมป

ฟอนต์บิตแมปเป็นฟอนต์ดั้งเดิมที่ใช้ที่ในระบบ X วินโดว์. การแสดงอักขระแต่ละตัวจะใช้จุดในการแสดงผล. ดังนั้นฟอนต์แต่ละฟอนต์จะมีขนาดตายตัว. ในกรณีที่ต้องแสดงตัวอักษรหลายขนาดต้องเตรียมไฟล์ฟอนต์ของแต่ละขนาด. หรือจะใช้การย่อขยายฟอนต์ที่มีอยู่ก็ได้แต่จะไม่สวยงามเพราะการแสดงอักขระกำหนดโดยจุดตายตัว. ฟอนต์แบบบิตแมปมีข้อดีที่ว่าไฟล์ฟอนต์แต่ละไฟล์สร้างเฉพาะสำหรับฟอนต์แต่ละขนาด, ดังนั้นจึงแสดงผลได้คมชัดสวยงาม. ความคมชัดของฟอนต์จะเห็นผลชัดเมื่อตัวอักขระที่แสดงมีขนาดเล็ก.

ฟอร์แมตของฟอนต์บิตแมปที่ใช้ใน X วินโดว์ได้แก่ *BDF (Bitmap Distribution*

Format) และ *PCF (Portable Compiled Fotmat)*. ไฟล์ฟอนต์ BDF มักจะมีส่วนขยายชื่อไฟล์เป็น `.bdf` เป็นไฟล์เท็กซ์ธรรมดาสามารถอ่านเข้าใจได้. ส่วนไฟล์ฟอนต์ PCF มักมีส่วนขยายชื่อไฟล์เป็น `.pcf` เป็นไฟล์ไบนารีที่แปลงมาจากไฟล์ฟอนต์ BDF อีกทีด้วยโปรแกรม `bdftopcf`. ไฟล์ฟอนต์ที่ใช้จริงในระบบมักจะเป็นไฟล์ PCF ที่บีบอัดด้วย `gzip` อีกที. ฟอนต์เหล่านี้สามารถสร้างด้วยโปรแกรมเช่น `xmbdfedit`, `fontforge` ฯลฯ.

ฟอนต์เอาต์ไลน์

ฟอนต์เอาต์ไลน์ที่เป็นที่รู้จักกันเช่นฟอนต์ฟอร์แมต Type 1 ซึ่งเป็นฟอนต์สำหรับภาษา PostScript และฟอนต์ทรูไทป์. ปัจจุบันแนวโน้มการใช้ฟอนต์เริ่มหันไปสู่ฟอนต์แบบ*โอเพนไทป์ (OpenType)* ขึ้นเรื่อย ๆ. ฟอนต์เหล่านี้สามารถสร้างด้วยโปรแกรม `fontforge` ในลินุกซ์.

PostScript ▶
ภาษาคอมพิวเตอร์สำหรับงานกราฟิก, นิยมใช้ในงานพิมพ์ต่าง ๆ. ในลินุกซ์จะมีตัวแปลภาษา PostScript ที่เรียกว่า Ghostscript ซึ่งใช้แปลภาษา PostScript แสดงผลในดีไวซ์ต่าง ๆได้.

- **ฟอนต์ไทป์ 1**
  เป็นฟอนต์ที่พัฒนาโดยบริษัท Adobe มีชื่อเรียกทั่วไปว่าฟอนต์ PostScript เพราะเป็นฟอนต์ที่ออกแบบใช้งานกับภาษา PostScript. รูปทรงของอักขระจะสร้างจากเส้นโค้ง Cubic Bézier เมื่อย่อหรือขยายจะไม่มีหยักเหมือนฟอนต์บิตแมปเพราะการแสดงผลจะวาดรูปทรงใหม่ตามเส้นโค้งที่กำหนดไว้ในฟอนต์.

  ฟอร์ปแมตของฟอนต์ไทป์ 1 มีสองประเภทคือ *PFA (PostScript Font ASCII)* ซึ่งเป็นไฟล์ฟอนต์ข้อมูลเท็กซ์ธรรมดา, และฟอร์แมต *PFB (PostScript Font Binary)* ซึ่งเป็นไฟล์ฟอนต์ข้อมูลไบนารี. ข้อมูลที่เกี่ยวกับตัวอักขระเช่นความกว้าง, การ*เคิร์นนิงค์ (kerning)* ฯลฯ จะอยู่ในไฟล์แยกต่างหากคือไฟล์ *AFM (Adobe Font Metric)*. ฟอนต์ PostScript นี้สามารถเก็บข้อมูลของอักขระได้มากที่สุด 256 ตัว.

Bezier ▶
เส้นโค้ง Bézier เป็นเส้นโค้งคณิตศาสตร์แบบพารามิเตอร์. นิยมใช้ในโปรแกรมกราฟิกต่าง ๆเช่น Gimp, Inkscape ฯลฯ. รูปทรงของฟอนต์ทรูไทป์เป็นเส้นโค้ง Quadratic Bezier ซึ่งอันดับ (เลขยกกำลัง) ของพารามิเตอร์เป็น 2. ส่วนฟอนต์ PostScript ใช้เส้นโค้ง Cubic Bezier มีอันดับของพารามิเตอร์เป็น 3.

kerning ▶
การปรับระดับ, ตำแหน่งของอักขระเมื่อมีการแสดงผลร่วมกับอักขระอื่นให้เหมาะสมสวยงาม.

- **ฟอนต์ทรูไทป์**
  เป็นฟอนต์ที่บริษัท Apple และ Microsoft ร่วมกันพัฒนาใช้กันอย่างกว้างขวางในระบบปฏิบัติการวินโดวส์และ Mac OS. ฟอนต์ทรูไทป์มีข้อมูลอักขระเป็นเส้นโค้ง Quadratic Bézier, ข้อมูลเกี่ยวกับรูปทรงเช่นความกว้าง, ขนาด ฯลฯ จะเก็บไว้ในไฟล์ฟอนต์เดียวกันไม่แยกเหมือนกับฟอนต์ PostScript.

- **ฟอนต์โอเพนไทป์**
  เป็นฟอนต์ที่เริ่มพัฒนาโดย Microsoft และเข้าร่วมพัฒนาโดย Adobe ภายหลัง. คุณสมบัติเด่นของฟอนต์โอเพนไทป์เช่นอิงการเข้ารหัสอักขระแบบ*ยูนิโค้ด (unicode)*, บรรจุข้อมูลอักขระได้มากถึง 65,536 ตัว, มีข้อมูลเฉพาะเกี่ยวกับภาษาประกอบ เป็นต้น.

สำหรับฟอนต์ของภาษาไทยที่มีอักขระมากกว่า 256 เช่นภาษาญี่ปุ่นจะใช้เทคนิคอื่นได้แก่ CID (Character IDentifier Keyed Font)

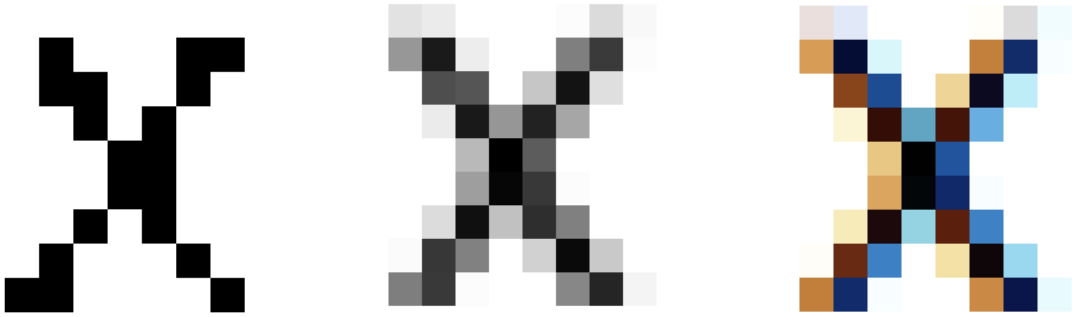
unicode ▶
มาตรฐานสำหรับการแสดงข้อมูลเท็กซ์หลายภาษาพร้อม ๆกัน. เดิมทีข้อมูลในคอมพิวเตอร์ไม่ได้คำนึงถึงการใช้ภาษาอื่น ๆทำให้ช่วงแรก ๆ คอมพิวเตอร์รองรับข้อมูลเท็กซ์เฉพาะภาษาอังกฤษซึ่งต้องการเนื้อที่สำหรับบันทึกอักขระแค่ 7 บิตก็เพียงพอ. ต่อมาคอมพิวเตอร์ใช้กันอย่างแพร่หลายทำให้ต้องเพิ่มเนื้อที่สำหรับภาษาอื่น ๆด้วยจึงเกิดมาตรฐานยูนิโค้ดขึ้น. อักขระทุกภาษาสามารถแสดงด้วยข้อมูล 16 บิต.

การแสดงอักขระของฟอนต์เวกเตอร์ทางหน้าจอ

โดยปรกติหน้าจอหรือเครื่องพิมพ์มักจะแสดงผลเป็นจุด. ข้อมูลฟอนต์แบบเวกเตอร์ต้องมีการแปลงข้อมูลให้แสดงด้วยจุดทางหน้าจอเวลาแสดงผล. การแสดงผลนี้เรียกว่า*การเ*

รนเดอร์ (rendering). การเรนเดอร์นี้เป็นการแปลงข้อมูลอนาล็อก (ข้อมูลฟอนต์เวกเตอร์) เป็นข้อมูลดิจิตอล (บิตแมป) ซึ่งบางกรณียากที่จะแสดงผลให้สวยงามเหมือนข้อมูลที่เป็นดิจิตอลเช่นฟอนต์บิตแมปอยู่แล้ว. ส่วนที่มักจะมีปัญหาแสดงผลได้ไม่ดีเช่นส่วนโค้งหรือส่วนเอียงของอักขระ. ปรากฏการณ์ที่แสดงผลได้ไม่สวยงามเพราะการแปลงข้อมูลเช่นนี้เรียกว่า*อเลียสซิงค์ (aliasing)*. *แอนติอเลียส (anti-alias)* เป็นวิธีแก้รอยหยักที่เกิดจากอเลียสซิงค์โดยเพิ่มพิกเซลที่มีค่าเฉลี่ยบริเวณส่วนที่หยักทำให้ผลที่ปรากฏทางหน้าจอดูดีขึ้น. ถึงแม้ว่าวิธีแอนติอเลียสจะทำให้ตัวอักขระที่แสดงดูดีขึ้นในระดับหนึ่ง, แต่การเพิ่มพิกเซลนี่เองทำให้บางครั้งตัวอักขระจะดูเลือนลางโดยเฉพาะอักขระที่มีขนาดเล็ก.

สำหรับจอภาพแบบ *LCD (Liquid Crytal Display)* หรือที่เรียกอีกอย่างว่าจอภาพ *TFT (Thin Film Transitor)* จะมีวิธีที่เรียกว่า*ซับพิกเซลเรนเดอริงค์ (sub-pixel rendering)* ช่วยทำให้คุณภาพการแสดงผลของฟอนต์หรือภาพดีกว่าปรกติ. วิธีนี้อาศัยหลักการที่ว่าจอภาพ LCD ในหนึ่งพิกเซลจะประกอบด้วยจุดย่อยหรือเรียกว่าซับพิกเซลซึ่งเป็นแม่สีสีแดง, สีเขียว, และสีน้ำเงินอีกสามจุด. ทำให้เหมือนกับว่าหน้าจอมีความละเอียดทางแนวนอนหรือทางแนวตั้งเพิ่มเป็น 3 เท่าแล้วแต่จอภาพ. ผลของซับเรนเดอริงค์จะเห็นได้ชัดโดยเฉพาะส่วนที่เป็นเส้นทแยง. เส้นบางเส้นจะใช้ซับพิกเซลช่วยแสดงผลแทนที่จะใช้พิกเซลทั้งหมด, ทำให้ผลที่ได้ดูละเอียดขึ้น.

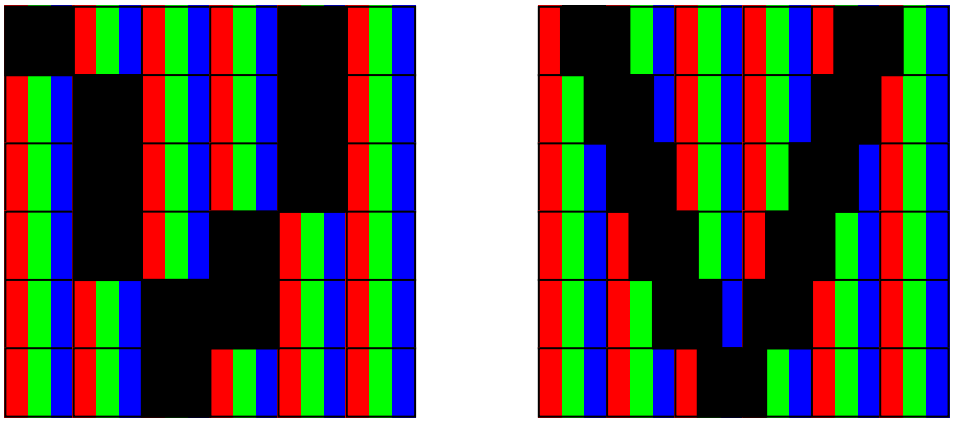


(ก) ไม่มีการปรับแต่ง (ข) แอนติอเลียส (ค) ซับพิกเซลเรนเดอร์

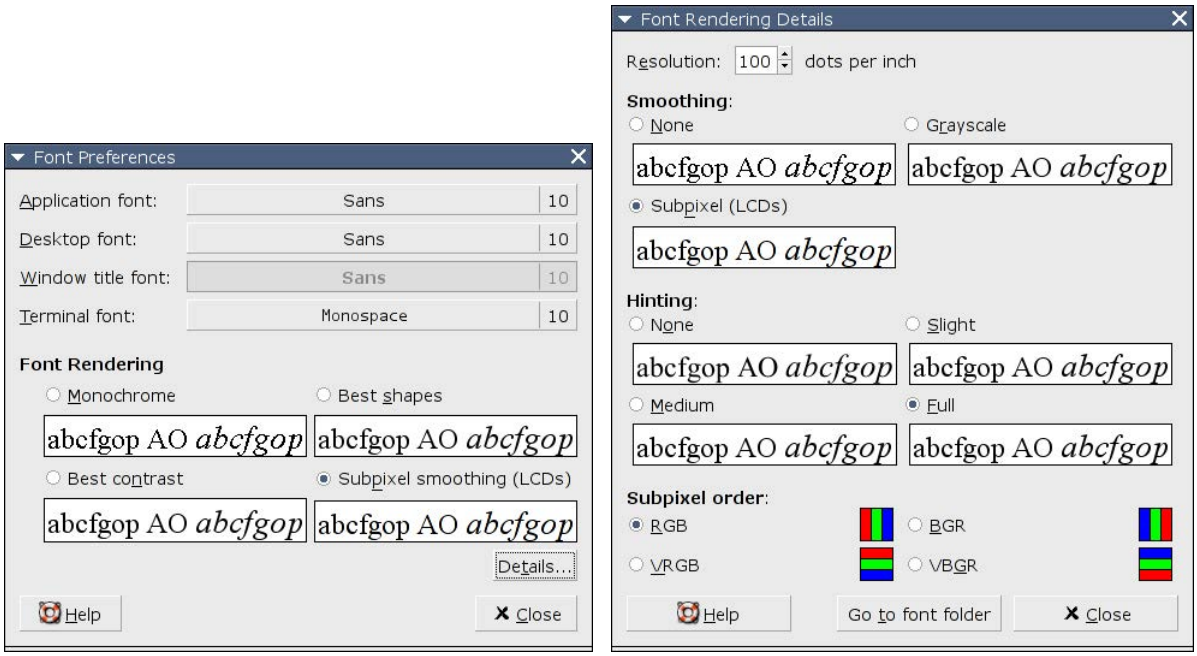
รูปที่ 6.9: การแสดงผลของอักขระด้วยการแอนติอเลียส.

การแสดงผลของอักขระให้สวยยังขึ้นอยู่กับฟอนต์ที่ใช้ด้วย. ฟอนต์บางฟอนต์มีข้อมูลภายในเรียกว่า*ฮินท์ (hint)* เป็นข้อมูลช่วยบอกใบ้ให้การแสดงผลอักขระตัวเล็กให้ถูกต้อง. ช่วยให้การแสดงผลในแต่ละพิกเซลอยู่ในตำแหน่งที่สมควร. นอกจากการใช้ฮินท์ช่วยแสดงผลอักขระตัวเล็ก ๆแล้ว, บางฟอนต์อาจจะมีข้อมูลบิตแมปผังอยู่ในฟอนต์ใช้แสดงผลเมื่ออักขระมีขนาดเล็ก. กล่าวคือมีข้อมูลเวกเตอร์และบิตแมปอยู่ในไฟล์เดียวกัน.

ในระบบ X วินโดว์สมัยใหม่ที่ใช้สภาพแวดล้อมเดสก์ท็อปเช่น GNOME สามารถตั้งค่าคุณสมบัติเพื่อช่วยให้แสดงผลฟอนต์ได้สวยงามขึ้นจากเมนู “Desktop preference” > “Font”.



รูปที่ 6.10: หลักการซับพิกเซลเรนเดอร์ถ้ามองหน้าจอโดยใช้แว่นขยาย.



รูปที่ 6.11: หน้าจอในสภาพแวดล้อมเดสก์ท็อป GNOME ให้เลือกการเรนเดอร์ฟอนต์แบบต่าง ๆ.

6.2.9 แป้นพิมพ์

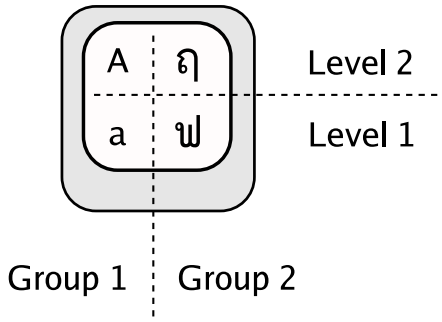
แป้นพิมพ์เป็นฮาร์ดแวร์อย่างหนึ่งในหลาย ๆตัวที่ใช้ทั่วไปกับคอมพิวเตอร์. เวลาเรากดคีย์ใดคีย์หนึ่งจะมีสัญญาณอิเล็กทรอนิกส์บอกสภาพของคีย์ว่ากำลังถูกกดอยู่หรือกำลังปล่อยหลังจากการกด. สภาพนี้เรียกว่า*อิเวนต์ (event)* และสัญญาณอิเล็กทรอนิกส์ที่ส่งมาจากแป้นพิมพ์เรียกว่า*สแกนโค้ด (scancode)*.

X เซิร์ฟเวอร์จะรับรู้อิเวนต์ของคีย์และสัญลักษณ์หรือค่าที่เรียกว่า*คีย์โค้ด (keycode)* ซึ่งมีแปลงมาจากสัญญาณสแกนโค้ดอีกทีหนึ่ง. คีย์โค้ดนี้เป็นเพียงแค่ค่า, ไม่ใช่สัญลักษณ์ที่แสดงบนคีย์เช่น “a”, “b”, “c”. สัญลักษณ์ที่มีความหมายและมนุษย์รับรู้ได้จะแปลงมา

อิเวนต์ที่เกี่ยวกับแป้นพิมพ์มีชื่อเรียกเฉพาะว่า*คีย์บอร์ดอิเวนต์ (keyboard event)*, อิเวนต์ที่เกี่ยวกับเมาส์มีชื่อเรียกเฉพาะว่า*เมาส์อิเวนต์ (mouse event)*.

จากคีย์โค้ดอีกทีหนึ่งเรียกว่า*คีย์ซิม (keysym)* เช่นการกดคีย์ที่มีอักษร “a” อยู่บนคีย์นั้นก็จะได้สัญญลักษณ์ที่ความหมายเป็นตัวอักษร “a”.

โดยปรกติในแป้นพิมพ์จะมีคีย์พิเศษเช่นคีย์ Control, Caps Lock และ Shift. คีย์เหล่านี้เรียกว่า*คีย์โมดิไฟเออร์ (modifier key)*. ถ้ากดคีย์โมดิไฟเออร์พร้อมกับกดคีย์ธรรมดาสามารถสร้างสัญลักษณ์ที่เกิดจากคีย์ธรรมดานั้นได้หลายแบบ. เช่นถ้าเรากดคีย์ Shift ค้างไว้และคีย์ “a” ตามจะได้คีย์ซิมเป็นสัญลักษณ์ “A”. ถ้าไม่กดคีย์โมดิไฟเออร์ใด ๆก็จะได้สัญลักษณ์คีย์ซิมปรกติของคีย์นั้น. คีย์โมดิไฟเออร์ในระบบ X วินโดว์มีอยู่ด้วยกัน 8 ตัว ได้แก่ Control, Shift, Lock, Mod1, Mod2, Mod3, Mod4 และ Mod5. Mod1 ถึง Mod5 เป็นชื่อของคีย์โมดิไฟเออร์ทั่วไป, ไม่มีชื่อเฉพาะและบนแป้นพิมพ์ก็ไม่มีคีย์ที่ชื่อ Mod1, Mod2 ฯลฯ. โดยปรกติแล้ว, คีย์ Alt จะใช้เป็นคีย์โมดิไฟเออร์ Mod1, ส่วนคีย์โมดิไฟเออร์ที่เหลือไม่นิยมใช้.



รูปที่ 6.12: ระดับและกลุ่มในโมดูล XKB ของ X เซิร์ฟเวอร์.

แป้นพิมพ์ภาษาอังกฤษโดยปรกติจะใช้แค่คีย์โมดิไฟเออร์ Shift ก็เพียงพอที่จะแสดงอักขระได้ทุกตัว. แต่สำหรับภาษาอื่น ๆเช่นลาติน (ยุโรป), ไทย ฯลฯ จำเป็นต้องใช้แป้นพิมพ์คนละชุดจึงเกิดความคิดขยายความสามารถของแป้นพิมพ์เป็นโมดูลใช้กับ X เซิร์ฟเวอร์เรียกว่า *XKB (X Keyboard Extension)*. โมดูล XKB ช่วยปรับแต่งคุณสมบัติของแป้นพิมพ์ในหลาย ๆด้านนอกจากภาษาด้วยเช่นการปรับผังหรือประเภทของแป้นพิมพ์เป็นต้น. โมดูล XKB ยังแบ่งสัญญลักษณ์ของคีย์ซิมเป็น*ระดับ (level)* และ*กลุ่ม (group)*. ระดับคือสภาพที่บอกว่ามีการใช่คีย์โมดิไฟเออร์ Shift ด้วยหรือไม่. กลุ่มเป็นการแสดงกลุ่มของอักขระที่ต้องการแสดงเช่นกลุ่มภาษาอังกฤษ, กลุ่มภาษาไทย เป็นต้น. การปรับแต่งค่าที่เกี่ยวกับโมดูล XKB นี้ทำได้จากไฟล์ตั้งค่าเริ่มต้นของ X เซิร์ฟเวอร์หรือจากบรรทัดคำสั่งก็ได้. ในบางกรณีเช่นสภาพแวดล้อมเดสก์ท็อป GNOME สามารถติดตั้งคีย์บอร์ดภาษาแบ่งกลุ่มได้จากเมนูหลักแทนที่จะไปติดตั้งในไฟล์เริ่มต้นของ X เซิร์ฟเวอร์.

6.2.10 ทรัพยากร

เราได้เรียนรู้ไปแล้วว่าโปรแกรม GUI ดั้งเดิมที่ใช้ไลบรารี Xt มีตัวเลือกเหมือน ๆกันใช้สำหรับปรับแต่งโปรแกรมเช่นตัวเลือก `-bg` สำหรับระบุสีที่ใช้เป็นพื้นหลังของตัวโปรแกรม ฯลฯ. คุณสมบัติที่ปรับแต่งได้เหล่านี้ในระบบ X วินโดว์จะเรียกว่า*ทรัพยากร (resource)*. โปรแกรมแต่ละตัวจะมีค่าทรัพยากรโดยปริยายระบุคุณสมบัติต่าง ๆของโปรแกรม

เช่น สีของเส้นที่ใช้, ฟอนต์, ขนาดหน้าต่าง ฯลฯ. ไฟล์ค่าทรัพยากรของโปรแกรมต่าง ๆ ในระบบโดยรวมจะอยู่ในไดเรกทอรี /usr/X11R6/lib/X11/app-defaults. เช่น ไฟล์ค่าทรัพยากรสำหรับโปรแกรม xterm คือ XTerm, ไฟล์ค่าทรัพยากรสำหรับโปรแกรม xman คือ Xman. ชื่อไฟล์เหล่านี้จริง ๆแล้วคือ*ชื่อคลาส (class name)* ของโปรแกรมที่ทำงานอยู่. แอพพลิเคชันทุกตัวในระบบ X วินโดว์จะมี*ชื่ออินสแตนต์ (instance name)* และชื่อคลาส. ชื่ออินสแตนต์อาจจะมีชื่อที่แตกต่างจากชื่อคลาสหรือเหมือนกับชื่อคลาสก็ได้และสามารถตั้งชื่ออินสแตนต์เองได้ด้วยตัวเลือก -name *instance-name*.

ชื่ออินสแตนต์และชื่อคลาสของโปรแกรมสามารถตรวจสอบได้ด้วยโปรแกรม xprop. โปรแกรม xprop เป็นแอพพลิเคชันมาตรฐานใช้บอกคุณสมบัติของโปรแกรมที่ใช้ในระบบ X วินโดว์. เวลารันโปรแกรม, พอยต์เตอร์ของเมาส์จะเป็นเครื่องหมายบวก. ให้ใช้พอยต์เตอร์คลิ้กหน้าต่างที่ต้องการตรวจสอบก็จะแสดงค่าคุณสมบัติต่าง ๆของหน้าต่างนั้นทางเทอร์มินอลที่รัน xprop.

ตัวอย่างที่ 6.6: ใช้ xprop ดูค่าคุณสมบัติต่าง ๆของหน้าต่าง.

```
$ xprop↵                                          ← แล้วใช้เมาส์คลิ๊กเลือกหน้าต่าง
WM_PROTOCOLS(ATOM): protocols  WM_DELETE_WINDOW
_NET_WM_ICON_GEOMETRY(CARDINAL) = 669, 1000, 105, 24
SM_CLIENT_ID(STRING) = "117f000001000110562372800000078480012"
XKLAVIER_STATE(INTEGER) = 0, 0
_NET_FRAME_EXTENTS(CARDINAL) = 1, 1, 21, 5
_NET_WM_DESKTOP(CARDINAL) = 0
_NET_WM_STATE(ATOM) =
WM_STATE(WM_STATE):
                window state: Normal
                icon window: 0x0
WM_COMMAND(STRING) =  "xman"
WM_LOCALE_NAME(STRING) = "C"
WM_CLASS(STRING) = "topBox", "Xman"          ← ในกรณีนี้ใช้คลิ๊กหน้าต่างหลักของ xman
WM_HINTS(WM_HINTS):
                Client accepts input or input focus: True
                Initial state is Normal State.
                bitmap id # to use for icon: 0x3c00005
WM_NORMAL_HINTS(WM_SIZE_HINTS):
                program specified size: 114 by 71
                window gravity: NorthWest
WM_CLIENT_MACHINE(STRING) = "toybox"
WM_ICON_NAME(STRING) = "Xman"
WM_NAME(STRING) = "xman"
```

ในบรรทัดที่ขึ้นต้นด้วย WM_CLASS จะแสดงชื่ออินสแตนต์และชื่อคลาส. ในตัวอย่างชื่ออินสแตนต์คือ topBox และชื่อคลาสคือ Xman. เพราะว่าชื่อคลาสของโปรแกรมคือ Xman ดังนั้นค่าทรัพยากรต่าง ๆจะเขียนอยู่ในไฟล์ชื่อ Xman ที่อยู่ใต้ไดเรกทอรี /usr/X11R6/lib/X11/app-defaults.

ตัวอย่างที่ 6.7: ไฟล์ค่าทรัพยากรของโปรแกรม GUI ที่ใช้ไลบรารี Xt.

```
$ pwd↵
/usr/X11R6/lib/X11/app-defaults
$ cat XMan↵
```

```
*input:                         True

*topBox:                        True
*topBox.Title:                  Xman
*topBox.IconName:               Xman

*manualBrowser.Title:           Manual Page
*manualBrowser.IconName:        Manual Page
*manualBrowser.geometry:        600x600

*manualFontBold:                -*-courier-bold-r-*-*-*-120-*-*-*-*-*-*
*manualFontItalic:              -*-courier-medium-o-*-*-*-120-*-*-*-*-*-*
--- แสดงผลต่อไปเรื่อยๆ ---
```

ข้อมูลในไฟล์ค่าทรัพยากรเป็นข้อมูลเท็กซ์อ่านเข้าใจได้. โดยมีรูปแบบเป็น

resource-name: *value*

ชื่อทรัพยากรจะเป็นชื่ออินสแตนต์ของวิดเจ็ตที่ใช้ในโปรแกรมบ้าง, ชื่อคุณสมบัติต่าง ๆ. ชื่อเหล่านี้จะใช้เครื่องหมายจุด . เป็นตัวขั้นแสดงโครงสร้าง. ในบางกรณีอาจจะใช้เครื่องหมายดอกจัน * แทนอะไรก็ได้. เช่น *font เป็นค่าทรัพยากรชื่อฟอนต์ที่ต้องการใช้ซึ่งอาจจะเป็นของ topBox หรือ manualBrowser ก็ได้.

วิธีการสำรวจชื่อทรัพยากรของโปรแกรมต่าง ๆอาจจะใช้โปรแกรม editres ช่วย. โปรแกรมนี้จะสามารถแสดงชื่อและค่าทรัพยากรของหน้าต่างที่เลือกเป็นแผนภาพโครงสร้างต้นไม้และยังสามารถตั้งค่าทรัพยากรได้ด้วย.

นอกจากโปรแกรม editres แล้ว, ในระบบ X วินโดว์จะมีคำสั่ง appres ใช้แสดงฐานข้อมูลของทรัพยากรตามชื่อคลาสหรือชื่ออินสแตนต์ที่ระบุเป็นอาร์กิวเมนต์ด้วย.

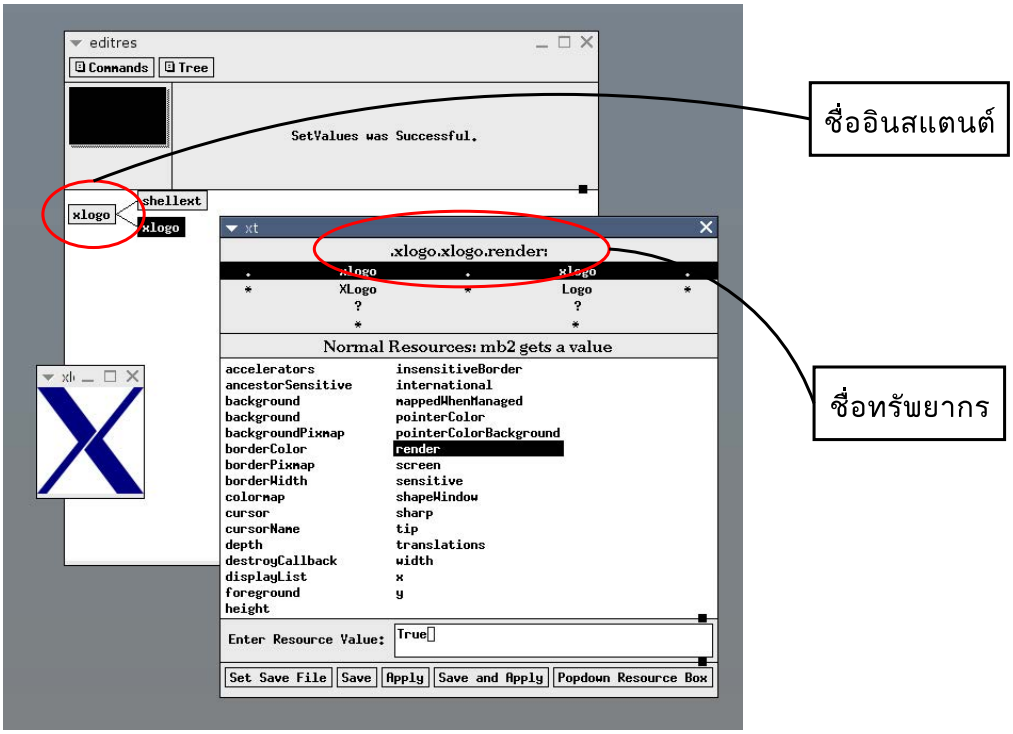
ตัวอย่างที่ 6.8: แสดงฐานข้อมูลทรัพยากรด้วยคำสั่ง appres.

```
$ appres Xman↵
*search*manualPage.Label:       Manual Page
*search*dialog*value:   Xman
*search*dialog.Label:   Type string to search for:
*search*cancel.Label:   Cancel
*search*apropos.Label:  Apropos
*search*label.BorderWidth:      0
*search.Label:  Search
*XmLabelGadget.background:      #eaeaea
*XmLabelGadget.foreground:      #000000
--- แสดงผลต่อไปเรื่อยๆ ---
```

ตั้งค่าทรัพยากรเฉพาะบุคคล

ไฟล์ที่อยู่ในไดเรกทอรี /usr/X11R6/lib/X11/app-defaults เป็นไฟล์ค่าทรัพยากรของโปรแกรมต่าง ๆในระบบ. ถ้าต้องการเปลี่ยนคุณสมบัติต่าง ๆให้แตกต่างจากค่าทรัพยากรที่ระบุในไฟล์เหล่านั้นอาจทำได้โดย

- ใช้ตัวเลือกที่เกี่ยวข้องกับทรัพยากรนั้น



รูปที่ 6.13: โปรแกรม editres แสดงชื่อทรัพยากรและตั้งค่า.

ตัวอย่างเช่น -bg จะเกี่ยวข้องกันทรัพยากร *background เป็นต้น. แต่ทรัพยากรทุกตัวไม่สามารถตั้งค่าผ่านทางตัวเลือกได้ทั้งหมดทุกตัว.

- ใช้ตัวเลือก -xrm "*resource-name: value*"
  ชื่อทรัพยากรและค่าสามารถเขียนเป็นอาร์กิวเมนต์ของตัวเลือก -xrm ได้เช่น -xrm '*background: black'.

- ไฟล์ .Xdefaults หรือ .Xresources ในโฮมไดเรกทอรี
  โดยปรกติ, เวลา X เซิร์ฟเวอร์เริ่มทำงานจะมีสคริปต์ตั้งค่าเริ่มต้นทั้งสำหรับระบบและระดับบุคคล. ถ้าเป็นสคริปต์สำหรับระบบโดยรวมแล้วจะใช้คำสั่ง xrdb ซึ่งเป็นคำสั่งอ่านฐานข้อมูลทรัพยากรจากไฟล์ที่เป็นอาร์กิวเมนต์. และไฟล์ที่อ่านมักจะใช้ชื่อเป็น .Xdefaults หรือ .Xresources ในโฮมไดเรกทอรี. ส่วนจะใช้ไฟล์ .Xdefaults หรือ .Xresources นั้นแล้วแต่ระบบ.

- ไฟล์ .Xdefaults-*hostname* ในโฮมไดเรกทอรี
  เวลารันโปรแกรมที่ใช้ไลบรารี Xt จะอ่านไฟล์ค่าทรัพยากรจากไฟล์ .Xdefaults-*hostname* ทุกครั้ง. ถ้าต้องการเปลี่ยนค่าทรัพยากรของแอพพลิเคชันต่าง ๆให้กำหนดในไฟล์นี้โดยไม่ต้องเรียกคำสั่ง xrdb เหมือนกับไฟล์ .Xdefaults หรือ .Xresources.

ไฟล์ .Xdefaults, .Xresources และ .Xdefaults-*hostname* จะมีรูปแบบคล้ายกับไฟล์ค่าทรัพยากรที่อยู่ในไดเรกทอรี app-defaults. สามารถระบุค่าทรัพยากรของโปรแกรมต่าง ๆได้ในไฟล์เดียวกัน.

สร้างไฟล์ .Xdefault-*hostname*

เนื่องจากไฟล์ ~/.Xdefaults-*hostname* เป็นไฟล์ที่โปรแกรมที่ใช้ไลบรารี Xt อ่านค่าทรัพยากร. ดังนั้นถ้าต้องการปรับคุณสมบัติของโปรแกรมต่าง ๆให้ใช้กำหนดในไฟล์นี้.

วิธีการสร้างไฟล์ .Xdefaults-*hostname* สามารถทำได้โดยคัดลอกบรรทัดต่าง ๆ จากไฟล์ที่อยู่ในไดเรกทอรี app-defaults แล้วเติมชื่อคลาสของโปรแกรมที่ต้องการตั้งค่า. ตัวอย่างเช่นถ้าต้องการตั้งค่าทรัพยากรเฉพาะของโปรแกรม xterm ให้คัดลอกเนื้อหาของไฟล์ XTerm มาไว้ใน ~/.Xdefaults-*hostname* แล้วเติมชื่อคลาส (XTerm) หรือชื่ออินสแตนต์ (xterm) ต้นบรรทัดทุกบรรทัดที่คัดลอกมา. จากนั้นก็ปรับแต่งค่าตามที่ต้องการ.

วิธีอีกแบบและสะดวกกว่าคือการใช้คำสั่ง appres ช่วยแสดงชื่อและค่าทรัพยากรของโปรแกรม (อินสแตนต์) ที่ต้องการ, เก็บผลลัพธ์ไว้ในไฟล์ .Xdefaults-*hostname*.

ตัวอย่างที่ 6.9: สร้างไฟล์ .Xdefault-hostname สำหรับ xterm โดยใช้ appres ช่วย.

```
$ appres xterm | grep '^xterm' >> .Xdefaults-`hostname`↵
$ tail ~/.Xdefault-`hostname`↵
xterm*XmLabel.background:       #eaeaea
xterm*XmLabel.foreground:       #000000
xterm*XmTearOffButtonGadget.background: #eaeaea
xterm*XmTearOffButtonGadget.foreground: #000000
xterm*beNiceToColormap: false
xterm*ScrollbarBackground:      #eaeaea
xterm*foreground:       #000000
xterm*borderColor:      black
xterm*background:       #eaeaea
xterm*ShapeStyle:       Rectangle
```

อาร์กิวเมนต์ของ appres คือชื่ออินสแตนต์ซึ่งในกรณีได้แก่ xterm. คำสั่ง appres จะแสดงทรัพยากรทั่ว ๆไปด้วยซึ่งจะไม่มีชื่ออินสแตนต์นำหน้า. ดังนั้นจึงใช้คำสั่ง grep กรองเอาเฉพาะบรรทัดที่มีชื่ออินสแตนต์เก็บไว้. หลังจากนั้นผู้ใช้สามารถแก้ไขไฟล์ .Xdefault-*hostname* ด้วยบรรณาธิกรณ์ตามต้องการ. สำหรับชื่อทรัพยากรอื่น ๆที่ไม่ได้เขียนไว้ในไฟล์, สามารถเพิ่มเติมได้โดยใช้โปรแกรม editres ช่วย.

อาร์กิวเมนต์ของคำสั่ง appres เป็นได้ทั้งชื่ออินสแตนต์และชื่อคลาสซึ่งจะให้ผลไม่เหมือนกัน. ถ้าจะอาร์กิวเมนต์เป็นชื่อคลาส, ต้องเปลี่ยนคำสั่งที่ใช้สร้างไฟล์ .Xdefaults-*hostname* ใช้ sed ช่วยเพิ่มชื่อคลาสที่ต้นบรรทัดทุกบรรทัด.

ตัวอย่างที่ 6.10: สร้างไฟล์ .Xdefault-hostname โดยใช้ชื่อคลาสแทนชื่ออินสแตนต์.

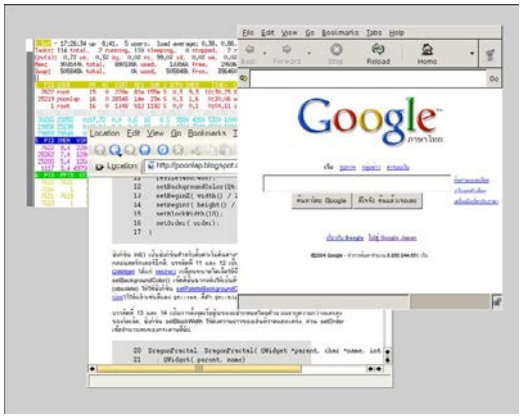
```
$ appres XTerm | sed -e 's/^\(\*.*\)$/XTerm\1/' >> ~/.Xdefault-`hostname`↵
```

ชื่อคลาสที่ระบุในไฟล์ .Xdefaults-*hostname* มีความสัมพันธ์กับตัวเลือกทั่วไป -name *instance-name*. โดยปรกติถ้าสั่งคำสั่ง xterm, ชื่ออินสแตนต์ของโปรแกรมจะเป็น xterm. ดังนั้นจะใช้ค่าทรัพยากรที่ขึ้นต้นด้วย xterm. ถ้ามีการใช้ตัว

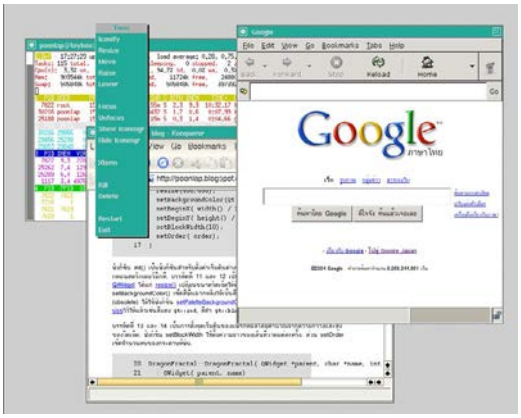
เลือก `-name` *`instance-name`* เช่น `xterm -name xterm-big`, โปรแกรมที่รันจะใช้ค่าทรัพยากรที่ขึ้นต้นด้วย xterm-big. ถ้าเราเตรียมค่าทรัพยากรชื่อ xterm-big และปรับแต่งให้ใช้ฟอนต์ตัวใหญ่, เวลารันคำสั่ง `xterm -name xterm-big` ก็จะได้โปรแกรม `xterm` ที่มีคุณสมบัติต่างจาก `xterm` ธรรมดา.

6.2.11 วินโดว์แมนเนเจอร์

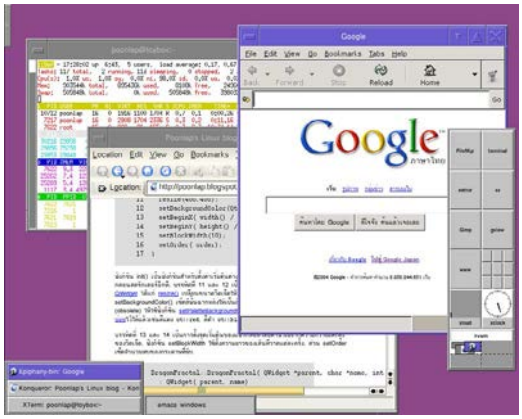
หน้าที่หนึ่งของ X เซิร์ฟเวอร์ได้แก่การสร้างหน้าและวาดรูปต่าง ๆเพื่อแสดงผลตามคำร้องขอจากไคลเอ็นต์. X เซิร์ฟเวอร์ไม่มีหน้าที่จัดตำแหน่งหรือขนาดว่าหน้าต่างใหม่ของโปรแกรมควรจะอยู่ตรงไหนอย่างไร. หน้าที่เหล่านี้เป็นหน้าที่ของโปรแกรมที่เรียกว่า*วินโดว์แมนเนเจอร์ (windows manager)*. ในกรณีที่ไม่มีวินโดว์แมนเนเจอร์เวลาโปรแกรมต่าง ๆแสดงผลจะไม่มีกรอบควบคุม. ผู้ใช้ไม่สามารถเปลี่ยนขนาดหรือเลื่อนตำแหน่งของโปรแกรมที่ไม่มีวินโดว์แมนเนเจอร์ได้เลยในรูปที่ 6.14(ก).



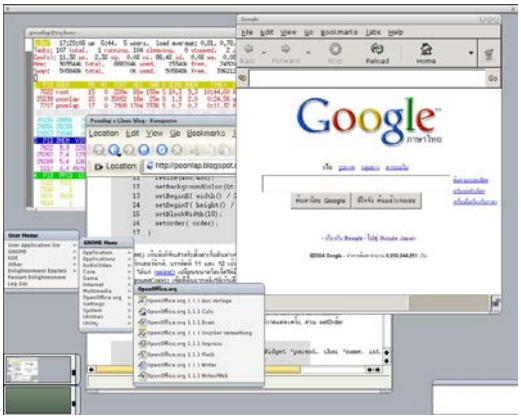
(ก) ไม่มีวินโดว์แมนเนเจอร์



(ข) วินโดว์แมนเนเจอร์ twm



(ค) วินโดว์แมนเนเจอร์ fvwm



(ง) วินโดว์แมนเนเจอร์ enlightenment

รูปที่ 6.14: วินโดว์แมนเนเจอร์แบบต่าง ๆ

วินโดว์แมนเนเจอร์ดั้งเดิมได้แก่ *twm (Tab Window Manager)* ที่แสดงในรูปที่ 6.14(ข) และ *mwm (Motif Window Manager)*. วินโดว์แมนเนเจอร์นี้เรียบง่ายเพราะ

เป็นวินโดว์แมนเนเจอร์ในยุคแรก ๆ. อย่างน้อยผู้ใช้สามารถควบคุมตำแหน่งและขนาดหน้าต่างของโปรแกรมต่าง ๆได้. นอกจากนั้นยังมีเมนูง่าย ๆให้ใช้เมื่อคลิ้กพื้นหลังด้วย. ต่อมามีการเพิ่มคุณสมบัติต่าง ๆให้กับวินโดว์แมนเนเจอร์เช่นเพิ่มเมนูเรียกใช้โปรแกรมต่าง ๆ ที่เตรียมไว้, *ทาสก์บาร์ (taskbar)*, *รายการวินโดว์ (window list)*, *เดสก์ท็อปเสมือน (virtual desktop)* ฯลฯ. วินโดว์แมนเนเจอร์ที่อาจจะเรียกได้ว่าเป็นยุคที่สองต่อจาก twm และ mwm ได้แก่ *fvwm (F* Virtual Window Manager)* (รูปที่ ??(ค)) และหลังจากนั้นมีผู้สร้างวินโดว์แมนเนเจอร์อื่น ๆให้มีคุณสมบัติมากขึ้น, สวยขึ้น, ดีขึ้นไปเรื่อย ๆเช่น Afterstep, Enlightenment, Sawfish, Fluxbox, Metacity ฯลฯ.

virtual desktop ▶ *เดสก์ท็อปเสมือน*. คุณสมบัติของวินโดว์แมนเนเจอร์ที่สามารถขยายเดสก์ท็อปให้มีหลายหน้าทำให้เนื้อที่การใช้งานเดสก์ท็อปกว้างขึ้น. แต่ในความเป็นจริงแล้วมีหน้าจอเพียงหน้าเดียว.

Fvwm เป็นการเล่นคำคือให้ตัวอักษร F แทนคำอะไรก็ได้ที่ขึ้นต้นด้วย F ตามแต่จะชอบ.

6.2.12 สภาพแวดล้อมเดสก์ท็อป

การมีวินโดว์แมนเนเจอร์ช่วยให้การทำงานแบบกราฟิกโดยเฉพาะด้านเดสก์ท็อปมีความสะดวกมากขึ้น. สำหรับผู้ที่ต้องการใช้เชลล์ก็เรียกใช้เทอร์มินอลเอมิวเลเตอร์มาตรฐานได้แก่ `xterm` ซึ่งจะจำลองหน้าจอแสดงผลแบบเท็กซ์และเรียกใช้เชลล์ในโปรแกรม. ถึงแม้ว่าว่าวินโดว์แมนเนเจอร์จะอำนวยความสะดวกต่าง ๆให้ผู้ใช้แต่ก็มีปัญหาบางอย่างเช่น

- โปรแกรม GUI สามารถเลือกไลบรารีทูลล์คิตที่ได้ตามต้องการ. รูปแบบของอินเทอร์เฟสของแต่ละทูลล์คิตมีความแตกต่างกันไม่เป็นหนึ่งเดียว.
- วินโดว์แมนเนเจอร์มีให้เลือกใช้มากเกินไปเกิดความสับสนสำหรับผู้ใช้ทั่วไป. บางระบบอาจจะใช้วินโดว์แมนเนเจอร์ที่ไม่เหมือนกันทำให้ผู้ใช้ต้องปรับตัว. ในกรณีเป็นความน่ารำคาญที่ต้องเรียนรู้ใหม่.
- ขาดความสามารถสื่อสารระหว่างโปรแกรม. เช่นไม่สามารถ*แดรกแอนด์ดรอป (drag & drop)* ซึ่งมีความจำเป็นสำหรับระบบเดสก์ท็อปชั้นสูง.

เนื่องจากปัญหาเหล่านี้จึงมีการสร้างระบบเดสก์ท็อปที่เรียกว่า*สภาพแวดล้อมเดสก์ท็อป (Desktop Environment)* หรือเรียกย่อ ๆว่า *DE* ทำให้เดสก์ท็อปที่ใช้ดูมีความเป็นหนึ่งเดียว, จัดเตรียมไอคอนเป็นชุดที่มีความเข้ากันได้, มีโปรแกรมสำคัญ ๆสำหรับงานเดสก์ท็อป, *ธีม (theme)*, วินโดว์แมนเนเจอร์, ความสามารถแดรกแอนด์ดรอป ฯลฯ. สภาพแวดล้อมเดสก์ท็อปที่นิยมใช้ในลินุกซ์ได้แก่ *Gnome (GNU Network Object Model Environment)* และ *KDE (K Desktop Environment)*.

6.3 ติดตั้งและปรับแต่ง X เซิร์ฟเวอร์

ระบบที่ต้องการใช้ X วินโดว์ต้องติดตั้งโปรแกรม X เซิร์ฟเวอร์ก่อน. โดยปรกติเวลาติดตั้งลินุกซ์มักจะมีเมนูให้เลือกติดตั้งและตั้งค่าเริ่มโดยอัตโมมัติ. หรือถ้าไม่ได้ติดตั้งแต่ตอนแรกก็สามารถติดตั้งแพ็กเกจที่เกี่ยวกับ X วินโดว์ภายหลังได้.

โปรแกรมเซิร์ฟเวอร์รวมถึงไคลเอ็นต์มาตรฐานในระบบ X วินโดว์มักของในไดเรกทอรี `/usr/X11R6/bin`. ชื่อโปรแกรมเซิร์ฟเวอร์คือ `X` ซึ่งเป็นชื่อทั่วไปและจะเป็นซิมโบลิกลิงก์ไปหาไฟล์โปรแกรมเซิร์ฟเวอร์ตัวจริง. ไฟล์ปรับแต่งค่าเริ่มต้นของของ X เซิร์ฟ

ไคลเอ็นต์มาตรฐานหมายถึงโปรแกรมที่มาพร้อมกับแพ็กเกจ X วินโดว์.

เวอร์โดยปรกติจะคือไฟล์ /etc/X11/xorg.conf. สำหรับบางดิสทริบิวที่ยังใช้ X เ ซิร์ฟเวอร์ของโปรเจค XFree86 จะเป็นไฟล์ /etc/X11/XF86Config. ข้อมูลของไฟล์ทั้งสองฉบับนี้มีความเข้ากันได้และไม่แตกต่างกันมากนัก. ในที่นี้จะอธิบายการสร้างไฟล์ตั้งค่าเริ่มต้นของ X เซิร์ฟเวอร์โดยสมมติว่าในระบบยังไม่มีไฟล์ xorg.conf. ผู้ใช้มีความจำเป็นที่ต้องรู้ข้อมูลเกี่ยวกับฮาร์ดแวร์ต่าง ๆที่จะประกอบกันเป็นระบบ X วินโดว์ ได้แก่

- **วิดีโอการ์ด (video card)**
  วิดีโอการ์ดหรือกราฟิกการ์ดเป็นส่วนสำคัญสำหรับระบบ X วินโดว์. ผู้ใช้ควรจะรู้ข้อมูลพื้นฐานได้แก่ บรษัทผู้ผลิต, ชื่อรุ่น, จำนวนหน่วยความจำของวิดีโอการ์ด เป็นต้น. วิดีโอการ์ดที่สามารถใช้ได้กับ X เซิร์ฟเวอร์สามารถตรวจสอบได้จากเว็บไซต์ของ XFree86 หรือจากโปรแกรมช่วยติดตั้งเช่น xorgcfg และ xorgxonfig.

- **จอภาพ (monitor)**
  ผู้ใช้ควรรู้ว่าจอภาพที่ใช้มี*ช่วงความถี่ตามแนวนอน (horizontal sync frequency)* และ*ช่วงความถี่ตามแนวตั้ง (vertical sync)* เท่าไร. ค่าเหล่านี้อาจจะดูได้จากเลเบลที่ติดไว้หลังจอภาพหรือจากเอกสารคู่มือการใช้จอภาพ.

- **แป้นพิมพ์ (keyboard)**
  แป้นพิมพ์เป็นส่วนที่มีปัญหาน้อยที่สุดเมื่อเทียบกับฮาร์ดแวร์อื่น ๆ. แป้นพิมพ์ที่ใช้อาจจะเป็นแบบ *PS/2* หรือแบบ USB (Universal Serial Bus) ก็ได้. สิ่งที่ต้องเลือกเวลาติดตั้ง X วินโดว์คือประเภทของแป้นพิมพ์เช่นแป้นพิมพ์ของคอมพิวเตอร์ส่วนบุคคลแบบ 101 คีย์ (Generic 101-key PC) ซึ่งมักใช้กันทั่วไปหรือประเภทอื่น ๆ. นอกจากนั้นสามารถเลือก*เลย์เอาท์ (layout)* ของแป้นพิมพ์เช่นแป้นพิมพ์ภาษาไทยได้โดยใช้โมดูล XKB (X Keyboard Extension).

- **เมาส์ (mouse)**
  เมาส์ที่ใช้ในระบบ X วินโดว์เป็นแบบ PS/2, Serial หรือ USB. ผู้ใช้ต้องเลือกรายละเอียดปลีกย่อยเช่นประเภทของเมาส์, จำลองปุ่มกลางถ้าเป็นเมาส์แบบสองปุ่ม, เลือกดีไวซ์ไฟล์ที่เมาส์ต่ออยู่เป็นต้น.

PS/2 ▶
ย่อมาจากคำว่า Personal System 2 เป็นอินเทอร์เฟสมาตรฐานสำหรับต่ออุปกรณ์แป้นพิมพ์หรือเมาส์สำหรับคอมพิวเตอร์ส่วยบุคคล.

USB ▶
คำย่อของ Universal Serial Bus เป็นมาตรฐานสำหรับต่ออุปกรณ์คอมพิวเตอร์ต่าง ๆตั้งแต่แป้นพิมพ์, เมาส์, เครื่องพิมพ์, ฮาร์ดดิสก์ ฯลฯ.

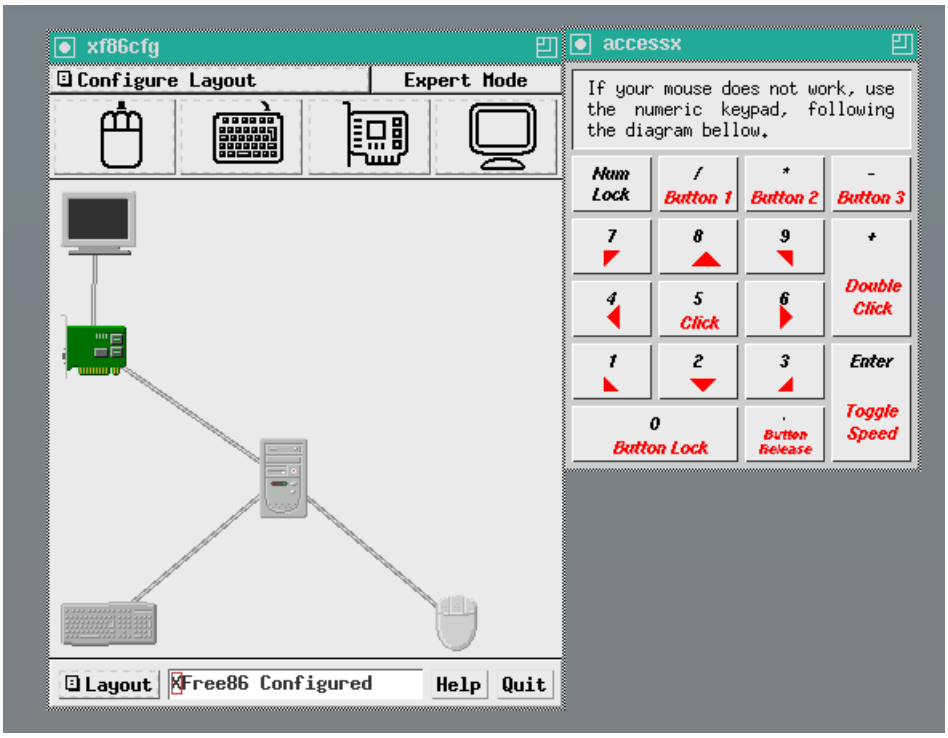
Serial ▶
มาตรฐานสำหรับสื่อสาร. ด้านหลังของคอมพิวเตอร์ส่วนบุคคลโดยทั่วไปจะมีซีเรียลอินเทอร์เฟส RS-232 อยู่.

ในแพกเกจของ X เซิร์ฟเวอร์จะมีโปรแกรมสำหรับช่วยสร้างไฟล์ xorg.conf ได้แก่โปรแกรม xorgcfg, xorgconfig และตัวเซิร์ฟเวอร์ X.

6.3.1 xorgcfg

โปรแกรม xf86cfg (xf86cfg) เป็นโปรแกรมแบบ GUI สำหรับสร้างไฟล์ตั้งค่าเริ่มต้น xorg.conf ใหม่หรือแก้ไขไฟล์ xorg.conf ที่มีอยู่แล้ว. ถ้าโปรแกรมนี้รันอยู่ในเท็กซ์โหมดเพื่อสร้างไฟล์ตั้งค่าเริ่มต้น, ตัวโปรแกรมจะรันคำสั่ง X -configure อีกต่อหนึ่งเพื่อสร้างไฟล์ตั้งค่าเริ่มต้นชื่อ xorg.conf.new ในโฮมไดเรกทอรีของผู้ใช้โดยอัตโนมัติ. และถ้าไฟล์ตั้งค่าเริ่มต้นที่สร้างสามารถใช้งานได้, ก็จะรัน X เซิร์ฟเวอร์และ

แสดงหน้าจอในรูปที่ 6.15. ถ้าสามารถใช้เมาส์ได้, ให้ใช้เมาส์เลือกเมนูต่าง ๆเพื่อปรับค่าต่าง ๆ. ถ้าไม่สามารถใช้เมาส์ได้แสดงว่าไฟล์ตั้งค่าเริ่มต้นที่สร้างโดยอัตโนมัติมีข้อผิดพลาด. แต่ผู้ใช้ยังสามารถดำเนินการต่อไปได้โดยใช้*คีย์แพด (keypad)* แทนเมาส์เลื่อนพอยต์เตอร์หรือคลิ้กที่หน้าต่างแทนเมาส์ได้. เมื่อปรับแต่งค่าต่าง ๆเรียบร้อยแล้วให้จบการทำงานของโปรแกรมโดยกดปุ่ม “Quit”. โปรแกรมจะบันทึกค่าไฟล์ตั้งค่าเริ่มต้นให้โดยอัตโนมัติแล้วกลบสู่เท็กซ์โหมดอีกครั้ง.



รูปที่ 6.15: โปรแกรม xorgcfg (xf86cfg) ปรับแต่ง X เซิร์ฟเวอร์และตั้งค่าเริ่มต้น.

จะมีบางกรณีที่โปรแกรมไม่สามารถทำงานได้เพราะไม่สามารถสร้างไฟล์ตั้งค่าเริ่มต้นได้ถูกต้องแต่แรก. เช่นหน้าจอดำแต่ไม่มีอะไรปรากฏ, ในกรณีนี้ให้กดคีย์ Ctrt+Alt+BackSpace เพื่อจบการทำงาน. หรือถ้าไม่โปรแกรมไม่สามารถรันได้แล้วกลับสู่เชลล์พรอมต์, อาจจะต้องแก้ไฟล์ xorg.conf.new ด้วยตัวเองแล้วรันคำสั่ง xorgcfg กับตัวเลือก -config *$HOME/xorg.conf.new* ใหม่อีกครั้ง. ถ้าไม่สำเร็จ, ให้ใช้วิธีอื่น.

6.3.2 xorgconfig

โปรแกรม xorgconfig (xf86config) เป็นโปรแกรมบรรทัดคำสั่งแสดงคำถามหรือตัวเลือกต่าง ๆเกี่ยวกับฮาร์ดแวร์ในระบบ X วินโดว์และให้ผู้ใช้เลือกตอบ. โปรแกรมนี้ใช้รันในเท็กซ์โหมดและลองรัน X เซิร์ฟเวอร์หลังจากที่สร้างไฟล์แล้ว. ถ้าโปรแกรมสร้างไฟล์ตั้งค่าเริ่มต้นให้ไม่ถูกต้องอาจจะต้องแก้ไขต่อด้วยการใช้บรรณาธิกรณ์เปิดไฟล์ /etc/X11/xorg.conf แก้ไขเอง.

ตัวอย่างที่ 6.11: ขั้นตอนการทำงานของโปรแกรม xorg.conf

```
# xorgconfig↵
This program will create a basic xorg.conf file, based on menu selections you
make.

The xorg.conf file usually resides in /usr/X11R6/etc/X11 or /etc/X11. A sample
xorg.conf file is supplied with Xorg; it is configured for a standard
VGA card and monitor with 640x480 resolution. This program will ask for a
pathname when it is ready to write the file.

You can either take the sample xorg.conf as a base and edit it for your
configuration, or let this program produce a base xorg.conf file for your
configuration and fine-tune it.

Before continuing with this program, make sure you know what video card
you have, and preferably also the chipset it uses and the amount of video
memory on your video card. SuperProbe may be able to help with this.

Press enter to continue, or ctrl-c to abort. ↵

First specify a mouse protocol type. Choose one from the following list:

 1.  Auto
 2.  SysMouse
 3.  MouseSystems
 4.  PS/2
 5.  Microsoft
--- แสดงผลต่อไปเรื่อยๆ ---
```

6.3.3 X

โปรแกรมเซิร์ฟเวอร์ X สามารถสร้างไฟล์ตั้งค่าเริ่มต้นให้โดยอัตโนมัติถ้ารันเซิร์ฟเวอร์ด้วยตัวเลือก `-configure`. คำสั่งนี้ต้องรันด้วย root และจะสร้างไฟล์ตั้งค่าเริ่มต้นชั่วคราว `xorg.conf.new` ไว้ในโฮมไดเรกทอรีของ root.

ตัวอย่างที่ 6.12: ใช้เซิร์ฟเวอร์เขียนไฟล์ตั้งค่าเริ่มต้น.

```
# X -configure↵
_XSERVTransSocketOpenCOTSServer: Unable to open socket for inet6
_XSERVTransOpen: transport open failed for inet6/toybox:1
_XSERVTransMakeAllCOTSServerListeners: failed to open listener for inet6

This is a pre-release version of the The X.Org Foundation X11.
It is not supported in any way.
--- แสดงผลต่อไปเรื่อยๆ ---
Markers: (--) probed, (**) from config file, (==) default setting,
        (++) from command line, (!!) notice, (II) informational,
        (WW) warning, (EE) error, (NI) not implemented, (??) unknown.
(==) Log file: "/var/log/Xorg.0.log", Time: Fri Jan 21 15:17:03 2005
--- แสดงผลต่อไปเรื่อยๆ ---
        dummy
        fbdev
        v4l
(++) Using config file: "/root/xorg.conf.new"
```

```
Xorg detected your mouse at device /dev/mouse.
Please check your config if the mouse is still not
operational, as by default Xorg tries to autodetect
the protocol.

Your xorg.conf file is /root/xorg.conf.new

To test the server, run 'X -config /root/xorg.conf.new'
```

ตัวโปรแกรมจะแสดงข้อความต่าง ๆทางหน้าจอ. ข้อความที่สำคัญ ๆได้แก่

- บรรทัดที่มีคำว่า "Log file" แสดงชื่อล็อกไฟล์ซึ่งในกรณีนี้คือ `/var/log/Xorg.0.log`. เซิร์ฟเวอร์จะสร้างล็อกไฟล์ทุกครั้งเมื่อทำงานและมีประโยชน์สำหรับตรวจสอบข้อผิดพลาดหรือข้อมูลที่เกี่ยวกับตัวเซิร์ฟเวอร์. ถ้าเกิดข้อผิดพลาดทำให้ X เซิร์ฟเวอร์ไม่สามารถทำงานได้, เราสามารถใช้คำสั่ง `less` เปิดล็อกไฟล์ดูแล้วหาบรรทัดที่มีคำว่า "EE" ซึ่งเป็นบรรทัดบ่งบอกสาเหตุของข้อผิดพลาด. เมื่อแก้ไขข้อผิดพลาดนั้นแล้วก็จะสามารถรัน X เซิร์ฟเวอร์ได้.

- บรรทัดสุดท้ายบอกวิธีทดสอบการทำงานของเซิร์ฟเวอร์โดยสั่งคำสั่ง `X -config /root/xorg.conf.new`. ถ้า X เซิร์ฟเวอร์ทำงานได้หน้าจอจะเปลี่ยนเป็นกราฟฟิคโหมดและแสดงพอยเตอร์ของเมาส์. ถ้าสามารถขยับเลื่อนเมาส์ได้ก็แสดงว่า X เซิร์ฟเวอร์ทำงานได้ถูกต้อง. ให้กดคีย์ `Ctrl+Alt+Del` จบการทำงานของ X เซิร์ฟเวอร์.

  มีบางกรณีที่ X เซิร์ฟเวอร์ทำงานได้แต่ทำงานได้ไม่สมบูรณ์เช่นแสดงพอยเตอร์ของเมาส์แต่ไม่สามารถขยับได้, และเป็นหน้าจอดำไม่สามารถทำอะไรต่อได้. ปัญหานี้เป็นเพราะ X เซิร์ฟเวอร์ไม่สามารถปรับแต่งค่าได้ถูกต้องร้อยเปอร์เซ็นต์. ในกรณีแรกอาจจะมีสาเหตุจากชื่อไฟล์ดีไวซ์ที่เมาส์ใช้หรือประเภทของเมาส์ไม่ถูกต้อง. ในกรณีหลังอาจจะเป็นเพราะจอภาพไม่สามารถแสดงผลได้เนื่องจากค่าของความถี่, *ความละเอียดของหน้าจอ (resolution)*ไม่ตรงกัน เป็นต้น. ไม่ว่าในกรณีใด ๆ, ให้ใช้บรรณาธิกรณ์เปิดไฟล์ `xorg.conf.new` แก้ไขส่วนที่เกี่ยวข้องแล้วลองใหม่จนกว่าจะสำเร็จ.

ถ้าไม่มีปัญหาใด ๆให้ย้ายไฟล์ `xorg.conf.new` ไปไว้ที่ไดเรกทอรี `/etc/X11` โดยเปลี่ยนชื่อเป็น `xorg.conf`.

6.4 ไฟล์ xorg.conf

ไฟล์ `/etc/X11/xorg.conf` เป็นไฟล์สำคัญที่กำหนดค่าต่างทำให้เซิร์ฟเวอร์ใช้งานได้ถูกต้อง. สำหรับเซิร์ฟเวอร์ของโครงการ XFree86 จะมีชื่อเป็น `XF86Config`. ไฟล์นี้จะแบ่งเป็นเซกชัน (section) ต่าง ๆแสดงในตารางที่ 6.2

ตัวอย่างที่ 6.13 แสดงไฟล์ `xorg.conf` ที่สร้างด้วยคำสั่ง `X -configure` และปรับแต่งให้ใช้งานได้จริง. โดยปรกติ, แพ็กเกจ X เซิร์ฟเวอร์จะมีตัวอย่างไฟล์ชื่อ

ตารางที่ 6.2: เซกชันต่าง ๆในไฟล์ xorg.conf

ชื่อเซกชัน	คำอธิบาย
ServerLayout	นิยามโครงสร้างสำคัญที่ประกอบเป็นเซิร์ฟเวอร์.
Files	ชื่อพาท (path) ไดเรกทอรีข้อมูลต่าง ๆเช่นฟอนต์, โมดูล ฯลฯ.
Module	โมดูลที่ต้องการใช้กับ X เซิร์ฟเวอร์.
InputDevice	นิยามดีไวซ์เช่นแป้นพิมพ์และเมาส์.
Monitor	นิยามค่าคุณสมบัติของจอภาพที่ใช้.
Device	ค่าคุณสมบัติเกี่ยวกับกราฟิกการ์ด.
Screen	นิยามค่าคุณสมบัติของหน้าจอแสดงผลเชิงซอฟต์แวร์.
ServerFlags	ปรับแต่งคุณสมบัติโดยรวมของ X เซิร์ฟเวอร์.

/etc/X11/xorg.conf.example (/etc/X11/XF86Config.example) มาให้ด้วยดูเป็นตัวอย่าง. ไฟล์ตัวอย่างนั้นไม่สามารถเอาใช้จริงได้แต่มีประโยชน์ไว้ศึกษาเพราะมีคอมเมนต์อธิบายส่วนต่าง ๆไว้ด้วย.

ตัวอย่างที่ 6.13: ไฟล์ /etc/xorg.conf

```
 1  Section "ServerLayout"
 2      Identifier     "XFree86 Configured"
 3      Screen         "Screen0"
 4      InputDevice    "Mouse0" "CorePointer"
 5      InputDevice    "Keyboard0" "CoreKeyboard"
 6  EndSection
 7
 8  Section "Files"
 9      # For XFS, uncomment this and comment the others
10      # FontPath  "unix/:-1"
11      RgbPath      "/usr/X11R6/lib/X11/rgb"
12      ModulePath   "/usr/X11R6/lib/modules"
13      FontPath     "/usr/share/fonts/misc/"
14      FontPath     "/usr/share/fonts/Speedo/"
15      FontPath     "/usr/share/fonts/Type1/"
16      FontPath     "/usr/share/fonts/CID/"
17      FontPath     "/usr/share/fonts/75dpi/"
18      FontPath     "/usr/share/fonts/100dpi/"
19  EndSection
20
21  Section "Module"
22      Load  "extmod"
23      Load  "dri"
24      Load  "dbe"
25      Load  "record"
26      Load  "xtrap"
27      Load  "glx"
28      Load  "speedo"
29      Load  "type1"
30  EndSection
31
```

```
32  Section "InputDevice"
33      Identifier  "Keyboard0"
34      Driver      "kbd"
35
36      Option     "XkbModel"      "pc101"
37      Option     "XkbLayout"     "us,th_tis"
38      Option     "XkbOptions"    "grp:alt_shift_toggle,grp_led:scroll,
lv3:ralt_switch"
39  EndSection
40
41  Section "InputDevice"
42      Identifier  "Mouse0"
43      Driver      "mouse"
44      Option      "Protocol" "IMPS/2"
45      Option      "Device" "/dev/mouse"
46      Option      "Buttons" "5"
47      Option      "ZAxisMapping" "4 5"
48  EndSection
49
50  Section "Monitor"
51      DisplaySize   380   310 # mm
52      Identifier   "Monitor0"
53      VendorName   "DEL"
54      ModelName    "DELL 1901FP"
55      Option      "DPMS"
56  EndSection
57
58  Section "Device"
59      Identifier  "Card0"
60      Driver      "nv"
61      VendorName  "nVidia Corporation"
62      BoardName   "Unknown Board"
63      BusID       "PCI:1:0:0"
64  EndSection
65
66  Section "Screen"
67      Identifier "Screen0"
68      Device     "Card0"
69      Monitor    "Monitor0"
70      DefaultDepth 24
71      SubSection "Display"
72          Depth     16
73          Modes "1280x1024" "1024x768" "800x600"
74      EndSubSection
75      SubSection "Display"
76          Depth     24
77          Modes "1280x1024" "1024x768" "800x600"
78      EndSubSection
79  EndSection
```

เซกชันละส่วนจะเริ่มต้นด้วยคำว่า Section และลงท้ายด้วยคำว่า EndSection. ทุกเซกชันจะมีค่า Identifier กำหนดชื่อเฉพาะของแต่ละเซกชัน. ในแต่ละเซกชันสามารถมีเซกชันย่อยแยกออกไปอีกเป็นซับเซกชัน (subsection). ส่วนที่เป็นซับเซกชันจะเริ่มต้นด้วยคีย์เวิร์ด SubSection และจบด้วย EndSubSection.

ในเซกชันหนึ่ง ๆจะมีชื่อคุณสมบัติและค่าที่ต้องการเซ็ต. ชื่อคุณสมบัติสำหรับแต่ละเซกชันเหล่านี้อธิบายอยู่ใน `man 5 xorg.conf` (`XF86Config`) ว่ามีอะไรบ้างและอธิบายประเภทของคุณสมบัติต่าง ๆว่าต้องเป็นสายอักขระหรือตัวเลข ฯลฯ.

6.4.1 เซกชัน ServerLayout

เซกชัน ServerLayout เป็นช่วงนิยามโครงสร้างส่วนประกอบของ X เซิร์ฟเวอร์และสามารถมีได้มากกว่าหนึ่งส่วน. ปรกติจะมี ServerLayout เดียวเท่านั้น.

- Identifier "*name*"
  ใช้กำหนดชื่อเฉพาะสำหรับแยกแยะเซกชัน.

- Screen "*screenid*"
  ระบุชื่อสกรีนที่ต้องการใช้แสดงผล. สามารถตั้งค่าได้มากกว่าหนึ่งตัวในกรณีที่ต้องการแสดงผลหลายสกรีนและฮาร์ดแวร์รองรับ. ในตัวอย่างบอกให้เซิร์ฟเวอร์ใช้สกรีนที่มีชื่อ (Identifier) เป็น "Screen0".

- InputDevice "*input-dev-id*" "*option*"
  ระบุดีไวซ์สำหรับข้อมูลนำเข้าเช่นแป้นพิมพ์และเมาส์.

ในเซกชันนี้อย่างน้อยต้องกำหนดข้อคุณสมบัติ Identifier และ Screen เพื่อให้ X วินโดว์ทำงานได้.

6.4.2 เซกชัน Files

เป็นช่วงที่ระบุชื่อไดเรกทอรีหรือไฟล์พาทเกี่ยวฐานข้อมูลต่าง ๆเช่นไฟล์ฐานข้อมูลสี, ไดเรกทอรีที่เก็บฟอนต์ หรือ ไดเรกทอรีที่เก็บโมดูลของ X เซิร์ฟเวอร์. ค่าของพาทที่ระบุในที่นี้สามารถระบุเปลี่ยนได้จากตัวเลือกเวลารันโปรแกรม X เซิร์ฟเวอร์.

- RgbPath "*rgb-file*"
  ระบุชื่อไฟล์ฐานข้อมูลสี. ในตัวอย่างได้แก่ไฟล์ `/usr/X11R6/lib/X11/rgb.txt` ซึ่งละเว้นส่วนขยายชื่อไฟล์ได้.

- ModulePath "*path*"
  ชื่อไดเรกทอรีที่เก็บไฟล์โมดูลสำหรับ X เซิร์ฟเวอร์. X เซิร์ฟเวอร์สามารถขยายความสามารถเพิ่มเติมโดยอาศัยการโหลดไฟล์โมดูลซึ่งแยกคอมไพล์ต่างหากกับ X เซิร์ฟเวอร์. การเลือกใช้โมดูลจะระบุในเซกชัน Module ภายหลัง.

- FontPath "*path*"
  ค่านี้สามารถเปลี่ยนโดยตัวเลือก `-fp` ของ X เซิร์ฟเวอร์เวลารัน. ในตัวอย่างระบุไดเรกทอรีที่เก็บไฟล์ฟอนต์ซึ่งเป็นการบอกให้เซิร์ฟเวอร์รู้ว่าฟอนต์อยู่ที่ไหน. ในกรณีที่ใช้ฟอนต์เซิร์ฟเวอร์, จะระบุฟอนต์เซิร์ฟเวอร์แทนชื่อไดเรกทอรี.

ฟอนต์เซิร์ฟเวอร์, ดูหน้า 287.

6.4.3 เซกชัน Module

ในเซกชันนี้จะระบุโมดูลที่ต้องการโหลดใช้กับ X เซิร์ฟเวอร์.

- Load "*module*"
  ระบุชื่อโมดูลที่ต้องการใช้เช่น
  - extmod เป็นโมดูลเพิ่มความสามารถพิเศษหลายอย่างให้กับเซิร์ฟเวอร์.
  - dri (Direct Rendering Infrastructure) เป็นโมดูลสำหรับเพิ่มความสามารถใช้เซิร์ฟเวอร์เข้าถึงฮาร์ดแวร์กราฟิกได้โดยตรง, ช่วยการแสดงผลด้านสามมิติ (3D).
  - dbe (Double Buffer Extension) โมดูลเพิ่มความสามารถช่วยในการแสดงผลภาพเคลื่อนไหวไม่ให้กระพริบไม่ลื่น.
  - record, xtrap โมดูลเพิ่มความสามารถการบันทึกจับโปรโตคอลของ X วินโดว์และอิเวนต์ต่าง ๆ.
  - glx (OpenGL Extension) โมดูลเพิ่มความสามารถ OpenGL.
  - speedo, type1, freetype โมดูลเพิ่มความสามารถเกี่ยวกับการใช้ฟอนต์แบบเวกเตอร์กับ X เซิร์ฟเวอร์.

OpenGL ▶
ย่อมาจาก Open Graphics Library เป็นไลบรารี (ซอฟต์แวร์) อินเทอร์เฟสสำหรับฮาร์ดแวร์กราฟิก, ใช้ในการสร้างภาพสามมิติคุณภาพสูง และเป็นมาตรฐานสากล.

หลังจากที่ X เซิร์ฟเวอร์ทำงานแล้วสามารถดูข้อมูลรายละเอียดเกี่ยวกับโมดูลต่างได้ด้วยคำสั่ง `xdpyinfo` กับตัวเลือก `-queryExtensions` หรือ `-ext all`.

6.4.4 เซกชัน InputDevices

เซกชัน InputDevices เป็นช่วงสำหรับตั้งค่าต่างสำหรับแป้นพิมพ์และเมาส์.

- Identifier "*name*"
  ชื่อเฉพาะใช้แยกแยะเซกชัน.

- Driver "kbd | mouse"
  ระบุประเภทของดีไวซ์ข้อมูลนำเข้าว่าเป็นแป้นพิมพ์หรือพอยเตอร์ (เมาส์).

- Option "*name*" "*value*"
  ตัวเลือกและค่าต่าง ๆของดีไวซ์. ตัวเลือกของดีไวซ์ยังแบ่งเป็นตัวเลือกสำหรับแป้นพิมพ์ใช้ระบุประเภทและคุณลักษณะของแป้นพิมพ์, และตัวเลือกสำหรับเมาส์ใช้ระบุจำนวนปุ่ม, ชื่อไฟล์ดีไวซ์ ฯลฯ.
  - "XkbModel" "*model*"
    ระบุประเภทแป้นพิมพ์เช่น pc101 หมายถึงแป้นพิมพ์ประเภทคอมพิวเตอร์ส่วนบุคคลที่มีจำนวนคีย์ 101 คีย์.

— "XkbLayout" "*layout*"
ระบุผังของแป้นพิมพ์, ภาษาที่ใช้กับแป้นพิมพ์. ตัวอย่างเช่น "us,th" เป็นการระบุการใช้แป้นพิมพ์ภาษาอังกฤษ (อเมริกัน) ร่วมกับแป้นพิมพ์ภาษาไทยโดยมีแป้นพิมพ์ภาษาอังกฤษเป็นแป้นพิมพ์หลัก (กลุ่มที่หนึ่ง) และแป้นพิมพ์ภาษาไทยเป็นแป้นพิมพ์รอง (กลุ่มที่สอง). ในกรณีที่ต้องการใช้แป้นพิมพ์มากกว่าสองภาษาสามารถเพิ่มต่อไปได้เรื่อย ๆ โดยการเขียนชื่อภาษาของแป้นพิมพ์แล้วคั่นด้วยเครื่องหมายลูกน้ำ. ชื่อผังแป้นพิมพ์ของภาษาต่าง ๆ สามารถดูได้จากไฟล์ `/usr/lib/X11/xkb/rules/xorg.lst`.
สำหรับการตั้งผังแป้นพิมพ์ภาษาไทยสามารถเลือกได้ 3 แบบได้แก่

1. *ผังแป้นพิมพ์เกษมณี (Kedmanee)* (รูปที่ 6.16) เป็นแป้นพิมพ์ดั้งเดิมที่ใช้กันทั่วไปในเครื่องพิมพ์ดีดและคอมพิวเตอร์ส่วนบุคคล. ปรับแต่งใช้แป้นพิมพ์ประเภทนี้โดยระบุคำว่า "th".
2. *ผังแป้นพิมพ์ปัตตะโชติ (Pattachote)* (รูปที่ 6.17) เป็นแป้นพิมพ์ที่วิจัยและออกแบบโดยคุณสฤษดิ์ ปัตตะโชติ. แป้นพิมพ์ออกแบบให้การใช้งานของมือทั้งสองใช้งานพอ ๆ กัน, ตำแหน่งคีย์ที่ใช้งานหนักจะอยู่ตรงกับนิ้วที่มีแรงมาก, และการเคลื่อนไหวของมือจะน้อยกว่าแป้นพิมพ์เกษมณี [43]. ปรับแต่งใช้แป้นพิมพ์ประเภทนี้โดยระบุคำว่า "th_pat".
3. *ผังแป้นพิมพ์ มอก. 820 (TIS-820.2538)* (รูปที่ 6.18) เป็นแป้นพิมพ์มาตรฐานที่กำหนดโดยสำนักงานมาตรฐานผลิตภัณฑ์อุตสาหกรรม [44] โดยพัฒนาต่อจากแป้นพิมพ์เกษมณีแต่สามารถพิมพ์เครื่องหมายพิเศษได้แก่ ◌ (ฟองมัน), ◌ (โคมูตร), ◌ (ยามักการ) และ ฯ (อังคั่นคู่) ได้ด้วย. ตำแหน่งของเครื่องหมายอังคั่นคู่ไม่ได้กำหนดไว้ในมาตรฐาน มอก. ตายตัว. ในระบบ XKB การพิมพ์อังคั่นคู่จะใช้คีย์เลือกระดับที่สาม (third level chooser) ช่วยซึ่งระบุด้วยตัวเลือก XkbOptions. คีย์ที่ใช้พิมพ์อังคั่นคู่ถูกกำหนดไว้อยู่ในตำแหน่งของตัวอักษรภาษาอังกฤษ "o". ปรับแต่งใช้แป้นพิมพ์ประเภทนี้โดยระบุคำว่า "th_tis".

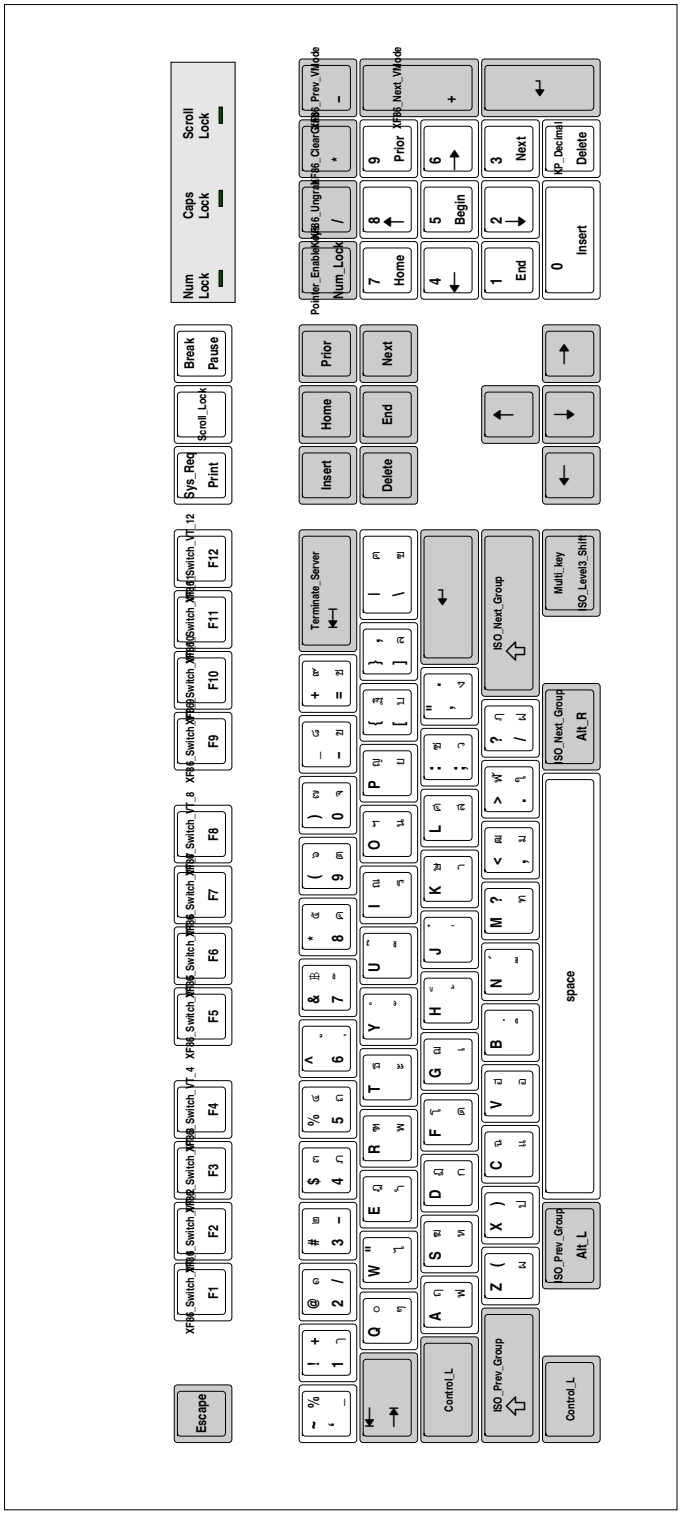
บ้างก็สะกดว่า Ketmanee ตามนามสกุลผู้คิดออกแบบผังแป้นพิมพ์คุณสุวรรณประเสริฐ เกษมณี [43].

บ้างก็สะกดว่า Pattajoti ตามนามสกุลผู้คิดออกแบบผังแป้นพิมพ์.

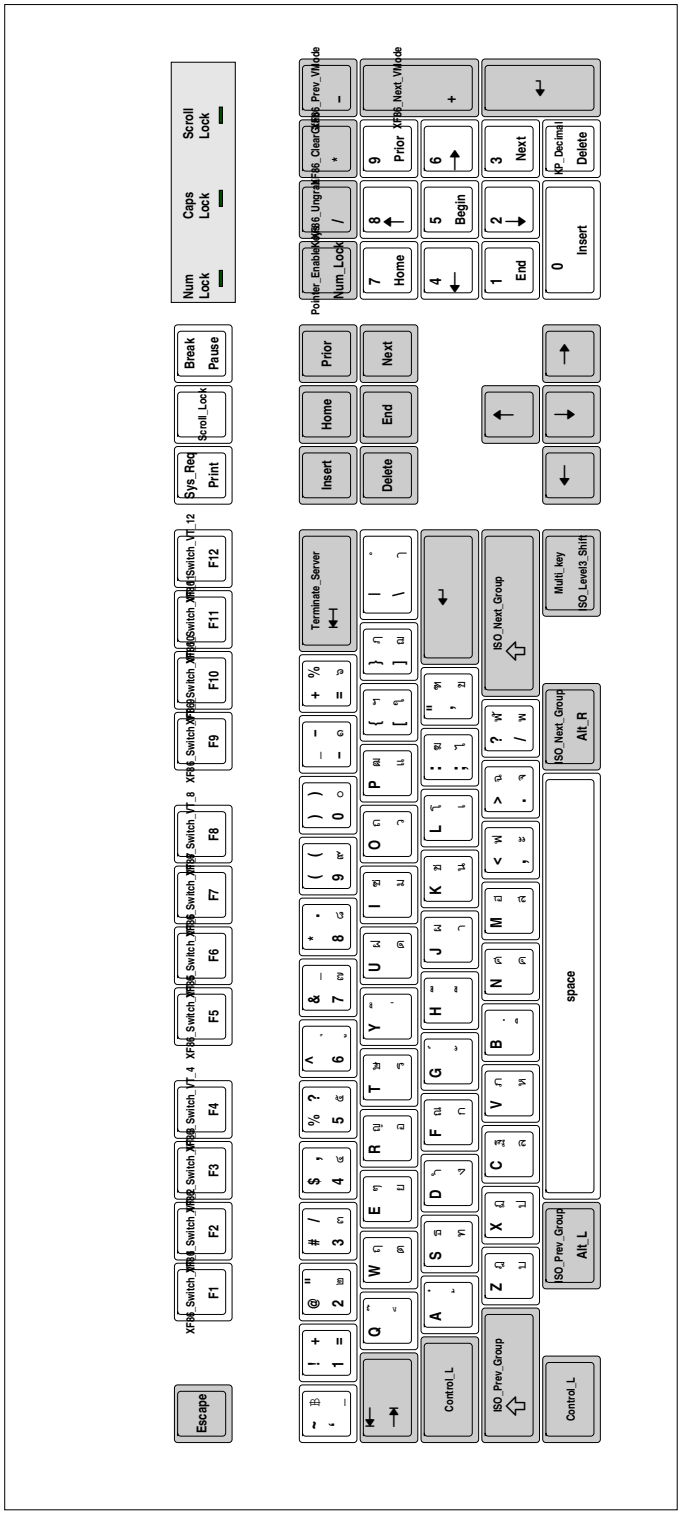
— "XkbOptions" "*option,option,...*"
ค่าตัวเลือกของ XKB มีหลายตัวขึ้นอยู่กับประเภทการปรับแต่งได้แก่.

* grp:*name*
สำหรับระบุคีย์ที่ใช้เปลี่ยนกลุ่มของแป้นพิมพ์ (ภาษาที่ใช้กับแป้นพิมพ์) เช่น grp:alt_shift_toggle เป็นการระบุสลับแป้นพิมพ์ด้วยการกดคีย์ `Alt+Shift`.

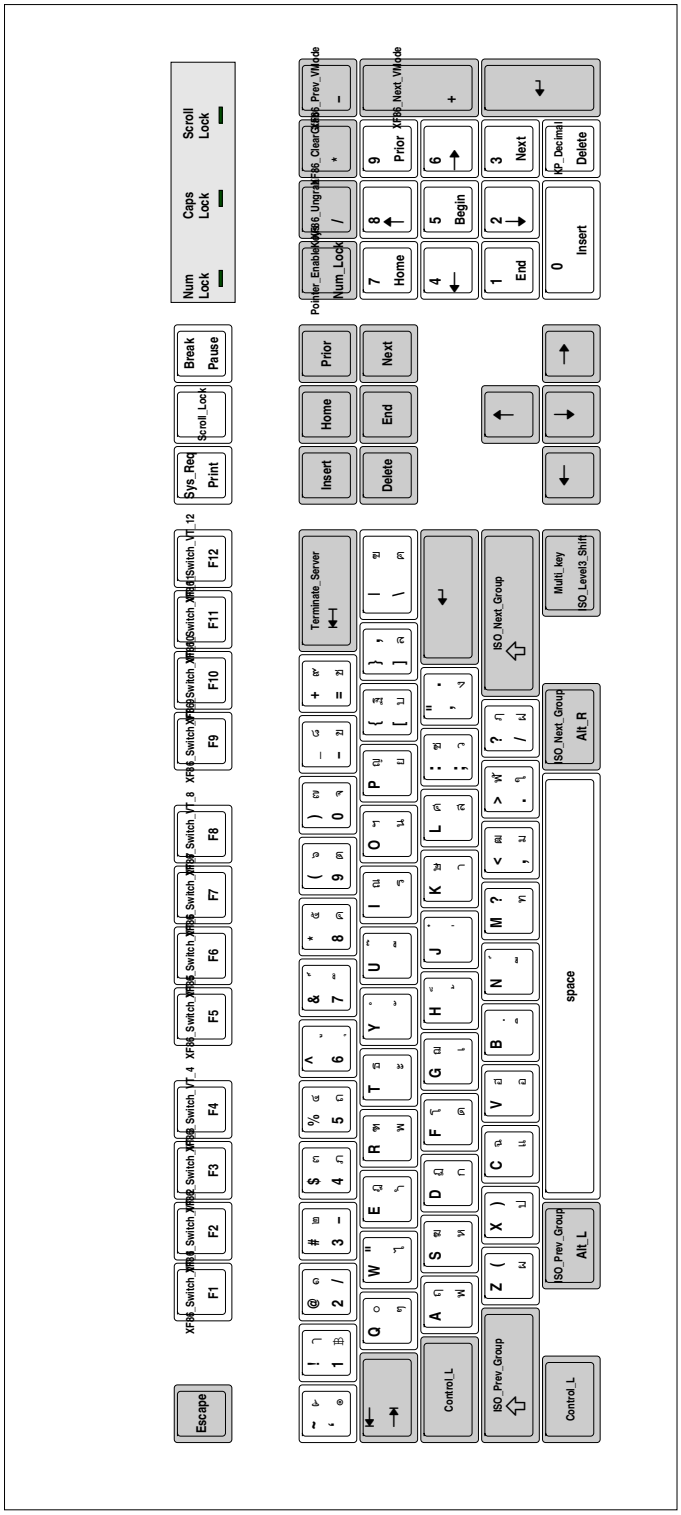
* lv3:*name*
สำหรับกำหนดคีย์ที่ใช้เลือกระดับที่ 3 (third level chooser). โดยปรกติแป้นพิมพ์ในแต่ละกลุ่มสามารถป้อนข้อมูลอักขระได้สองระดับคือการกด `Shift` และไม่กด `Shift`. ในกรณีที่อักขระมีมากและไม่สามารถได้หมดสามารถใช้คีย์เลือกระดับที่สามช่วยได้. ตัวอย่างเช่นแป้นพิมพ์



รูปที่ 6.16: แป้นพิมพ์เกษมณี (Kedmanee).



รูปที่ 6.17: แป้นพิมพ์ปัตตโชติ (Pattachote).



รูปที่ 6.18: แป้นพิมพ์สมอ 820 (tis-820.2538).

ภาษาไทยแบบ มอก. 820 จะใช้คีย์เลือกระดับที่สามในการพิมพ์เครื่องหมายอังคั่นคู่ (ฯ) โดยการกดคีย์เลือกระดับที่สามค้างไว้และกดคีย์ “o”. เช่น lv3:ralt_switch จะใช้คีย์ `Alt` ที่อยู่ทางขวามือเป็นคีย์เลือกระดับที่สาม.

* ctrl:*name*
  สำหรับปรับพฤติกรรมของคีย์ `Ctrl` เช่นใช้สลับเปลี่ยนตำแหน่งคีย์ `Caps Lock` และ `Ctrl` (ctrl:swapcaps), ตั้งค่าไม่ใช้คีย์ `Caps Lock` (ctrl:nocaps) เป็นต้น.

* grp_led:*name*
  ระบุใช้ไฟ LED (Light Emitting Diode) เป็นตัวบอกว่ามีสลับเปลี่ยนกลุ่มแป้นพิมพ์. เช่นค่า grp_led:scroll จะใช้ไฟ LED ของ Scroll lock เป็นตัวบอกสภาพกลุ่มของแป้นพิมพ์.

— “Protocol” “*protocol*”
ระบุโปรโตคอลของเมาส์ที่ใช้. ค่าทั่วไปได้แก่ “Auto” ซึ่งให้เซิร์ฟเวอร์จัดการโดยอัตโนมัติ. แต่ถ้าไม่สามารถเลื่อนพอยเตอร์ได้ต้องระบุเจาะจงเช่น “PS/2” สำหรับเมาส์ธรรมดาที่ต่อกับพอร์ต PS/2, “IMPS/2” สำหรับเมาส์ *IntelliMouse* ที่ต่อกับพอร์ต PS/2 หรือ USB. เมาส์แบบ IntelliMouse คือเมาส์ที่มีล้อ (wheel) ตรงกลางเลื่อนขึ้นลงหรือกดได้.

— “Device” “*device*”
ระบุไฟล์ดีไวซ์ของเมาส์ที่ใช้. ไฟล์ดีไวซ์โดยทั่วไปได้แก่ไฟล์

* `/dev/psaux` สำหรับเมาส์ที่ต่อกับพอร์ต PS/2.
* `/dev/mouse` มักจะเป็นซิมโบลิกลิงก์ไปหาไฟล์ดีไวซ์ตัวจริง.
* `/dev/input/mice` สำหรับเมาส์ที่ต่อกับพอร์ต USB.
* `/dev/input/mouse0` สำหรับเมาส์ที่ต่อกับพอร์ต USB.
* `/dev/ttyS0` สำหรับเมาส์ที่ต่อกับพอร์ตซีเรียล (serial).

วิธีทดสอบว่าเมาส์ที่ใช้ต่อกับไฟล์ดีไวซ์ไหนอาจจะตรวจสอบได้โดยการใช้คำสั่ง `cat` อ่านไฟล์ดีไวซ์นั้น, แล้วเลื่อนเมาส์ไปมา. ถ้ามีปฏิกริยาโต้ตอบทางหน้าจอแสดงว่าเป็นไฟล์ดีไวซ์นั้น.

— “Buttons” “*number*”
ระบุจำนวนปุ่มของเมาส์ที่ใช้.

— “ZAxisMapping” “*value*”
สำหรับเมาส์ที่มีล้อหมุนเช่นกรณีเมาส์ที่มีสองปุ่มและล้อหมุนหนึ่งอันจะถือว่ามีปุ่มทั้งหมด 5 ปุ่ม. เวลาหมุนล้อขึ้นข้างบนจะเป็นปุ่มที่ 4 และหมุนล้อลงจะเป็นปุ่มที่ 5. ตัวเลือก XAxisMapping จะจับคู่การหมุนล้อให้เข้ากับการเลื่อนเนื้อหาของหน้าต่างขึ้นลง (scroll) ตามที่ต้องการ. ตัวอย่างเช่น ZAxisMappin “4 5” ถ้าหมุนล้อไปข้างหน้า (ปุ่มที่ 4) จะเลื่อนหน้า

ต่างขึ้น (scroll up), ถ้าหมุนล้อไปข้างหลัง (ปุ่มที่ 5) จะเลื่อนหน้าต่างลงเป็นต้น. เมาส์ที่มีจำนวนปุ่มมากกว่า 5 ปุ่มจะมีการปรับแต่งที่แตกต่างจากนี้, สามารถอ่านรายละเอียดได้จากเอกสารการรองรับการใช้งานเมาส์ของ X11R6.8.1 [45].

6.4.5 เซกชัน Monitor

เป็นเซกชันสำหรับปรับแต่งจอภาพแสดงผล. ถ้าโปรแกรม `xorgcfg` ตั้งค่าให้โดยอัตโนมัติและใช้งานก็ใช้ค่าที่ตั้งให้ได้เลย. แต่ถ้ามีความจำเป็นต้องปรับค่าเองควรจะหาคู่มือของจอภาพเพื่อรวบรวมข้อมูลต่าง ๆ.

- VendorName "*vendor*"
  ชื่อของบริษัทผู้ผลิตจอภาพ. ค่านี้เป็นตัวเลือกไม่ระบุก็ได้.

- ModelName "*model*"
  ชื่อรุ่นของจอภาพ. ค่านี้เป็นตัวเลือกไม่ระบุก็ได้เช่นกัน.

- HorizSync *horizsync-range*
  ค่าช่วงความถี่ของจอภาพตามแนวนอนในหน่วยกโลเฮิร์ต (kHz) โดยปริยาย. ค่านี้สามารถดูได้จากคู่มือของจอภาพหรือด้านหลังจอภาพอาจจะมีเขียนบอกไว้. ถ้าไม่ระบุคุณสมบัตินี้จะใช้ค่าเป็น 28-33 kHz โดยปริยาย.

- VertRefresh *vertrefresh-range*
  ค่าช่วงความถี่ของจอภาพตามแนวตั้งในหน่วยเฮิร์ต (Hz) โดยปริยาย. ถ้าไม่ระบุคุณสมบัตินี้จะใช้ค่าเป็น 43-72 Hz โดยปริยาย.

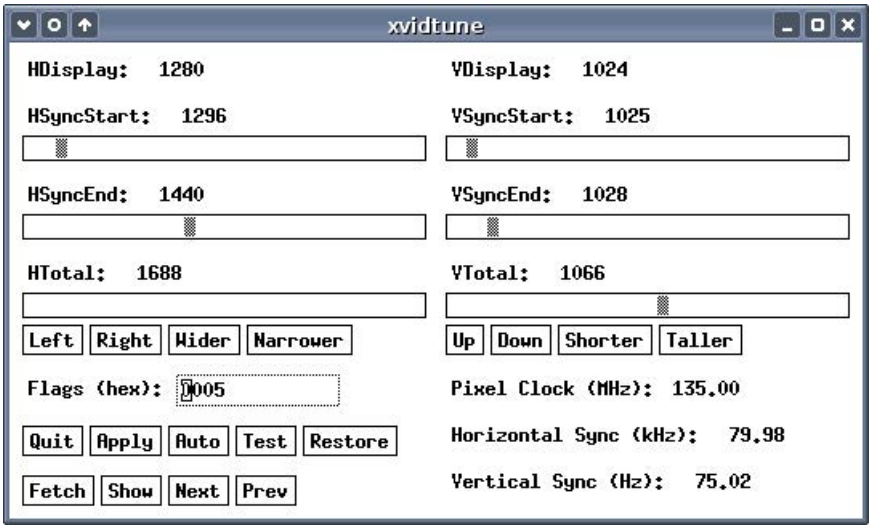
- DisplaySize *width height*
  ขนาดความกว้างและความสูงของของภาพในหน่วยมิลลิเมตร. X เซิร์ฟเวอร์สามารถใช้ค่านี้ในการคำนวณความละเอียด DPI (Dot Per Inch) ของหน้าจอได้.

ค่าคุณสมบัติอื่น ๆ ที่สำคัญ ๆ ได้แก่ Mode และ ModeLine ซึ่งใช้ปรับแต่งจอภาพโดยละเอียดยุ่งยากซับซ้อน. สำหรับจอภาพมาตรฐานโดยทั่วไปแล้วไม่จำเป็นต้องใช้ก็ได้. ถ้ามีความจำเป็นต้องใช้อาจจะใช้โปรแกรม `xvidtune` ช่วยปรับแต่งและสร้างค่า ModeLine ให้ได้. แต่มีข้อควรระวังคือโปรแกรมที่อาจจะทำให้วิดีโอการ์ดหรือจอภาพเสียหายได้.

6.4.6 เซกชัน Device

เซกชันนี้เป็นการกำหนดค่าคุณสัมบัติต่าง ๆของวิดีโอการ์ด. ค่าคุณสัมบัติที่จำเป็นต้องมีในเซกชันนี้ได้แก่ Identifier และ Driver.

- Driver "*name*"



รูปที่ 6.19: โปรแกรมช่วยปรับแต่งจอภาพ X เซิร์ฟเวอร์.

ระบุชื่อไดร์เวอร์ของวิดีโอการ์ดเช่น nv (nVidia), ati, i740 (Intel), i810 (Intel) เป็นต้น. โดยปรกติโปรแกรมช่วยสร้างไฟล์ `xconf.org` จะเขียนค่าเหล่านี้ให้. สำหรับผู้ใช้ต้องรู้ว่ากราฟิกการ์ดที่ตัวเองใช้มียี่ห้ออะไรรุ่นอะไรเป็นอย่างน้อย.

ค่าคุณสมบัติอื่น ๆอ่านได้จาก`man xorg.conf` และตัวเลือกเฉพาะของแต่ละไดร์เวอร์อ่านได้จากเอกสารของ X11R6 ทางเว็บไซต์.

6.4.7 เซกชัน Screen

เซกชันนี้เป็นส่วนสำคัญรวมนิยามส่วนประกอบต่าง ๆของ X เซิร์ฟเวอร์เข้าด้วยกันได้แก่

- Device "*device-id*"
  ชื่อดีไวซ์ที่ตั้งไว้ในเซกชัน Device.

- Monitor "*monitor-id*"
  ชื่อจอภาพที่ตั้งไว้ในเซกชัน Monitor.

- DefaultDepth *depth*
  ระบุความลึกของสีโดยปริยายในกรณีที่มีปรับแต่งให้ใช้ความลึกของสีได้หลายค่า. ความลึกของสีคือจำนวนบิตที่ใช้แสดงสีได้แก่ 8, 16, 24 หรือ 32 บิต.

ในเซกชันนี้สามารถแบ่งออกเป็นซับเซกชันชื่อ Display ได้อีกใช้ระบุว่ารายละเอียดของหน้าจอว่ามีความลึกของสีเท่าไร, มีความละเอียดของหน้าจอเป็นอย่างไร.

- Depth *depth*
  ความลึกของสีในซับเซกชัน.

- Modes “res1” “res2” ...
  ความละเอียดของหน้าจอในหน่วยพิกเซลเช่น 1280x1024, 1024x768 เป็นต้น. การเปลี่ยนความละเอียดหน้าจอหลังขณะใช้ X วินโดว์ทำได้โดยกดคีย์ `Ctrl+Alt` และเครื่องหมาย + หรือ −.

6.4.8 เซกชัน ServerFlags

เซกชันนี้เป็นเซกชันไม่บังคับ, ใช้ปรับแต่งคุณสมบัติโดยรวมของเซิร์ฟเวอร์. คุณสมบัติที่อยู่ในเซกชันนี้จะเป็นตัวเลือกทั้งหมด. ตัวเลือกที่น่าสนใจได้แก่.

- Option “DontVTSwitch” “on | off”
  ปรกติเราสามารถเปลี่ยนเทอร์มินอลเสมือนได้ด้วยการกดคีย์ `Ctrl+Alt+Fn` โดยที่ `Fn` คือคีย์ฟังก์ชัน F1, F2, ฯลฯ. ตัวเลือก DontVTSwitch สามารถบังคับให้ผู้ใช้ไม่สามารถเปลี่ยนเทอร์มินอลเสมือนโดยการตั้งค่าเป็น “on”.

- Option “DontZap” “on | off”
  ถ้าตั้งค่าเป็น “on” จะทำให้ผู้ใช้ไม่สามารถกดคีย์ `Ctrl+Alt+Backspace` เพื่อจบการทำงานของ X เซิร์ฟเวอร์.

ตัวเลือกอื่น ๆในช่วง ServerFlags สามารถอ่านได้จาก `man xorg.conf`.

6.5 X เซิร์ฟเวอร์

โปรแกรม X เซิร์ฟเวอร์ที่ใช้กันทั่วไปคือโปรแกรมที่มีชื่อว่า X ซึ่งจะแสดงผลกราฟิกทางหน้าจอและใช้เทอร์มินอลเสมือนที่อยู่ในตำแหน่งฟังก์ชันคีย์ F7 (รูปประกอบหน้า 29). ในระบบสามารถรัน X เซิร์ฟเวอร์ได้มากกว่าหนึ่งโปรเซส. โดยปรกติ X เซิร์ฟเวอร์ตัวแรกจะมีชื่อหน้าจอเป็น `:0`. X เซิร์ฟเวอร์ตัวถัดไปสามารถใช้ชื่อหน้าจออะไรก็ได้ที่มีชื่อไม่ซ้ำกันและผู้ใช้สามารถสลับเปลี่ยนหน้าจอโดยใช้คีย์ `Ctrl+Alt+Fn` โดยที่ `Fn` คือฟังชันคีย์ตั้งแต่ 7 เป็นต้นไป.

ในระบบที่มี X วินโดว์ทำงานอยู่แล้วและต้องการรัน X เซิร์ฟเวอร์เพิ่มสามารถสั่งคำสั่ง `X` *`display-name`* โดยระบุชื่อหน้าจอที่ไม่ซ้ำกับหน้าจอที่มีอยู่.

ตัวอย่างที่ 6.14: รัน X เซิร์ฟเวอร์เพิ่มจากที่มีอยู่.

```
$ X :1↵
```

จากตัวอย่างเป็นการระบุชื่อหน้าจอ `:1` ให้เป็นหน้าจอสำหรับ X เซิร์ฟเวอร์ตัวใหม่. หน้าจอของ X เซิร์ฟเวอร์ตัวใหม่จะมีแค่รูทวินโดว์เท่านั้น, ไม่มีประโยชน์สำหรับการใช้งานจริง. ในกรณีที่ต้องการใช้งานจริงให้ใช้คำสั่ง `startx` แทน. ในกรณีนี้ให้แน่ใจว่ามีไฟล์ตั้งค่าเริ่มต้นเช่น `~/.xinitrc` ถูกต้อง.

เกี่ยวกับไฟล์ `~/.xinitrc` ดูที่หน้า 278.

ตัวอย่างที่ 6.15: รัน X เซิร์ฟเวอร์เพิ่มจากที่มีอยู่โดยใช้คำสั่ง `startx`.

```
$ startx -- :1↵
```

6.6 การเริ่มต้น X เซิร์ฟเวอร์

การเริ่มทำงานของ X เซิร์ฟเวอร์แบ่งออกกว้าง ๆได้สองแบบคือสั่งทำงานจากเชลล์และสั่งเริ่มทำงานโดยใช้*ดิสเพลยแมนเนเจอร์ (display manager)*.

6.6.1 เริ่มต้น X เซิร์ฟเวอร์จากบรรทัดคำสั่ง

การเริ่มต้น X เซิร์ฟเวอร์จากบรรทัดคำสั่งหมายถึงการสั่งคำสั่งที่รัน X เซิร์ฟเวอร์จากเชลล์ไม่ว่าจะอยู่ในเท็กซ์โหมดหรือกราฟิกโหมด. โดยปรกติมักจะกระทำเวลาอยู่ในเท็กซ์โหมดเมื่อต้องการใช้ระบบ X วินโดว์. เช่นหลังจากที่สร้างไฟล์เริ่มต้นของ X เซิร์ฟเวอร์แล้วต้องการทดสอบเป็นต้น. นอกจากนั้น, เรายังสามารถรัน X เซิร์ฟเวอร์ในขณะที่มี X เซิร์ฟเวอร์ทำงานอยู่แล้ว (กราฟิก) ก็ได้. ในกรณีที่มี X เซิร์ฟเวอร์ทำงานอยู่แล้วต้องระบุให้เซิร์ฟเวอร์ใช้หน้าจอที่ไม่ซ้ำกับเซิร์ฟเวอร์ที่ทำงานอยู่.

startx อ้างอิงหน้า 427

คำสั่งทั่วไปที่ใช้เริ่มต้นการทำงานในระบบ X วินโดว์ได้แก่ `startx`. คำสั่ง `startx` เป็นเชลล์สคริปต์ที่จะตั้งค่าตัวแปรและเตรียมการเรียกสั่งคำสั่ง `xinit` ซึ่งเป็นโปรแกรมเริ่มต้น X เซิร์ฟเวอร์ต่อไป. โดยปรกติจะสั่งคำสั่ง `startx` โดยไม่มีตัวเลือกเพื่อเปลี่ยนการทำงานจากเท็กซ์โหมดเข้าสู่ระบบ X วินโดว์.

คำสั่ง `startx` จะเรียกใช้คำสั่ง `xinit` ซึ่งเป็นโปรแกรมเริ่มต้น X เซิร์ฟเวอร์. และคำสั่ง `xinit` มีไฟล์ตั้งค่าเริ่มต้นได้แก่ไฟล์ `/etc/X11/xinit/xinitrc` ← เนื้อหาของไฟล์ในแต่ละดิสโทรอาจแตกต่างกัน. และจะอ่านไฟล์นี้เหมือนเชลล์สคริปต์ทั่วไป. ถ้าในโฮมไดเรกทอรีของผู้สั่งคำสั่ง `startx` หรือ `xinit` มีไฟล์ `~/.xinitrc` ก็จะรันคำสั่งที่อยู่ในไฟล์นั้น.

คำสั่ง `startx` เป็นอินเทอร์เฟสสำหรับคำสั่ง `xinit` ช่วยสร้าง*เซสชัน (session)* และอำนวยความสะดวกต่าง ๆสำหรับรันโปรแกรม `xinit`. ดังนั้นถ้าต้องการเริ่มต้นระบบ X วินโดว์จากบรรทัดคำสั่ง, ให้ใช้คำสั่ง `startx` ดีกว่าการใช้คำสั่ง `xinit` โดยตรง. ตัวอย่างต่อไปนี้ตัวอย่างไฟล์ `xinitrc` ซึ่งเป็นเชลล์สคริปต์รันโปรแกรมต่าง ๆที่ต้องการหลังจากเริ่มระบบ X วินโดว์. ไฟล์นี้จะถูกอ่านไม่ว่าจะเริ่มต้นระบบ X วินโดว์ด้วย `startx` หรือ `xinit` ก็ตาม.

ตัวอย่างที่ 6.16: ไฟล์ `xinitrc`

```
$ cat -n /etc/X11/xinit/xinitrc↵
     1  #!/bin/sh
     2  # $Xorg: xinitrc.cpp,v 1.3 2000/08/17 19:54:30 cpqbld Exp $
     3
     4  userresources=$HOME/.Xresources
     5  usermodmap=$HOME/.Xmodmap
     6  xinitdir=/usr/X11R6/lib/X11/xinit
     7  sysresources=$xinitdir/.Xresources
     8  sysmodmap=$xinitdir/.Xmodmap
     9
    10  # merge in defaults and keymaps
    11
    12  if [ -f $sysresources ]; then
    13      xrdb -merge $sysresources                ← ตั้งค่าทรัพยากรจากฐานข้อมูล
```

```
14  fi
15
16  if [ -f $sysmodmap ]; then
17      xmodmap $sysmodmap                          ← ปรับแต่งคีย์ของแป้นพิมพ์
18  fi
19
20  if [ -f $userresources ]; then
21      xrdb -merge $userresources                  ← ตั้งค่าทรัพยากรจากฐานข้อมูล
22  fi
23
24  if [ -f $usermodmap ]; then
25      xmodmap $usermodmap                         ← ปรับแต่งคีย์ของแป้นพิมพ์
26  fi
27
28  # First try ~/.xinitrc
29  if [ -f "$HOME/.xinitrc" ]; then                ← ถ้ามีไฟล์ ~/.xinitrc
30          XINITRC="$HOME/.xinitrc"
31          exec /bin/sh "$HOME/.xinitrc"           ← เปลี่ยนโปรเซสที่กระทำการ
32  # If not present, try the system default
33  elif [ -n "`/etc/X11/chooser.sh`" ]; then
34          exec "`/etc/X11/chooser.sh`"
35  # Failsafe
36  else
37          # start some nice programs
38          twm &
39          xclock -geometry 50x50-1+1 &
40          xterm -geometry 80x50+494+51 &
41          xterm -geometry 80x20+494-0 &
42          exec xterm -geometry 80x66+0+0 -name login
43  fi
```

ไฟล์ xinitrc ของระบบตามตัวอย่างที่ 6.16 ในช่วงแรกเป็นการใช้คำสั่ง xrdb เพื่อตั้งค่าทรัพยากรต่าง ๆ, สั่งคำสั่ง xmodmap เพื่อปรับแต่งคีย์ต่าง ๆของแป้นพิมพ์. ช่วงสุดท้ายจะตรวจสอบดูว่าในโฮมไดเรกทอรีมีไฟล์ .xinitrc หรือไม่. ถ้ามีก็จะใช้คำสั่ง exec ซึ่งเป็นคำสั่งประกอบภายในเชลล์เปลี่ยนโปรเซสเชลล์ที่ทำงานอยู่ไปเป็นโปรเซสของโปรแกรมที่ต้องการรัน. กล่าวคือหลังจากที่รันคำสั่งนั้นแล้วจะไม่กลับมารันโปรแกรมบรรทัดถัดไปอีก. ตัวอย่างเช่นบรรทัดที่ 31, เมื่อรันคำสั่ง exec /bin/sh "$HOME/.xinitrc" แล้วโปรเซสที่ทำงานอยู่จะเปลี่ยนเป็นโปรเซสของ /bin/sh ที่สั่งไป, และไม่กลับมารันคำสั่งที่อยู่ในบรรทัดถัดจากบรรทัดที่ 31 อีกต่อไป.

ถ้าไม่มีไฟล์ .xinitrc ในโฮมไดเรกทอรีก็จะตรวจสอบว่ามีไฟล์ /etc/X11/chooser.sh หรือไม่. ถ้ามีไฟล์นี้ก็จะรันโปรแกรมต่าง ๆในไฟล์นี้. สุดท้ายถ้าไม่มีไฟล์อะไรเลยก็จะรัน twm, xclock และ xterm เป็นโปรแกรมเริ่มต้น.

ขั้นตอนเหล่านี้อาจจะไม่เหมือนกันทุกประการ, แล้วแต่ดิสโทรที่ใช้.

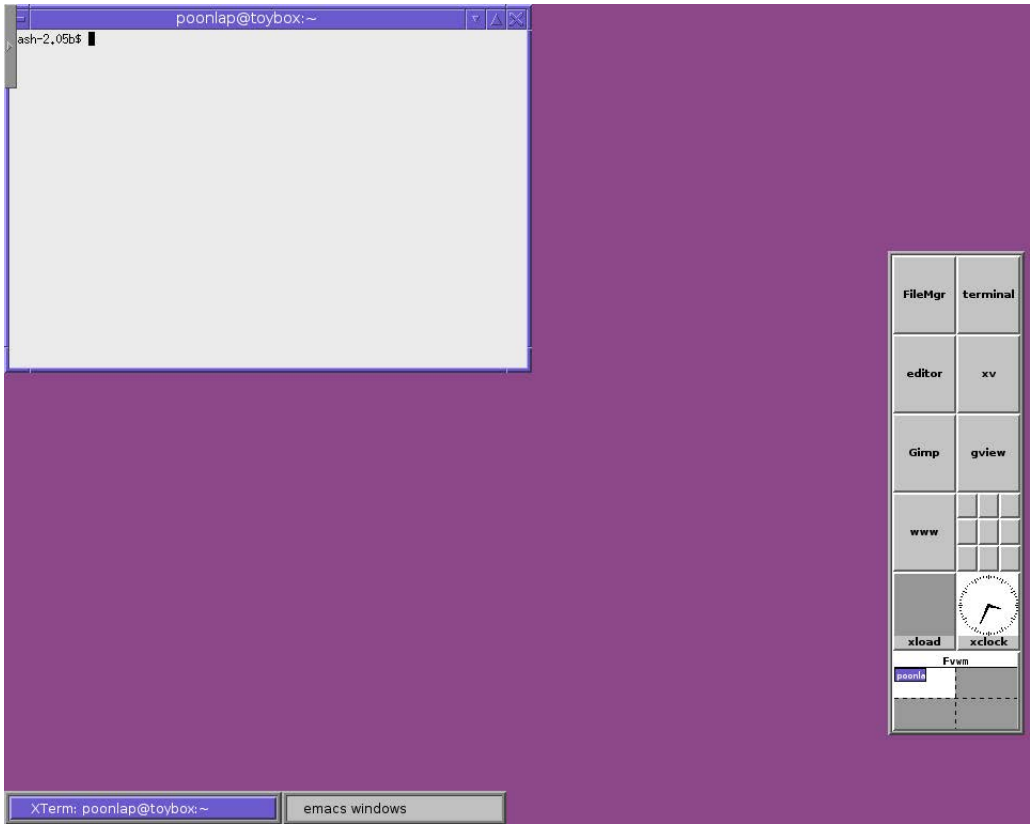
ให้สังเกตุว่าการรันโปรแกรมต่าง ๆจะรันแบบแบ็กกราวน์และโปรแกรมที่รันตัวสุดท้าย (ในตัวอย่างเช่น xterm -geometry 80x66+0+0 -name login) จะรันแบบฟอร์กราวน์และใช้คำสั่งประกอบภายในเชลล์ exec รัน. ถ้าโปรแกรมที่รันตัวสุดท้ายจบการทำงาน, X เซิร์ฟเวอร์ก็จะจบการทำงานกลับสู่สภาพก่อนรันคำสั่ง startx หรือ xinit. ถ้าโปรแกรมตัวสุดท้ายทำงานแบบแบ็กกราวน์, X เซิร์ฟเวอร์จะจบการทำงานทันทีเหมือนไม่มีอะไรเกิดขึ้น.

ไฟล์ ~/.xinitrc

ไฟล์ ~/.xinitrc ก็รูปแบบคล้ายกับไฟล์ xinitrc คือเขียนโปรแกรมที่ต้องการใช้หลังจากเริ่มระบบ X วินโดว์. โดยปรกติโปรแกรมที่เขียนในไฟล์นี้ได้แก่วินโดว์แมนเนเจอร์, โปรแกรมที่ใช้ในระบบ X วินโดว์เช่น xterm ฯลฯ. หรือโปรแกรมเริ่ม*เซสชัน (session)* ของสภาพแวดล้อมเดกสท็อปเช่น gnome-session, startkde, xfce4-session เป็นต้น.

ตัวอย่างที่ 6.17: ไฟล์ ~/.xinitrc ที่ไม่ได้ใช้สภาพแวดล้อมเดกสท็อป.

```
$ cat ~/.xinitrc↵
xterm &                                    ← โปรแกรมที่สั่งต้องอยู่ใน $PATH
exec fvwm2                                 ← วินโดว์แมนเนเจอร์ที่ต้องการใช้
```



รูปที่ 6.20: ภาพหน้าจอหลังจากรันคำสั่ง startx ประกอบกับไฟล์ ~/.xinitrc ตัวอย่างที่ 6.17.

ถ้าต้องการใช้สภาพแวดล้อมเดกสท็อป, ให้เขียนโปรแกรมเริ่มเซสชันสภาพแวดล้อมที่ต้องการในไฟล์ ~/.xinitrc. ในกรณีนี้โปรแกรมเริ่มต้นจะมีการจัดการวินโดว์แมนเนเจอร์, โปรแกรมที่ต้องการรันเริ่มต้นอยู่แล้ว. ผู้ใช้สามารถปรับแต่งคุณสมบัติเหล่านี้ได้จากเมนูของสภาพแวดแล้วเดกสท็อปนั้น ๆ.

ตัวอย่างที่ 6.18: ไฟล์ ~/.xinitrc ที่ใช้สภาพแวดล้อมเดกสท็อป.

```
$ cat ~/.xinitrc↵
# exec startkde                          ← บรรทัดที่ขึ้นต้นด้วย # เป็นคอมเมนต์
# exec xfce4-session
exec gnome-session                       ← โปรแกรมเริ่มเซสชันสำหรับ Gnome
```

รูปที่ 6.21: ภาพหน้าจอหลังจากรันคำสั่ง `startx` ประกอบกับไฟล์ `~/.xinitrc` ตัวอย่างที่ 6.18.

การสั่งคำสั่ง `startx` หรือ `xinit` มักจะสั่งคำสั่งโดยไม่มีตัวเลือกประกอบ. แต่ถ้าต้องการใช้ตัวเลือกประกอบด้วย, จะแบ่งเป็นเลือกสำหรับไคล์เอ็นต์และตัวเลือกสำหรับ X เซิร์ฟเวอร์โดยมีรูปแบบดังนี้.

```
startx [ [client] options ... ] [ -- [server] options ... ]
```

client คือไฟล์โปรแกรมที่ต้องการรันเป็นไคลเอ็นต์หลังจากเข้าสู่ระบบ X วินโดว์. *server* คือไฟล์โปรแกรม X เซิร์ฟเวอร์.

ในสภาพที่มี X เซิร์ฟเวอร์ทำงานอยู่แล้วและต้องการรันโปรแกรม `firefox` อย่างเดียวในหน้าจอที่ 2 ซึ่งเป็นคนละหน้าจอกับที่ใช้อยู่, สามารถทำได้ตามตัวอย่างต่อไปนี้.

ตัวอย่างที่ 6.19: คำสั่ง `startx` และตัวเลือกที่เกี่ยวกับไคลเอ็นต์, เซิร์ฟเวอร์.

```
$ startx /usr/bin/firefox http://linux.thai.net -- :1 -depth 16↵
```

ในกรณีนี้ไคลเอ็นต์ที่ทำงานเมื่อเข้าสู่ระบบ X วินโดว์ได้แก่ `firefox` และมีอาร์กิวเมนต์คือ URL ของเว็บไซต์ที่ต้องการเปิดดู. หลังเครื่องหมาย `--` จะเป็นส่วนที่เกี่ยวข้องกับ X เซิร์ฟเวอร์และ `:1` ได้แก่ชื่อหน้าจอที่สอง (`:1`), ตัวเลือก `-depth` สำหรับ X เซิร์ฟเวอร์, และไม่มีการระบุโปรแกรม X เซิร์ฟเวอร์ที่ต้องการใช้ซึ่งจะหมายถึง `/usr/X11R6/bin/X` โดยปริยาย. หน้าจอของ X วินโดว์จะใช้อยู่ที่เทอร์มินอลเสมือนตัวที่ยังไม่ได้ใช้ (ดูรูป 2.4 หน้า 29 ประกอบ).

หน้าจอที่หนึ่งคือ `:0`

6.6.2 เริ่มต้น X เซิร์ฟเวอร์ด้วยดิสเพลยแมเนเจอร์

การใช้*ดิสเพลยแมนเนเจอร์ (display manager)* เป็นอีกวิธีหนึ่งสำหรับการเริ่มต้นระบบ X วินโดว์. ดิสเพลยแมนเนเจอร์ที่นิยมใช้กันอยู่ในปัจจุบันได้แก่ `xdm`, `gdm` และ `kdm`. `xdm` เป็นดิสเพลยแมนเนเจอร์มาตรฐานที่มาพร้อมกับ X เซิร์ฟเวอร์, ส่วน `gdm` และ `kdm` เป็นโปรแกรมดิสเพลยแมนเนเจอร์ของสภาพแวดล้อม GNOME และ KDE ตามลำดับ.

วิธีการเริ่มต้นของดิสเพลยแมนเนเจอร์และการเลือกประเภทของดิสเพลยแมนเนเจอร์จะแตกต่างกันไปตามดิสโทรที่ใช้.

การเริ่มต้นดิสเพลยแมนเนเจอร์โดยปรกติจะเป็นเชลล์สคริปต์ที่รันตอนบูตเครื่องขึ้นอยู่กับดิสโทรแต่ละค่าย. ดิสโทรตระกูล Red Hat จะมีเชลล์สคริปต์ `/etc/X11/prefdm` รันอยู่ในโปรแกรม `init` ซึ่งเขียนอยู่ในไฟล์ `/etc/inittab`. ประเภทของดิสเพลยแมนเนเจอร์สามารถระบุด้วยตัวแปร `DISPLAYMANAGER="`*`type`*`"` ในไฟล์ `/etc/sysconfig/desk` เช่น `DISPLAYMANAGER="GDM"`.

เรื่องเกี่ยวกับ `init` ให้ดูหน้าที่ ??.

ดิสโทรตระกูล Debian จะระบุดิสเพลยแมนเนเจอร์ที่ใช้ในไฟล์ `/etc/X11/default-display-ma`

ดิสโทรตระกูล Gentoo จะใช้เชลล์สคริปต์ `/etc/init.d/xdm` เป็นตัวเริ่มต้นดิสเพลยแมนเนเจอร์และเขียนประเภทของดิสเพลยแมนเนเจอร์ที่ต้องการใช้ในไฟล์ `/etc/rc.conf` เช่น `DISPLAYMANGER="gdm"` เป็นต้น.

อย่างไรก็ตามเชลล์สคริปต์ต่าง ๆเหล่านี้จะรันโปรแกรม `xdm`, `gdm` หรือ `kdm` ซึ่งดิสเพลยแมนเนเจอร์ของแต่ละแบบ, และผู้ใช้สามารถรันคำสั่งเหล่านี้ได้จากบรรทัดคำสั่งด้วย.

XDMCP โปรโตคอล

หน้าที่หลักของดิสเพลยแมนเนเจอร์คือเป็นอินเทอร์เฟสสำหรับล็อกอินเข้าสู่ระบบแบบกราฟิก. การล็อกอินโดยใช้ดิสเพลยแมนเนเจอร์สามารถล็อกอินได้จากเครื่องตัวเองหรือผ่านทางเน็ตเวิร์กก็ได้. การล็อกแบบกราฟิดผ่านทางเน็ตเวิร์กจะอาศัยโปรโตคอลที่เรียกว่า *XDMCP (X Display Manager Control Protocol)*. สมมติว่ามีเครื่องคอมพิวเตอร์ A และ B และคอมพิวเตอร์ A รัน X เซิร์ฟเวอร์และอนุญาตให้เครื่องคอมพิวเตอร์อื่นติดต่อกับ X เซิร์ฟเวอร์ผ่านทาง XDMCP ได้. จากคอมพิวเตอร์ B สามารถรัน X เซิร์ฟเวอร์และเรียกหน้าจอล็อกอินของคอมพิวเตอร์ A มาแสดงที่หน้าจอคอมพิวเตอร์ได้.

การติดต่อขอหน้าจอล็อกอินด้วยโปรโตคอล XDMCP จะระบุด้วยตัวเลือกตอนที่รัน X เซิร์ฟเวอร์ได้แก่

1. ติดต่อโฮสโดยตรง (query)
   ในกรณีที่รู้ชื่อโฮสหรือ IP แอดเดรสของเครื่องที่อนุญาตให้ล็อกอินผ่านโปรโตคอล XDMCP สามารถใช้ตัวเลือก `-qnuery` *`host`* เช่นตัวอย่างต่อไปนี้เป็นการรัน X เซิร์ฟเวอร์โดยใช้หน้าจอ `:1` และเรียกหน้าจอล็อกอินจากเครื่องคอมพิวเตอร์ที่มี IP แอดเดรส 192.168.11.2.

   ตัวอย่างที่ 6.20: ขอหน้าจอล็อกอินโดยตรงโดยใช้ XDMCP โปรโตคอล.

   ```
   $ X -query 192.168.11.2 :1↵
   ```

2. บรอดแคสต์ (broadcast)
   ใช้วิธี*บรอดแคสต์ (broaadcast)* ประกาศหาโฮสที่อนุญาตให้ติดต่อหน้าจอล็อกอินในเครือข่ายท้องถิ่นที่ใช้อยู่. X เซิร์ฟเวอร์จะติดต่อกับโฮสเครื่องแรกที่หาเจอด้วยการบรอดแคสต์และแสดงหน้าจอล็อกอินของโฮสนั้น.

   ตัวอย่างที่ 6.21: ขอหน้าจอล็อกอินด้วยวิธีบรอดแคสต์โดยใช้ XDMCP โปรโตคอล.

   ```
   $ X -brodcast :1↵
   ```

3. ติดต่อโฮสโดยทางอ้อม (indirect)
   วิธีนี้คล้ายกับการบรอดแคสต์แต่จะระบุตัวเลือก `-indirect` *`host`* ให้แสดงหน้าจอรายชื่อโฮสที่อนุญาตเปิดรับ XDMCP โปรโตคอล. รายชื่อโฮสนี้ได้มาจากการสอบถามไปที่โฮส *`host`* ที่ระบุเป็นอาร์กิวเมนต์ของตัวเลือก.

   ตัวอย่างที่ 6.22: ขอหน้าจอล็อกอินด้วยวิธีอ้อมโดยใช้ XDMCP โปรโตคอล.

   ```
   $ X -indirect localhost :1↵
   ```

การปรับแต่งดิสเพลยแมนเนเจอร์

การปรับแต่งดิสเพลยแมนเจอร์สามารถใช้โปรแกรมปรับแต่งเฉพาะของแต่ละระบบได้เช่น `gdmconfig` หรือ `kdm_config`. โปรแกรมเหล่านี้มักจะมีให้เลือกจากเมนูในสภาพแวดล้อมเดกส์ทอปอยู่แล้ว. ส่วนการปรับแต่ง `xdm` ทำได้โดยการปรับแต่งไฟล์ต่าง ๆที่เกี่ยวข้องใต้ไดเรกทอรี `/etc/X11/xdm` ซึ่งสามารถอ่านได้จาก `man xdm`.

6.7 X เซิร์ฟเวอร์แบบพิเศษ

นอกจากโปรแกรม X เซิร์ฟเวอร์ที่แสดงผลกราฟิกทางหน้าจอแล้วในระบบ X วินโดว์ยังมีโปรแกรม `Xnest`, `Xvfb` และ `Xvnc` ซึ่งเป็นโปรแกรม X เซิร์ฟเวอร์แต่เป็น X เซิร์ฟเวอร์แบบพิเศษใช้เฉพาะงานต่างกันไป.

รูปที่ 6.22: การขอหน้าจอล็อกอินโดยใช้วิธี indirect.

6.7.1 Xnest

โปรแกรม Xnest เป็น X เซิร์ฟเวอร์ประเภทหนึ่งที่แสดงหน้าจอเป็นหน้าต่างแอพพลิเคชันในหน้าจอ X วินโดว์ที่ทำงานอยู่. ตัวอย่างการใช้งานเช่นใช้ Xnest ขอหน้าจอล็อกอินจากเครื่องคอมพิวเตอร์เครื่องอื่นผ่านเน็ตเวิร์กเป็นต้น.

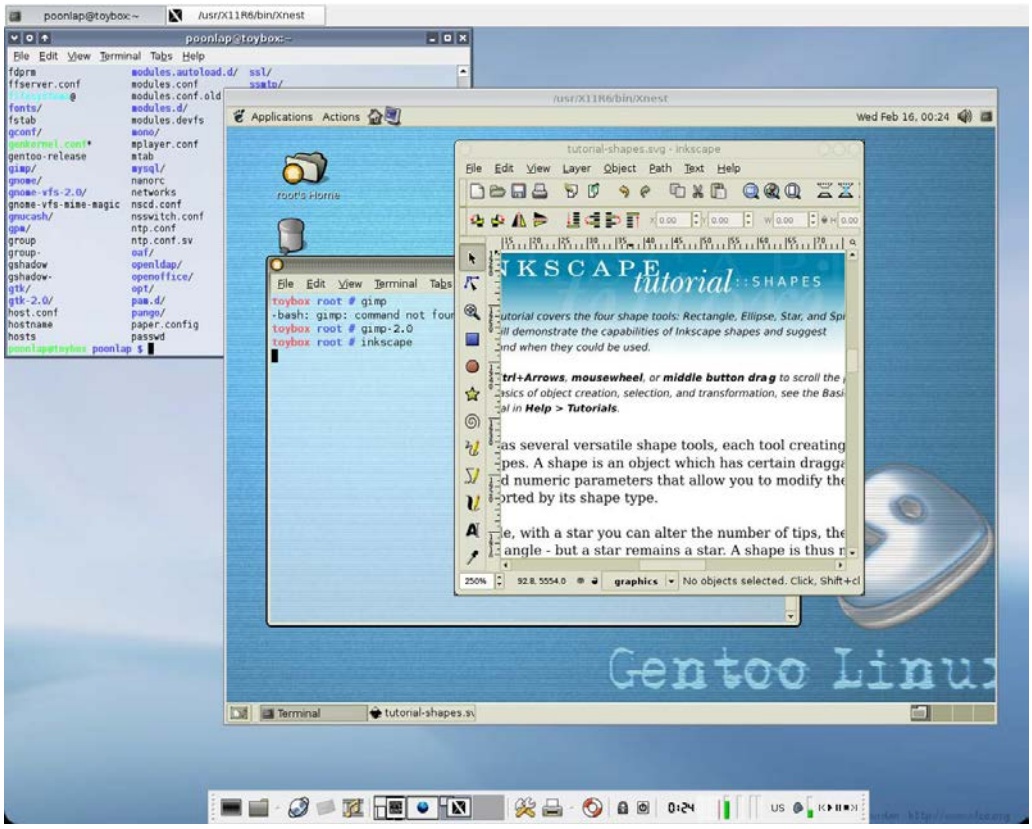
ตัวอย่างที่ 6.23: ตัวอย่างการใช้ Xnest.

```
$ Xnest -indirect localhost :1 &↵                          ← ใช้เดี่ยวๆ
$ startx /usr/X11R6/bin/xlogo -- /usr/X11R6/bin/Xnest :2 &↵   ← ใช้กับ startx
```

ในสภาพแวดล้อมเดกสท็อป GNOME มีโปรแกรมคล้ายกับ Xnest สำหรับจัดการแสดงผลโปรแกรม Xnest ให้โดยอัตโนมัติที่ผู้ใช้ไม่ต้องรันโปรแกรม Xnest โดยตรงได้ แก่โปรแกรม gdmXnest, gdmXnestchooser และ gdmflexiserver.

ตัวอย่างที่ 6.24: โปรแกรมเกี่ยวกับ Xnest ในสภาพแวดล้อมเดกสท็อป GNOME.

```
$ gdmXnest &↵                                  ← แสดงหน้าจอ Xnest
$ gdmXnestchooser &↵                ← แสดง Xnest กับตัวเลือก -indirect
$ gdmflexiserver -n &↵               ← แสดงหน้าจอล็อกอิน gdm ใน Xnest
```



รูปที่ 6.23: หน้าต่างโปรแกรม Xnest ในหน้าจอ X วินโดว์.

โปรแกรม gdmXnest และ gdmXnestchooser จะแสดงชื่อหน้าจอที่กำหนดให้อัตโนมัติทางเทอร์มินอลที่สั่งคำสั่ง. ผู้ใช้สามารถรันโปรแกรมอะไรก็ได้ให้ไปแสดงผลใน Xnest โดยระบุชื่อหน้าจอนั้น.

6.7.2 Xvfb

Xvfb ย่อมาจาก X server Virtual Frame Buffer เป็น X เซิร์ฟเวอร์เสมือน, คือไม่มีการแสดงผลให้เห็นทางหน้าจอแต่จะเป็นการจำลองการทำงานของ X เซิร์ฟเวอร์. ในระบบที่ไม่มีอุปกรณ์แสดงผลเช่นไม่มีหน้าจอและไม่มีอุปกรณ์ป้อนข้อมูลเข้า, แต่ในบางกรณีต้องการรันคำสั่งที่ใช้ X เซิร์ฟเวอร์ประมวลผลอะไรสักอย่าง, สามารถใช้โปรแกรม Xvfb ช่วยได้.

สมมติว่าเราล็อกอินเข้าสู่ระบบผ่านทางเน็ตเวิร์กเข้าไปในเครื่องคอมพิวเตอร์เครื่องหนึ่งผ่านทางเทอร์มินอล. และในระบบมีโปรแกรม testprogram ซึ่งจะแสดงผลในระบบ X วินโดว์แต่มีข้อจำกัดทางเทคนิคทำให้ติดต่อกับคอมพิวเตอร์เครื่องนั้นได้ทางเทอร์มินอลเท่านั้น. ในกรณีนี้ถ้าต้องการดูผล (รูป) ของโปรแกรม, สามารถรัน Xvfb สร้าง X เซิร์ฟเวอร์เสมือนก่อนแล้วให้โปรแกรมที่ต้องการรันแสดงผลทางหน้าจอที่ระบุ. จากนั้นใช้โปรแกรม xwd แปลงการแสดงผลของ X เซิร์ฟเวอร์เป็นไฟล์รูปภาพได้.

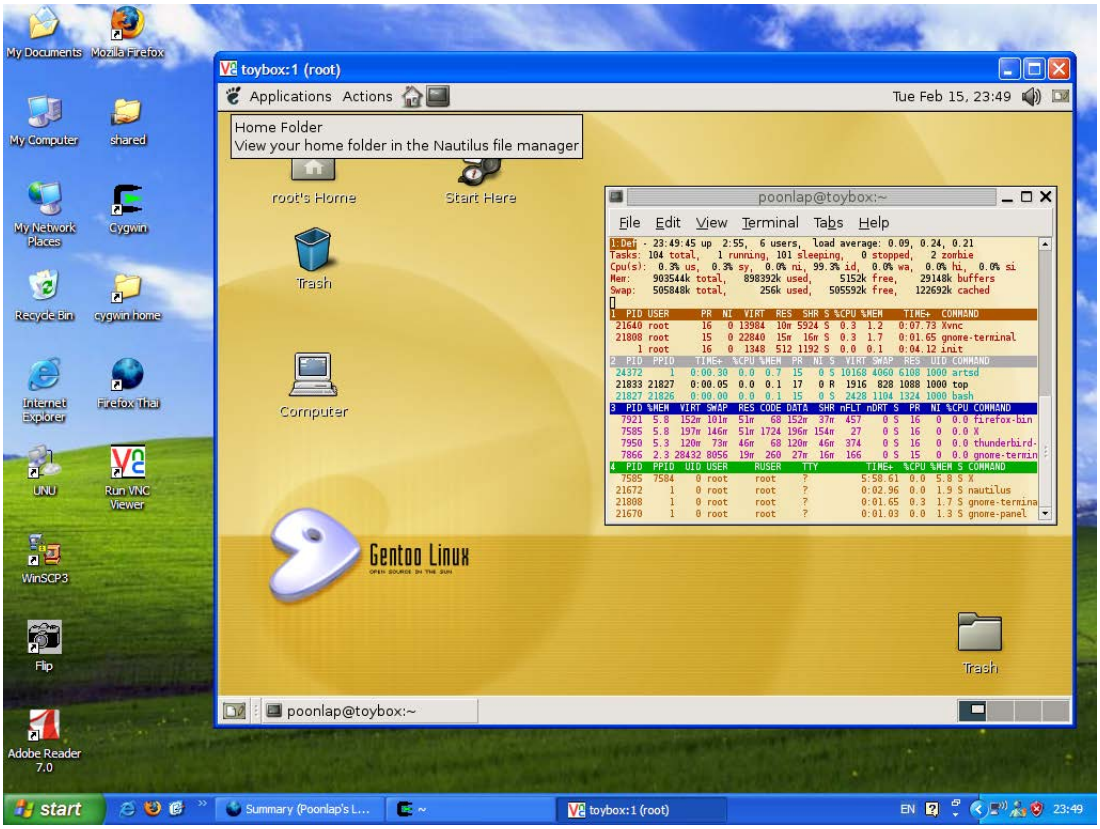
xwd อ้างอิงหน้า 429

ตัวอย่างที่ 6.25: การใช้ Xvfb ในระบบที่ไม่มีจอภาพ.

```
$ Xvfb :1 -ac &↵                                    ← ไม่มีการแสดงผลทางหน้าจอ
$ DISPLAY=:1 testprogram &↵                         ← แสดงผลไปที่ Xvfb
$ xwd -root -display :1 -out testprogram.xwd↵       ← จับภาพหน้าจอ
$ convert testprogram.xwd testprogram.png↵          ← แปลงฟอร์แมตรูป
```

6.7.3 Xvnc

Xvnc เป็น X เซิร์ฟเวอร์ที่มีคุณสมบัติสามารถผลผ่านทางเน็ตเวิร์กโดยใช้โพรโทคอล *VNC (Vertual Network Computing)* แทน X โพรโตคอล. โปรแกรม X เซิร์ฟเวอร์นี้ไม่ใช่โปรแกรมในระบบ X วินโดว์มาตรฐานแต่เป็นโปรแกรมในแพ็กเกจของ VNC [46]. ข้อดีของการใช้โพรโตคอล VNC คือมีไคลเอ็นต์สำหรับควบคุมและดูหน้าจอในหลายแพลตฟอร์มคือสามารถดูหน้าจอได้จากระบบปฏิบัติการใด ๆก็ได้ที่มีโปรแกรม VNC viewer.



รูปที่ 6.24: หน้าต่างโปรแกรม VNC viewer บนระบบปฏิบัติการวินโดวส์.

เวลาใช้งานจะไม่รันคำสั่ง Xvnc โดยตรงแต่จะใช้คำสั่ง vncserver แทน. โดยระบุชื่อหน้าจอเป็นอาร์กิวเมนต์.

ตัวอย่างที่ 6.26: ใช้งาน Xvnc.

```
$ vncserver -geometry 800x600 :1↵
```

```
You will require a password to access your desktops.

Password:                                                    ← ใส่รหัสผ่าน
Verify:

New 'toybox:1 (poonlap)' desktop is toybox:1                 ← toybox คือชื่อโฮส

Creating default startup script /home/poonlap/.vnc/xstartup
Starting applications specified in /home/poonlap/.vnc/xstartup
Log file is /home/poonlap/.vnc/toybox:1.log
```

ถ้าเป็นการสั่งคำสั่ง `vncserver` เป็นครั้งแรก, ตัวโปรแกรมจะสร้างไดเรกทอรี `~/.vnc` ให้ผู้ใช้ตั้งรหัสผ่านสำหรับเวลาที่ติดต่อแสดงผล. ในไดเรกทอรี `~/.vnc` จะมีไฟล์ `xstartup` เป็นไฟล์ที่มีหน้าที่เหมือนกับ `~/.xinitrc` รันโปรแกรมต่าง ๆหลังจาก X เซิร์ฟเวอร์เริ่มทำงาน.

การติดต่อดูหน้าจอจะใช้โปรแกรม VNC viewer ใช้ได้ในหลายแพล็ตฟอร์ม, ในระบบปฏิบัติการลินุกซ์ได้แก่โปรแกรม `vncviewer`. วิธีใช้เบื้องต้นให้ใส่ชื่อหน้าจอที่ต้องการติดต่อดูผ่าน VNC โพรโทคอลเป็นอาร์กิวเมนต์ของโปรแกรม. ในกรณีที่ใช้ `vncviewer` ผ่านทางเน็ตเวิร์ก, ให้ใช้ IP แอดเดรสหรือชื่อโฮสตามด้วยชื่อหน้าจอ. เช่น `192.168.1.5:1` เป็นต้น.

ตัวอย่างที่ 6.27: ใช้ `vncviewer` ติดต่อดูหน้าจอ.

```
$ vncviewer :1↵                          ← หรือจะใช้ชื่อหน้าจอเป็น toybox:1 ก็ได้
```

นอกจากจะใช้ `vncviewer` ติดต่อกับ `Xvnc` แล้วยังสามารถใช้ติดต่อกับหน้าจอของระบบปฏิบัติการใด ๆก็ที่รัน VNC เซิร์ฟเวอร์อยู่ เช่นใช้ดูหน้าจอไมโครซอฟต์วินโดวส์ที่รัน VNC เซิร์ฟเวอร์จากลินุกซ์เป็นต้น. ตัวหน้าต่างโปรแกรม `vncviewer` จะมีคีย์พิเศษ F8 เมื่อกดแล้วจะแสดงเมนูเพื่อกระทำการต่าง ๆเฉพาะกรณีเช่นส่งคีย์ `Ctrl+Alt+Del` ให้กับเครื่องที่ติดต่ออยู่ปลายทาง, ปรับหน้าจอให้แสดงผลเต็มจอภาพ เป็นต้น.

วิธีการจบการทำงานของ `vncserver` ทำได้โดยใช้ตัวเลือก `-kill` และตามด้วยชื่อหน้าจอ.

ตัวอย่างที่ 6.28: จบการทำงานของ `vncserver`.

```
$ vncserver -kill :1↵
Killing Xvnc process ID 31339
```

โปรแกรม `vncserver` จะสร้าง X เซิร์ฟเวอร์ตัวใหม่เพิ่มจาก X เซิร์ฟเวอร์ที่มีอยู่โดยใช้โปรแกรม `Xvnc` ถ้าต้องการแสดงหรือแชร์ (share) หน้าจอที่ทำงานอยู่แล้ว (หน้าจอ `:0`) ให้ใช้โปรแกรม `x0vncserver`. โปรแกรมนี้จะแสดงผลหน้าจอ `:0` หรือหน้าจอที่ระบุด้วยตัวเลือก `-display=`*`dpy`*. ตัวเลือกอื่น ๆสามารถดูได้จากการใช้ตัวเลือก `-h` ประกอบ.

6.8 ระบบจัดการฟอนต์

ระบบจัดการฟอนต์เพื่อแสดงอักขระต่าง ๆในระบบ X วินโดว์แบ่งออกเป็นสองวิธีได้แก่ระบบจัดการฟอนต์ดั้งเดิมที่เรียกกันว่า *X คอร์ฟอนต์ (X core fonts)* และ Xft. ระบบจัดการฟอนต์ทั้งสองแตกต่างกันที่เซิร์ฟเวอร์จะเป็นตัวจัดการการแสดงผลฟอนต์ในระบบดั้งเดิม, ไคลเอ็นต์จะเป็นตัวจัดการแสดงผลฟอนต์ในระบบ Xft โดยอาศัยไลบรารี. เมื่อเปรียบเทียบเรื่องเกี่ยวกับการเพิ่มฟอนต์หรือปรับแต่งฟอนต์ในระบบ, การดูแลระบบจัดการฟอนต์แบบ Xft จะง่ายกว่าระบบดั้งเดิมมาก. ในช่วงนี้เป็นการแนะนำระบบจัดการฟอนต์ทั้งสองแบบซึ่งมีวิธีการจัดการแตกต่าง. การรู้จักระบบจัดการฟอนต์ทั้งสองแบบช่วยให้ผู้ใช้ทำความเข้าการติดตั้งฟอนต์ใหม่ ๆเช่นฟอนต์ภาษาไทยได้ดีขึ้น.

6.8.1 ระบบจัดการฟอนต์แบบดั้งเดิม

ระบบการจัดการฟอนต์แบบดั้งเดิมกระทำโดยตัว X เซิร์ฟเวอร์เองและใช้จัดการแสดงผลฟอนต์แบบบิตแมปเป็นหลัก. ต่อมามีการเพิ่มความสามารถแสดงผลฟอนต์แบบเวกเตอร์เช่นฟอนต์ PostScript หรือทรูไทป์ในรูปของโมดูล. โมดูลที่ช่วยให้ X เซิร์ฟเวอร์สามารถใช้ฟอนต์ทรูไทป์ในช่วงแรกได้แก่ xfstt และ xfsft [47]. ผู้ใช้ต้องเลือกใช้ระหว่างโมดูลทั้งสองซึ่งไม่มีความเข้ากัน. ต่อมาความนิยมของ xfsft ซึ่งใช้ไลบรารี FreeType [48] มีมากขึ้นและถูกรวมไว้ในโครงการ XFree86 ในที่สุด. ในปัจจบันถ้าต้องการใช้ฟอนต์ทรูไทป์ที่จัดการด้วย X เซิร์ฟเวอร์ให้โหลดโมดูล “freetype” ในเซกชัน “Module”.

ไดเรกทอรีที่เก็บฟอนต์

การติดตั้งและปรับแต่งฟอนต์จะระบุไดเรกทอรีหรือฟอนต์เซิร์ฟเวอร์ที่เก็บฟอนต์ต่าง ๆในไฟล์ตั้งค่าเริ่มต้นของเซิร์ฟเวอร์เช่นไฟล์ `xorg.conf`. ชื่อไดเรกทอรีที่เก็บฟอนต์จะเป็นค่าของพารามิเตอร์ FontPath ในเซกชัน “Files” (หน้า 266). ในไดเรกทอรีหรือฟอนต์เซิร์ฟเวอร์ที่เก็บฟอนต์เหล่านั้นจะมีไฟล์ต่าง ๆที่เกี่ยวข้องกับการติดตั้งฟอนต์ในระบบได้แก่

- ไฟล์ `fonts.dir`
  เป็นไฟล์ที่สร้างด้วยคำสั่ง `mkfontdir` บันทึกชื่อไฟล์ฟอนต์และชื่อฟอนต์ที่สัมพันธ์กัน. บรรทัดแรกจะเป็นจำนวนของฟอนต์ทั้งหมดที่บันทึกในไฟล์. คำสั่ง `mkfontdir` จะตรวจสอบไฟล์ฟอนต์บิตแมป `.bdf`, `.pcf`, ไฟล์ฟอนต์ที่บีบอัดด้วย `gzip` และไฟล์ `fonts.scale` ที่อยู่ในไดเรกทอรีที่ทำงานอยู่และสร้างไฟล์ `fonts.dir` ในไดเรกทอรีที่รันคำสั่ง.

ตัวอย่างที่ 6.29: ไฟล์ `fonts.dir`

```
$ head /usr/share/fonts/misc/fonts.dir↵
516
10x20-ISO8859-1.pcf.gz -misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-1
10x20-ISO8859-10.pcf.gz -misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-10
10x20-ISO8859-11.pcf.gz -misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-11
10x20-ISO8859-13.pcf.gz -misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-13
```

```
10x20-ISO8859-14.pcf.gz -misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-14
10x20-ISO8859-15.pcf.gz -misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-15
10x20-ISO8859-16.pcf.gz -misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-16
```

- ไฟล์ `fonts.scale` เป็นไฟล์ที่สร้างด้วยคำสั่ง `mkfontscale` หรือ `ttmkfdir`. คำสั่งเหล่านี้จะตรวจสอบไฟล์ฟอนต์เวกเตอร์เช่นทรูไทป์หรือ PostScript ที่อยู่ในไดเรกทอรีที่ทำงานอยู่แล้วสร้างไฟล์ `fonts.scale`. ไฟล์ที่สร้างจะเป็นข้อมูลนำเข้าสำหรับคำสั่ง `mkfontdir` เพื่อสร้างไฟล์ `fonts.dir` อีกต่อไป.

ตัวอย่างที่ 6.30: ไฟล์ `fonts.scale`

```
$ head /usr/share/fonts/TTF/fonts.scale↵
luximb.ttf -b&h-Luxi Mono-bold-r-normal--0-0-0-0-m-0-adobe-standard
luximb.ttf -b&h-Luxi Mono-bold-r-normal--0-0-0-0-m-0-ascii-0
luximb.ttf -b&h-Luxi Mono-bold-r-normal--0-0-0-0-m-0-iso10646-1
luximb.ttf -b&h-Luxi Mono-bold-r-normal--0-0-0-0-m-0-iso8859-1
luximb.ttf -b&h-Luxi Mono-bold-r-normal--0-0-0-0-m-0-iso8859-10
luximb.ttf -b&h-Luxi Mono-bold-r-normal--0-0-0-0-m-0-iso8859-13
luximb.ttf -b&h-Luxi Mono-bold-r-normal--0-0-0-0-m-0-iso8859-15
luximb.ttf -b&h-Luxi Mono-bold-r-normal--0-0-0-0-m-0-iso8859-16
luximb.ttf -b&h-Luxi Mono-bold-r-normal--0-0-0-0-m-0-iso8859-2
```

- ไฟล์ `fonts.alias` ชื่อฟอนต์ในไฟล์ `fonts.dir` เป็นชื่อฟอนต์มาตรฐานที่เรียกว่า XLFD ซึ่งยาวและเข้าใจยาก. ไฟล์ `fonts.alias` จะเป็นไฟล์ที่ระบุชื่อสั้น ๆ (นามแฝง) ที่ใช้แทนชื่อ XLFD.

ตัวอย่างที่ 6.31: ไฟล์ `fonts.alias`

```
$ head /usr/share/fonts/misc/fonts.alias↵
! $Xorg: fonts.alias,v 1.3 2000/08/21 16:42:31 coskrey Exp $
fixed        -misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
variable     -*-helvetica-bold-r-normal-*-*-120-*-*-*-*-iso8859-1
5x7          -misc-fixed-medium-r-normal--7-70-75-75-c-50-iso8859-1
5x8          -misc-fixed-medium-r-normal--8-80-75-75-c-50-iso8859-1
6x9          -misc-fixed-medium-r-normal--9-90-75-75-c-60-iso8859-1
6x10         -misc-fixed-medium-r-normal--10-100-75-75-c-60-iso8859-1
6x12         -misc-fixed-medium-r-semicondensed--12-110-75-75-c-60-iso8859-1
6x13         -misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
6x13bold     -misc-fixed-bold-r-semicondensed--13-120-75-75-c-60-iso8859-1
```

ฟอนต์เซิร์ฟเวอร์

การระบุไดเรกทอรีที่เก็บฟอนต์ด้วยพารามิเตอร์ FontPath ในไฟล์ `xorg.conf` เป็นวิธีการเบื้องต้นสำหรับบอกให้ X เซิร์ฟเวอร์รู้แหล่งที่เก็บฟอนต์. วิธีการนี้เข้าใจง่ายแต่มีข้อเสียบางอย่างได้แก่

- ฟอนต์ใหม่ที่เพิ่มเข้าไปในระบบไม่สามารถใช้งานได้ทันทีจนกว่าจะเริ่มต้น X เซิร์ฟเวอร์ใหม่อีกครั้ง.
- ฟอนต์ที่สามารถใช้ได้ขึ้นอยู่กับ X เซิร์ฟเวอร์ว่ามีฟอนต์อะไรใช้ได้บ้าง. ในกรณี

ที่ใช้โพรโตคอล XDMCP ล็อกอินข้ามโฮส, เครื่องที่ติดต่อสำหรับล็อกอินกับ X เซิร์ฟเวอร์ที่แสดงผลเป็นคนละเครื่องกัน. ดังนั้นเครื่องที่ติดต่อล็อกอินกับฟอนต์ที่ X เซิร์ฟเวอร์สามารถแสดงผลได้อาจจะไม่เหมือนกันหรือฟอนต์บางตัว X เซิร์ฟเวอร์ที่ใช้แสดงผลอาจจะไม่มี.

ปัญหาเหล่านี้สามารถแก้ได้โดยการใช้*ฟอนต์เซิร์ฟเวอร์ (X font server)*. ในกรณีที่ใช้ฟอนต์เซิร์ฟเวอร์จะสามารถใช้ฟอนต์ใหม่ที่เพิ่มเข้าสู่ระบบได้ทันทีโดยไม่ต้องเริ่ม X เซิร์ฟเวอร์ใหม่เพียงแต่เริ่มต้นการทำงานของฟอนต์เซิร์ฟเวอร์เท่านั้น. และสามารถใช้ฟอนต์ผ่านทางเน็ตเวิร์กได้ด้วย.

โปรแกรมสำหรับรันฟอนต์เซิร์ฟเวอร์ได้แก่ `xfs` ซึ่งจะมีไฟล์ตั้งค่าเริ่มต้นโดยปริยายได้แก่ `/usr/X11R6/lib/X11/fs/config`. โดยปรกติผู้ใช้จะไม่รันฟอนต์เซิร์ฟเวอร์ด้วยคำสั่ง `xfs` โดยตรงแต่จะใช้ init script เช่น `/etc/init.d/xfs` เป็นเชลล์สคริปต์สำหรับเริ่มต้นทำงานฟอนต์เซิร์ฟเวอร์และไฟล์ตั้งค่าเริ่มต้นจะเป็น `/etc/X11/fs/config`.

init script, ดูหน้า ??

ตัวอย่างที่ 6.32: ไฟล์ตั้งค่าเริ่มต้นฟอนต์เซิร์ฟเวอร์.

```
$ cat /etc/X11/fs/config↵
#
# X Font Server configuration file
#

# อนุญาตให้ไคลเอ็นต์ติดต่อกับเซิร์ฟเวอร์ได้ไม่เกิน 4 ตัว
#client-limit = 4

# ไม่ใช้ tcp, ใช้ socket เท่านั้น
no-listen  =  tcp

# ถ้ามีการติดต่อใช้เซิร์ฟเวอร์เกินกำหนดให้สร้างเซิร์ฟเวอร์ตัวใหม่
clone-self = on

# ฟอนต์เซิร์ฟเวอร์อื่นสำหรับให้ไคลเอ็นต์ใช้
#alternate-servers = foo:7101,bar:7102

# ระบุไดเรกทอรีที่เก็บฟอนต์
#
catalogue = /usr/share/fonts/75dpi,
        /usr/share/fonts/100dpi,
        /usr/share/fonts/misc,
        /usr/share/fonts/Type1,
        /usr/share/fonts/truetype,
        /usr/share/fonts/TTF

# ขนาดฟอนต์ปริยาย, 12 พอยต์
default-point-size = 120

# 100 x 100 และ 75 x 75
default-resolutions = 75,75,100,100

# บันทึกล็อก
use-syslog = on

# ควบคุม cache. หน่วยเป็น KB
```

```
cache-hi-mark = 2048
cache-low-mark = 1433
cache-balance = 70
```

ส่วนที่สำคัญในไฟล์ config ได้แก่ค่าพารามิเตอร์ catalogue ใช้สำหรับระบุไดเรกทอรีที่เก็บฟอนต์ต่าง ๆ. ในกรณีที่ระบุไดเรกทอรีมากกว่าหนึ่งที่ให้ใช้เครื่องหมายลูกน้ำ (,) คั่น.

ฟอนต์เซิร์ฟเวอร์จะเริ่มทำงานโดยอัตโนมัติเมื่อระบบเริ่มต้นทำงาน, แต่ถ้าต้องการรันหรือหยุดการทำงานของฟอนต์เซิร์ฟเวอร์ด้วยตนเองสามารถใช้สคริปต์ /etc/init.d/xfs และใช้อาร์กิวเมนต์ start, stop หรือ restart ซึ่งจะเหมือนกับการรันเซิร์ฟเวอร์โดยใช้ init script ทั่วไป.

ตัวอย่างที่ 6.33: การรันฟอนต์เซิร์ฟเวอร์โดยตรง.

```
# /etc/init.d/xfs start↵
 * Scanning font directories...                                    [ ok ]
 * Starting X Font Server...                                       [ ok ]
# /etc/init.d/xfs restart↵
 * Stopping X Font Server...                                       [ ok ]
 * Scanning font directories...                                    [ ok ]
 * Starting X Font Server...                                       [ ok ]
# /etc/init.d/xfs stop↵
 * Stopping X Font Server...                                       [ ok ]
```

ปรับแต่งให้ระบบใช้ฟอนต์เซิร์ฟเวอร์

การปรับแต่งให้ X เซิร์ฟเวอร์ใช้ฟอนต์เซิร์ฟเวอร์ทำได้โดยเขียนชื่อฟอนต์เซิร์ฟเวอร์ในพารามิเตอร์ FontPath ในไฟล์ xorg.conf โดยมีรูปแบบดังนี้.

```
trans/hostname:port-number
```

trans หมายถึง transport type คือวิธีติดต่อกับฟอนต์เซิร์ฟเวอร์ว่าจะติดต่อผ่านยูนิกซ์โดเมนซ็อกเก็ตหรือผ่านโพรโตคอล tcp. ถ้าเครื่องที่รัน X เซิร์ฟเวอร์มีฟอนต์เซิร์ฟเวอร์ทำงานอยู่ด้วย, สามารถติดต่อกับเครื่องตัวเองด้วยยูนิกซ์โดเมนซ็อกเก็ต. ในกรณีนี้จะเขียนเป็น

tcp ▶
ย่อมาจากคำว่า Transfer Control Protocol เป็นโปรโตคอลมาตรฐานสำหรับควบคุมข้อมูลที่ส่งผ่านทางอินเทอร์เน็ต. มีระบบสร้างคอนเน็กชันระหว่างโฮส, ตรวจเช็กข้อมูล ฯลฯ

```
FontPath        "unix/:-1"
```

ถ้าใช้การติดต่อเซิร์ฟเวอร์ผ่าน tcp ไปยังเครื่องอื่นจะเขียนเป็น

```
FontPath        "tcp/hostname-or-ip:7100"
```

hostname-or-ip เป็นชื่อโฮสหรือ IP แอดเดรส. ตัวเลขที่อยู่หลังเครื่องหมาย : คือหมายเลขพอร์ตของฟอนต์เซิร์ฟเวอร์ซึ่งปรกติจะมีค่าเป็น 7100.

ชื่อฟอนต์แบบ XLFD

ชื่อฟอนต์ที่ใช้ในระบบ X วินโดว์แบบดั้งเดิมมีชื่อเรียกว่า *XLFD (X Logical Font Description)* และคำสั่งที่ใช้แสดงชื่อฟอนต์ในระบบได้แก่ `xlsfonts`.

xlsfonts อ้างอิงหน้า 428

ตัวอย่างที่ 6.34: แสดงชื่อฟอนต์แบบ XLFD ในระบบ.

```
$ xlsfonts↵
-adobe-courier-bold-o-normal--0-0-100-100-m-0-iso10646-1
-adobe-courier-bold-o-normal--0-0-100-100-m-0-iso8859-1
-adobe-courier-bold-o-normal--0-0-100-100-m-0-iso8859-10
--- แสดงผลต่อไปเรื่อยๆ ---
10x20                                          ← ชื่อฟอนต์แบบไม่เป็นทางการ, นามแฝง
12x24
12x24kana
12x24romankana
5x7
--- แสดงผลต่อไปเรื่อยๆ ---
```

เนื่องจากชื่อฟอนต์แบบ XLFD ยาวมากและยากที่จำ, ชื่อฟอนต์บางตัวอาจจะมีนามแฝง (alias) ได้เช่น `10x20` ก็เป็นชื่อฟอนต์ที่ใช้ได้ในระบบเช่นกัน. ชื่อฟอนต์แบบ XLFD ประกอบด้วยส่วนต่าง ๆคั่นด้วยเครื่องหมาย - โดยมีรูปแบบดังต่อไปนี้.

```
-fndry-fmly-wght-slant-sW-adstyl-pxlsz-ptSz-resx-resy-spc-avgW-reg-enc
```

- Foundry (`fndry`)
  ผู้สร้างฟอนต์เช่น adobe, bitstream, mise ฯลฯ.
- Font family (`fmly`)
  ชื่อตระกูลฟอนต์เช่น times, angsana, fixed ฯลฯ.
- Weight (`wght`)
  แสดงความหนาของตัวอักขระในฟอนต์ได้แก่ bold, demibold, medium, regular และ small.
- Slant (`slant`)
  แสดงความเอียงของตัวอักขระในฟอนต์ได้แก่ i (italic), o (oblique) และ r (roman). รูปทรง italic และ oblique ดูคล้ายกันซึ่งจริง ๆแล้วมีความหมายไม่เหมือนกัน. รูปทรง italic หมายถึงรูปทรงวาดให้อักขระเอียงและมีความเป็นโค้ง (cursive) ปรับแต่งให้สวยงามด้วย. ส่วน oblique นั้นเป็นเพียงแค่รูปทรงที่จับให้เอียงเช่นการทำให้รูปทรงตรงเอียง. ทรง roman คืออักขระภาษาอังกฤษแต่ในที่นี้จะหมายถึงรูปทรงอักขระตั้งตรง.
- Set-width name (`sW`)
  ความกว้างของตัวอักษรเช่น normal, condensed, semicondensed, narrow, double wide เป็นต้น.

- Additional style (`adstyl`)
  ทรงเพิ่มเติมเช่น ja, ko, sans หรือเว้นว่างไว้.

- Pixel size (`pxlsz`)
  ความสูงในหน่วยพิกเซล.

- Point size (`ptSz`)
  ความสูงในหน่วย 10 เท่าของพอยต์ (point). เช่นค่า 100 หมายถึงฟอนต์ขนาด 10 พอยต์

1 พอยต์มีค่าเท่ากับ 1/72 นิ้ว

- Dots-per-inch (`resx` และ `resy`)
  ค่า dpi (dot per inch) ในแนวนอนและแนวตั้ง.

- pppSpacing (`spc`)
  ระยะตัวอักษรได้แก่ c (character cell), m (monospace) และ p (proportional). monospace หมายถึงอักขระทุกตัวในฟอนต์มีความกว้างเท่ากันทุกตัวซึ่งตรงข้ามกับประเภท propotional ซึ่งความกว้างของทุกอักขระไม่จำเป็นต้นเท่ากัน. สำหรับฟอนต์แบบ proportional, อักขระบางตัวอาจจะไม่มีความกว้างเช่นสระหรือวรรณยุกต์บางตัวไม่มีความกว้าง. ฟอนต์แบบ character cell เป็นฟอนต์แบบ monospace แบบหนึ่งแต่ออกแบบมาให้ใช้กับเทอร์มินอลเช่น `xterm` โดยเฉพาะ.

monospace มีชื่อเรียกอีกอย่างว่า fixed-width. ส่วน proportional มีชื่อเรียกอีกอย่างว่า variable-width.

- Average width (`avgW`)
  ความกว้างโดยเฉลี่ยของอักขระในหน่วย 10 เท่าของพิกเซล.

- Character registry (`reg`)
  ประเภทของอักขระแบ่งตามภาษา. เช่น iso8859 เป็นฟอนต์สำหรับภาษาที่ใช้ในยุโรป, iso10646 เป็นฟอนต์แบบยูนิโค้ดมีอักขระของหลายชาติรวมในฟอนต์เดียว, tis620 เป็นฟอนต์ภาษาไทย เป็นต้น.

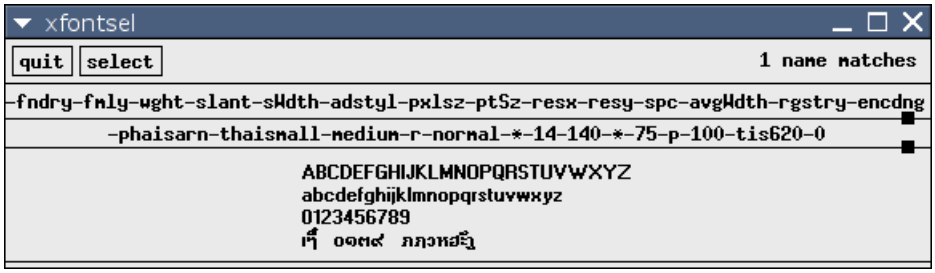
- Encoding (`enc`)
  รายละเอียดย่อยของประเภทอักขระแบ่งตามภาษา. มีค่าเป็นตัวเลขและใช้ร่วมกับค่า character registry. เช่น iso8859-1 หมายถึงฟอนต์ลาติน (ภาษาอังกฤษและอักขระลาติน), tis620-0 หมายถึงฟอนต์ภาษาไทยพื้นฐาน เป็นต้น.

การเลือกฟอนต์

ในระบบ X วินโดว์มีโปรแกรมมาตรฐานที่ใช้เลือกฟอนต์แบบดั้งเดิมโดยระบุส่วนประกอบต่าง ๆเพื่อสร้างชื่อฟอนต์ XLFD ได้แก่โปรแกรม `xfontsel`. โปรแกรมนี้จะแสดงตัวอย่างรูปทรงของอักขระให้เห็นด้วย.

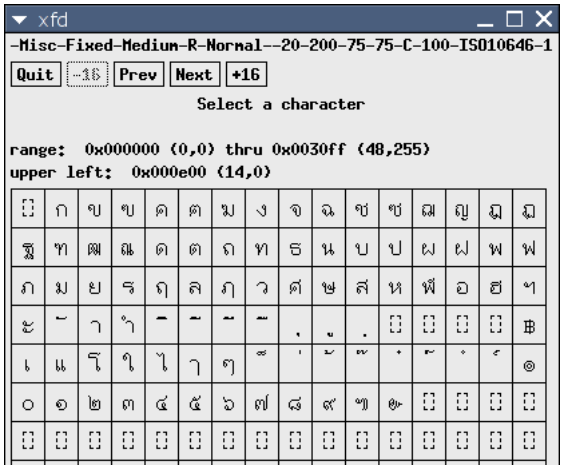
ในกรณีที่ต้องการตรวจสอบดูว่าฟอนต์หนึ่ง ๆประกอบด้วยอักขระอะไรบ้างและมีรูปร่างอย่างไร, ให้ใช้คำสั่ง `xfd` และระบุชื่อฟอนต์ที่ต้องการตรวจสอบด้วยตัวเลือก `-fn` *`fontname`* ส่วนที่เป็นชื่อฟอนต์จะเป็นชื่อแบบ XLFD หรือนามแฝงก็ได้. ในหน้าจอจะแสดง

`xfd` อ้างอิงหน้า 428



รูปที่ 6.25: เลือกฟอนต์ด้วยโปรแกรม `xfontsel`.

อักขระต่างอยู่ในตาราง, ถ้าคลิ้กอักขระใดอักขระหนึ่งก็จะแสดงค่าของอักขระนั้นด้วยเลขฐานต่าง ๆให้ดูด้วย.



รูปที่ 6.26: แสดงอักขระที่อยู่ในฟอนต์.

การติดตั้งฟอนต์ใหม่ในระบบ

การติดตั้งฟอนต์ใหม่ในระบบดั้งเดิมมีขั้นตอนต่อไปนี้.

1. สร้างไดเรกทอรีสำหรับเก็บฟอนต์
   ผู้ใช้สามารถไดเรกทอรีสำหรับเก็บฟอนต์ไว้ที่ไหนก็ได้ไม่มีกำหนดตายตัว. ฟอนต์ของระบบโดยรวมมักจะอยู่ที่ `/usr/share/fonts` ซึ่งใต้ไดเรกทอรีนั้นแบ่งแยกเป็นไดเรกทอรีย่อยต่อไปอีก. สำหรับผู้ใช้ที่ไม่สิทธิ์สร้างไดเรกทอรีย่อยใต้ไดเรกทอรีนั้นสามารถเลือกสร้างไดเรกทอรีไว้ที่ไหนก็ได้.

2. ก็อปปี้ไฟล์ฟอนต์ไว้ในไดเรกทอรีที่ต้องการ
   ในกรณีที่มีไฟล์ `.bdf` อาจจะแปลงเป็นไฟล์ `.pcf` ด้วยคำสั่ง `bdftopcf` ก่อนเพื่อให้ขนาดเล็กลง. หลังจากนั้นอาจจะใช้ `gzip` บีบอักให้มีขนาดเล็กลงอีกก็ได้.

bdftopcf อ้างอิงหน้า 426

ตัวอย่างที่ 6.35: การแปลงฟอนต์ .bdf ให้เป็น .pcf และบีบอัดข้อมูล.

```
$ bdftopcf thai8x16.bdf | gzip -c > thai8x16.pcf.gz↵
```

ถ้าเป็นฟอนต์แบบทรูไทป์หรือ PostScript ไม่ต้องทำอะไรเพิ่มเติม.

3. สั่งคำสั่ง mkfontscale
   สำหรับไดเรกทอรีที่มีฟอนต์เวคเตอร์เช่นฟอนต์ทรูไทป์หรือ PostScript, ให้สั่งคำสั่ง mkfontscale เพื่อสร้างไฟล์ fonts.scale.

mkfontscale อ้างอิงหน้า 426

ตัวอย่างที่ 6.36: สร้างไฟล์ fonts.scale.

```
$ ls↵
TlwgTypewriter.ttf  thai8x16.bdf
$ mkfontscale -a tis620-0↵                          ← ระบุรหัสอักขระเพิ่มเติม
$ ls↵
TlwgTypewriter.ttf  fonts.scale  thai8x16.bdf
$ cat fonts.scale↵
5
TlwgTypewriter.ttf -misc-tlwg typewriter-medium-r-normal--0-0-0-0-p-0-iso10646-1
TlwgTypewriter.ttf -misc-tlwg typewriter-medium-r-normal--0-0-0-0-p-0-iso8859-1
TlwgTypewriter.ttf -misc-tlwg typewriter-medium-r-normal--0-0-0-0-p-0-iso8859-11
TlwgTypewriter.ttf -misc-tlwg typewriter-medium-r-normal--0-0-0-0-p-0-tis620-0
TlwgTypewriter.ttf -misc-tlwg typewriter-medium-r-normal--0-0-0-0-p-0-tis620-2
```

คำสั่ง mkfontscale จะตรวจสอบไฟล์ฟอนต์แบบเวคเตอร์ที่อยู่ในไดเรกทอรีที่ทำงานอยู่และสร้างไฟล์ fonts.scale ให้. ตัวคำสั่งจะใช้ฐานข้อมูลรหัสอักขระที่อยู่ในไฟล์ encodings.dir และโดยปริกติไฟล์นี้จะอยู่ในไดเรกทอรี /usr/share/fonts/encodings. จากตัวอย่าง, ตัวเลือก -a tis620-0 เป็นการระบุรหัสอักขระเพิ่มเติมจากค่าปริยาย. ถ้าไม่ระบุ, ตัวโปรแกรมจะตรวจสอบว่ามีฟอนต์นั้นสามารถใช้รหัสอักขระอะไรได้บ้างแล้วเขียนไฟล์ fonts.scale ให้อัตโนมัติ.

4. สั่งคำสั่ง mkfontdir
   คำสั่ง mkfontdir จะสำรวจฟอนต์บิตแมปที่อยู่ในไดเรกทอรีที่ทำงานอยู่และไฟล์ fonts.scale เพื่อใช้เป็นข้อมูลนำเข้าสร้างไฟล์ fonts.dir. X เซิร์ฟเวอร์และฟอนต์เซิร์ฟเวอร์จะใช้ไฟล์นี้เป็นฐานข้อมูลสำหรับจับคู่ชื่อฟอนต์ XLFD และไฟล์ฟอนต์.

ตัวอย่างที่ 6.37: สร้างไฟล์ fonts.dir.

```
$ mkfontdir↵
6
TlwgTypewriter.ttf -misc-tlwg typewriter-medium-r-normal--0-0-0-0-p-0-iso10646-1
TlwgTypewriter.ttf -misc-tlwg typewriter-medium-r-normal--0-0-0-0-p-0-iso8859-1
TlwgTypewriter.ttf -misc-tlwg typewriter-medium-r-normal--0-0-0-0-p-0-iso8859-11
TlwgTypewriter.ttf -misc-tlwg typewriter-medium-r-normal--0-0-0-0-p-0-tis620-0
TlwgTypewriter.ttf -misc-tlwg typewriter-medium-r-normal--0-0-0-0-p-0-tis620-2
thai8x16.bdf -thai-fixed-medium-r-normal--16-114-100-100-p-100-tis620-0
```

5. สร้างไฟล์ `fonts.alias`
ถ้าต้องการตั้งชื่อสั้น ๆให้กับชื่อฟอนต์ XLFD ให้สร้างไฟล์ `fonts.alias` โดยมีรูปแบบดังนี้

```
<alias>    <XLFD name>
```

ตัวอย่างที่ 6.38: ไฟล์ `fonts.alias`.

```
$ cat fonts.alias↵
thai8x16 -thai-fixed-medium-r-normal--16-114-100-100-p-100-tis620-2
```

6. ทำให้ X เซิร์ฟเวอร์รับรู้ไดเรกทอรีที่เก็บฟอนต์
วิธีง่ายที่สุดที่จะทำให้ X เซิร์ฟเวอร์รับรู้ไดเรกทอรีที่เก็บฟอนต์ตัวใหม่อย่างถาวร คือการแก้ไฟล์ `xorg.conf` เพิ่มพารามิเตอร์ FontPath โดยมีค่าเป็นชื่อไดเรกทอรีที่เก็บฟอนต์ใหม่. ในกรณีนี้จะต้องเริ่มต้น X เซิร์ฟเวอร์ใหม่อีกครั้ง.

ถ้าต้องการตั้งค่า FontPath ให้ X เซิร์ฟเวอร์รับรู้ทันทีแบบชั่วคราว, สามารถทำได้โดยใช้คำสั่ง `xset`. คำสั่ง `xset` เป็นคำสั่งที่ใช้ปรับแต่งพารามเตอร์ต่าง ๆเฉพาะบุคคลในระบบ X วินโดว์. เราสามารถเพิ่มค่า FontPath ได้ดังตัวอย่างต่อไปนี้.

xset อ้างอิงหน้า 428

ตัวอย่างที่ 6.39: เพิ่ม FontPath ชั่วคราว.

```
$ xset fp+ ~/xfonts↵
$ xset q↵
--- แสดงผลต่อไปเรื่อยๆ ---
Colors:
  default colormap:  0x20    BlackPixel:  0    WhitePixel:  16777215
Font Path:
  /usr/share/fonts/misc/,/usr/share/fonts/Type1/,/usr/share/fonts/75dpi/,/usr/
share/fonts/100dpi/,/home/poonlap/xfonts/          ← ค่า FontPath ตัวใหม่
Bug Mode: compatibility mode is disabled
DPMS (Energy Star):
  Standby: 7200    Suspend: 7200    Off: 14400
  DPMS is Disabled
File paths:
  Config file:  /etc/X11/xorg.conf
  Modules path: /usr/X11R6/lib/modules
  Log file:     /var/log/Xorg.0.log
$ xset fp rehash↵
```

ในกรณีที่ใช้ฟอนต์เซิร์ฟเวอร์, ให้แก้ไฟล์ `/etc/X11/fs/config` เพิ่มไดเรกทอรีที่เก็บฟอนต์ใหม่และสั่งคำสั่ง `/etc/init.d/xfs restart`.

7. ตรวจสอบฟอนต์ใหม่ที่ติดตั้ง
ใช้คำสั่ง `xlsfonts` หรือโปรแกรม `xfd` ตรวจสอบดูว่า X เซิร์ฟเวอร์รับรู้ฟอนต์ที่ติดตั้งใหม่หรือไม่.

6.8.2 ฟอนต์ในระบบ Xft

ตามที่ได้แนะนำไปแล้วว่าในระบบฟอนต์แบบดั้งเดิมสามารถใช้ฟอนต์ได้ทั้งแบบบิตแมปและฟอนต์เวกเตอร์. อย่างไรก็ตามวิธีการจัดการฟอนต์แบบดั้งเดิมยังขาดความสมบูรณ์อยู่หลายประการเช่นชื่อฟอนต์แบบ XLFD ยาวยุ่งยากไม่เหมาะกับการใช้งานจริง, วิธีการติดตั้งและระบุการใช้ฟอนต์ยุ่งยาก ฯลฯ. สาเหตุต่าง ๆเหล่านี้เป็นที่มาของระบบฟอนต์ Xft.

Xft (X FreeType Interface library) เป็นไลบรารีสำหรับแสดงรูปทรงอักขระของฟอนต์ในระบบ X วินโดว์โดยใช้ไลบรารี FreeType. ปัจจุบันโปรแกรม GUI สมัยใหม่ที่ใช้ทูลล์คิต Gtk+ หรือ Qt หันมาใช้ไลบรารี Xft เป็นหลักทำให้การกำหนดชื่อฟอนต์ง่ายขึ้น, แสดงผลอักขระแบบแอนติอเลียสได้ ฯลฯ. ในระบบ Xft ยังแยกเป็น Xft1 และ Xft2 ซึ่งเป็นไลบรารีเดียวกันแต่คนละรุ่น, และ API แตกต่างกันทำให้ผู้ใช้และผู้พัฒนาซอฟต์แวร์ต้องระวังเวลาใช้.

ไลบรารี Xft2 เป็นไลบรารีสำหรับจัดการแสดงผล, ส่วนการจัดการปรับแต่งฟอนต์และการระบุชื่อฟอนต์ที่ต้องการใช้นั้นจะใช้ไลบรารีและชุดคำสั่งต่างหากที่เรียกว่า *fontconfig*. การจัดการฟอนต์แบบ fontconfig จะมีไฟล์ปรับแต่งของระบบโดยรวมได้แก่ `/etc/fonts/fonts.conf`. ถ้าเป็นการปรับแต่งในระดับบุคคลจะใช้ไฟล์ `~/.fonts.conf` ซึ่งมีรูปแบบของไฟล์เหมือนกันเป็นแบบ XML.

XML ▶
Extensible Markup Language. รูปแบบข้อมูลเท็กซ์ใช้คำที่กำหนดไว้เรียกว่า tag กำกับส่วนต่าง ๆให้มีความหมายต่าง ๆ. XML เป็นฟอร์แมตที่สร้างสืบทอดมาจาก SGML (Standard Generalized Markup Language) เช่นเดียวกันกับ HTML (HyperText Markup Language). HTML จะเป็นซับเซ็ตของ XML.

ไฟล์ `fonts.conf`

ในไฟล์ `fonts.conf` จะมีส่วนที่ระบุไดเรกทอรีที่เก็บฟอนต์ด้วย `<dir>...</dir>` และมีผลไปถึงไดเรกทอรีย่อยที่อยู่ใต้ไดเรกทอรีที่ระบุด้วย.

ตัวอย่างที่ 6.40: ส่วนที่ระบุไดเรกทอรีฟอนต์ในไฟล์ /etc/fonts/fonts.conf

```
$ cat /etc/fonts/fonts.conf↵
--- แสดงผลต่อไปเรื่อยๆ ---
        <dir>/usr/share/fonts</dir>
        <dir>/usr/local/share/fonts</dir>
        <dir>~/.fonts</dir>                  ← ไดเรกทอรีสำหรับเก็บฟอนต์เฉพาะบุคคล
--- แสดงผลต่อไปเรื่อยๆ ---
```

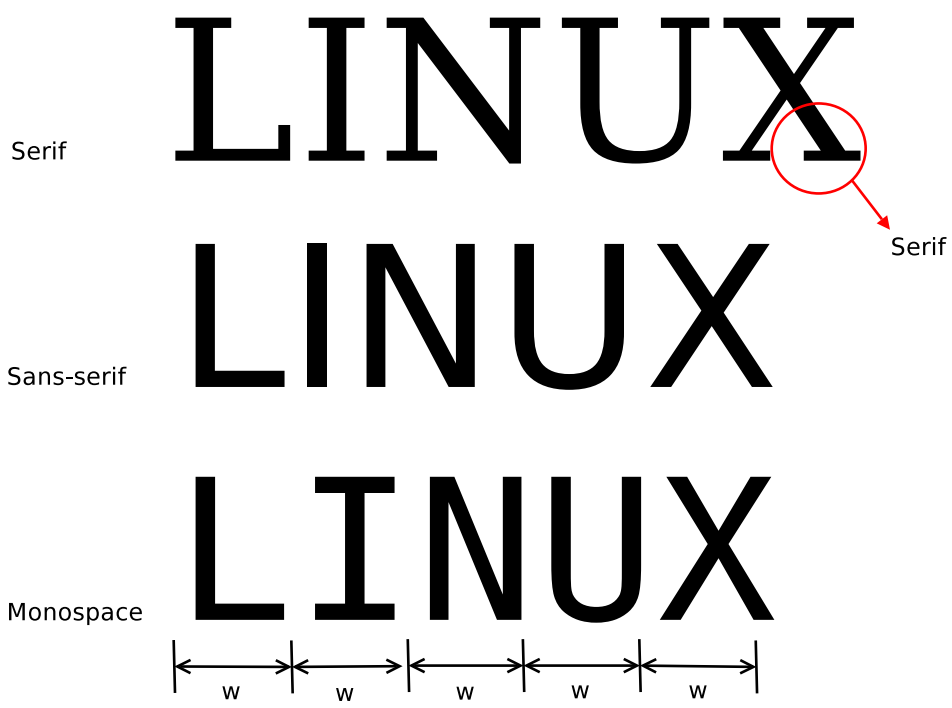
ในไฟล์ `fonts.conf` สามารถรวมไฟล์ตั้งค่าเริ่มต้นจากไฟล์อื่น ๆได้ด้วยโดยใช้ไวยกรณ์ `<include>...</include>`. ถ้าต้องการปรับแต่งระบบ fontconfig จะไม่แก้ไฟล์ `fonts.conf` โดยตรงแต่จะแก้ไขไฟล์ `/etc/fonts/local.conf` แทน. ส่วนไฟล์ `~/.fonts.conf` จะเป็นไฟล์ตั้งค่าเริ่มต้นของระบบ fontconfig เฉพาะบุคคล.

ตัวอย่างที่ 6.41: ส่วนที่รวมไฟล์ปรับตั้งค่าเริ่มอื่น ๆในไฟล์ /etc/fonts/fonts.conf.

```
$ cat /etc/fonts/fonts.conf↵
--- แสดงผลต่อไปเรื่อยๆ ---
<!--
  Load per-user customization file
-->
        <include ignore_missing="yes">~/.fonts.conf</include>
```

```
<!--
  Load local system customization file
-->
        <include ignore_missing="yes">local.conf</include>
--- แสดงผลต่อไปเรื่อยๆ ---
```

ชื่อฟอนต์สามัญและฟอนต์ที่ใช้จริง



รูปที่ 6.27: ชื่อฟอนต์สามัญที่ใช้ในระบบ fontconfig.

ในระบบจัดการฟอนต์แบบ fontconfig จะมีตระกูลฟอนต์ (family) ที่เป็นตัวแทนฟอนต์อื่น ๆที่ใช้จริงได้แก่

- Serif หมายถึงฟอนต์ที่มีส่วนประดับตรงปลายเส้นของรูปทรงอักขระ.
- Sans-serif หมายถึงฟอนต์ที่ไม่มีส่วนประดับตรงปลายเส้นของรูปทรงอักขระ.
- Monospace หมายถึงฟอนต์ที่อักขระทุกตัวมีความกว้างเท่ากัน.

ในไฟล์ `fonts.conf` มีส่วนที่กำหนดชื่อฟอนต์จริงที่ใช้แทนชื่อฟอนต์สามัญในช่วง `<alias>...</alias>`.

ตัวอย่างที่ 6.42: กำหนดชื่อฟอนต์สามัญและชื่อฟอนต์ที่ใช้จริง.

```
$ cat /etc/fonts/fonts.conf↵
--- แสดงผลต่อไปเรื่อยๆ ---
        <alias>
                <family>Bitstream Vera Serif</family>
                <family>Times New Roman</family>
                <family>Luxi Serif</family>
                <family>Kochi Mincho</family>
                <default><family>serif</family></default>
        </alias>
--- แสดงผลต่อไปเรื่อยๆ ---
```

ถ้าในแอพพลิเคชันเลือกใช้ฟอนต์ serif, ระบบ fontconfig จะเลือกฟอนต์ที่ใช้จริงจากรายชื่อฟอนต์ที่กำหนด, ในตัวอย่างได้แก่ “Bitstream Vera Serif”, “Times New Roman” ฯลฯ ตามลำดับ.

ความสามารถของ fontconfig อื่น ๆในการเลือกฟอนต์ยังมีอีกมากมายเช่นผู้ใช้สามารถตั้งเงื่อนไขถ้าขนาดของฟอนต์เล็กกว่าที่กำหนดแล้วให้ใช้ฟอนต์บิตแมปแทนการแอนติอเลียสเป็นต้น.

ชื่อฟอนต์

คำสั่งที่ใช้แสดงชื่อฟอนต์ในระบบ fontconfig ได้แก่ `fc-list`.

ตัวอย่างที่ 6.43: แสดงชื่อฟอนต์ในระบบ fontconfig.

```
$ fc-list↵
Luxi Serif:style=Regular
LucidaBright:style=Italic
Utopia:style=Bold Italic
Nimbus Sans L:style=Regular Italic
Bitstream Vera Sans Mono:style=Bold
--- แสดงผลต่อไปเรื่อยๆ ---
```

ชื่อฟอนต์ในระบบ fontconfig มีรูปแบบดังต่อไปนี้.

families-point:*name1*=*values1*:*name2*=*values2*...

- *families*
  ชื่อตระกูลฟอนต์ (ชื่อฟอนต์) เช่น Times, Luxi Serif ฯลฯ.
- *point*
  ขนาดของอักขระในหน่วยพอยต์. ตัวอย่างเช่น “Times-12” หมายถึงฟอนต์ Times ขนาด 12 พอยต์.
- *nameN*, *valueN*
  ชื่อคุณสมบัติและค่าของฟอนต์.

ตารางที่ 6.3: คุณสมบัติต่าง ๆของฟอนต์.

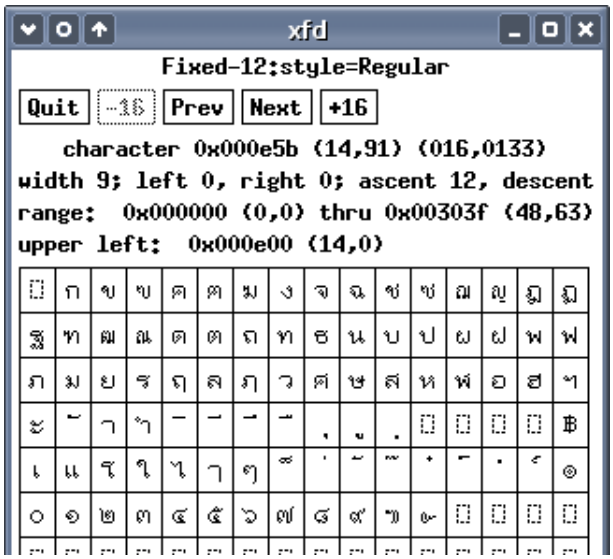
ชื่อคุณสมบัติ	ความหมาย
style	รูปทรงของอักขระเช่น Recular, Italic, Bold, Bold Italic ฯลฯ.
weight	ความหนักของอักขระได้แก่ light, medium, demibold, bold หรือ black.
size	ขนาดอักขระหน่วยเป็นพอยต์.
pixelsize	ขนาดอักขระในหน่วยพิกเซล.
spacing	คุณสมบัติความกว้างของอักขระ proportional, monospace หรือ charcell.
foundry	ชื่อตระกูลฟอนต์.
outline	ฟอนต์นั้นเป็นฟอนต์แบบเอาท์ไลน์ (true) หรือไม่ (false).
scalable	ฟอนต์นั้นเป็นฟอนต์ที่ย่อขยายได้ (true) หรือไม่ (false).
lang	ภาษาของอักขระในฟอนต์เช่น th, ja ฯลฯ.

คำสั่ง `fc-list` นอกจะใช้แสดงชื่อฟอนต์ต่าง ๆที่มีอยู่ในระบบแล้วยังสามารถแสดงชื่อฟอนต์ตามเงื่อนไขที่ต้องการโดยระบุคุณสมบัติของฟอนต์.

ตัวอย่างที่ 6.44: หาชื่อฟอนต์โดยระบุคุณสมบัติ.

```
$ fc-list :lang=th:scalable=true↵              ← ฟอนต์ภาษาไทยและสามารถย่อขยายได้
Tlwg Typewriter:style=Bold
Norasi:style=Oblique
TlwgMono:style=Medium
Norasi:style=BoldOblique
DBThaiText:style=Medium
PseudoMono:style=Bold
Loma:style=Oblique
--- แสดงผลต่อไปเรื่อยๆ ---
$ fc-list :lang=th:scalable=false↵                    ← ฟอนต์ภาษาไทยแบบบิตแมป
Fixed:style=Bold
Fixed:style=SemiCondensed
ClearlyU:style=Regular
Fixed:style=Oblique
Fixed:style=Regular
$ fc-list : family file↵                  ← แสดงชื่อไฟล์ของฟอนต์และชื่อตระกูลฟอนต์
/usr/share/fonts/75dpi/UTB___18.pcf.gz: Utopia
/usr/share/fonts/Type1/l048033t.pfa: Luxi Sans
/usr/X11R6/lib/X11/fonts/100dpi/courO12.pcf.gz: Courier
/usr/share/fonts/75dpi/timB12.pcf.gz: Times
--- แสดงผลต่อไปเรื่อยๆ ---
```

โปรแกรมที่ใช้ดูอักขระต่าง ๆที่อยู่ในฟอนต์ได้แก่ `xfd` แต่จะใช้ตัวเลือก `-fa` *`fontname`* สำหรับระบุชื่อฟอนต์ในระบบ fontconfig. ตัวอย่างเช่นคำสั่ง "`xfd -fa Fixed-12:lang=th:scala` แสดงอักขระต่าง ๆของฟอนต์ Fixed ที่เป็นภาษาไทย.



รูปที่ 6.28: ระบุชื่อฟอนต์ในระบบ fontconfig ให้กับโปรแกรม `xfd`.

ลำดับการเลือกฟอนต์ในระบบ fontconfig

สำหรับโปรแกรมที่ใช้ระบบ fontconfig และมีการใช้ฟอนต์เป็นชื่อสามัญเช่น Serif, ระบบ fontconfig จะหาฟอนต์จริงที่เหมาะสมให้กับโปรแกรมใช้แสดงผล. การใช้ชื่อฟอนต์สามัญในลักษณะนี้เป็นการลดภาระของตัวโปรแกรมโดยที่ตัวโปรแกรมไม่ต้องรับรู้ชื่อฟอนต์จริง ๆซึ่งถ้าเป็นเครื่องคอมพิวเตอร์คอมคนละเครื่องกันก็ยังใช้ชื่อฟอนต์สามัญเหมือนกันได้.

fontconfig รุ่น 2.3.x จะมีโปรแกรม `fc-match` ใช้ตรวจดูว่า fontconfig เลือกฟอนต์อะไรในการใช้งานจริง.

ตัวอย่างที่ 6.45: รายชื่อฟอนต์ที่ fontconfig จะเลือกใช้.

```
$ fc-match :family=serif↵                          ← แสดงชื่อฟอนต์ที่ใช้จริงสำหรับ serif
VeraSe.ttf: "Bitstream Vera Serif" "Roman"
$ fc-match --sort :family=serif↵                   ← แสดงรายชื่อฟอนต์ที่ใช้จริงสำหรับ serif
VeraSe.ttf: "Bitstream Vera Serif" "Roman"
n021004l.pfb: "Nimbus Roman No9 L" "Medium"
kochi-mincho-subst.ttf: "Kochi Mincoh" "Regular"
Norasi.ttf: "Norasi" "Regular"
--- แสดงผลต่อไปเรื่อยๆ ---
$ LANG=th_TH fc-match :family=serif↵
Norasi.ttf: "Norasi" "Regular"                     ← ฟอนต์ภาษาไทยที่ถูกเลือกเป็นฟอนต์ serif
```

จากตัวอย่างข้างบนจะเห็นว่าในระบบจะเลือกใช้ฟอนต์ “Bitstream Vera Serif” เป็นฟอนต์ serif โดยปริยายและถ้าไม่สามารถใช้ฟอนต์ “Bitstream Vera Serif” ได้ก็จะเลือกฟอนต์ที่เหมาะสมถัดมาตามลำดับ.

ในกรณีที่โปรแกรมต้องแสดงผลหลายภาษา, ระบบ fontconfig จะเลือกฟอนต์ที่เหมาะสมกับภาษานั้น. จากตัวอย่างที่ 6.45, fontconfig จะใช้ฟอนต์ Norasi เป็นฟอนต์ serif

เพราะเป็นฟอนต์แรกในรายการที่สามารถแสดงอักขระภาษาไทยได้. ถ้าต้องการใช้ฟอนต์อื่นเช่น Kinnari เป็นฟอนต์ serif แทนฟอนต์ Norasi ต้องปรับแต่งระบบ fontconfig โดยเพิ่มบรรทัดต่อไปนี้ไฟล์ปรับแต่ง `~/.fonts.conf` หรือ `/etc/fonts/local.conf`.

ตัวอย่างที่ 6.46: จัดลำดับฟอนต์ที่ต้องการใช้.

```
<fontconfig>
    <alias>
        <family>serif</family>
        <prefer>
            <family>Bitstream Vera Serif</family>
            <family>Kinnari</family>
            <family>Norasi</family>
        </prefer>
    </alias>
    <alias>
        <family>sans-serif</family>
        <prefer>
            <family>Bitstream Vera Sans</family>
            <family>Loma</family>
            <family>Garuda</family>
        </prefer>
    </alias>
    <alias>
        <family>monospace</family>
        <prefer>
            <family>Bistream Vera Sans Mono</family>
            <family>Tlwg Typewriter</family>
            <family>TlwgMono</family>
        </prefer>
    </alias>
</fontconfig>
```

ชื่อฟอนต์ที่อยู่ในช่วง `<prefer>...</prefer>` จะเป็นช่วงที่ผู้ใช้กำหนดลำดับฟอนต์ที่ต้องการใช้ให้ fontconfig รับรู้. ในตัวอย่างจะเอาชื่อฟอนต์ภาษาอังกฤษขึ้นนำเป็นตัวแรกเพราะเป็นฟอนต์ที่มีคุณภาพสูงและโปรแกรมที่ใช้ส่วนใหญ่จะเป็นภาษาอังกฤษ. ถ้าโปรแกรมต้องการแสดงภาษาไทยก็จะเลือกฟอนต์ภาษาไทยที่ระบุไว้ถัดมาตามลำดับ.

ตัวอย่างที่ 6.47: ผลของคำสั่ง `fc-match` หลังจากแก้ไขไฟล์ `~/.fonts.conf`.

```
$ LANG=C fc-match :family=serif↵
VeraSe.ttf: "Bitstream Vera Serif" "Roman"
$ LANG=th_TH fc-match :family=serif↵
Kinnari.ttf: "Kinnari" "Medium"
$ LANG=ja_JP fc-match :family=serif↵
kochi-mincho-subst.ttf: "Kochi Mincho" "Regular"
```

6.9 วิธีการติดตั้งฟอนต์

วิธีการติดตั้งฟอนต์ของระบบ fontconfig ง่ายกว่าระบบดั้งเดิมมาก. สรุปขั้นตอนต่าง ๆได้ดังนี้.

1. ก็อปปี้ไฟล์ฟอนต์ไว้ในไดเรกทอรีที่ระบบ fontconfig รับรู้
   ถ้าต้องการติดตั้งฟอนต์สำหรับทั้งระบบ, ให้ก็อปปี้ฟอนต์ใหม่หรือสร้างสร้างไดเรกทอรีเก็บฟอนต์ใหม่ใต้ไดเรกทอรีที่กำหนดในช่วง `<dir>...</dir>` ในไฟล์ `fonts.conf` เช่นใต้ไดเรกทอรี `/usr/share/fonts`.

   ถ้าเป็นการติดตั้งฟอนต์เฉพาะบุคคล, สามารถก็อปปี้ฟอนต์ใหม่เก็บไว้ในไดเรกทอรี `~/.fonts`.

2. สั่งคำสั่ง `fc-cache` เพื่อสร้างฐานข้อมูลฟอนต์
   คำสั่ง `fc-cache` จะสร้างไฟล์ `fonts.cache` ในไดเรกทอรีที่เก็บฟอนต์เพื่อเป็นฐานข้อมูลสำหรับการเรียกใช้ฟอนต์. โดยปรกติมักจะใช้ตัวเลือก `-fv` เพื่อบังคับให้อัปเดตฐานข้อมูลและแสดงข้อมูลการทำงานบนหน้าจอ.

   fc-cache อ้างอิงหน้า 426

   ตัวอย่างที่ 6.48: สร้างฐานข้อมูลฟอนต์ในระบบ fontconfig.

   ```
   $ fc-cache -fv↵
   fc-cache: "/usr/X11R6/lib/X11/fonts/Type1": skipping, no write access
   fc-cache: "/usr/share/fonts": skipping, no write access
   fc-cache: "/usr/local/share/fonts": skipping, no write access
   fc-cache: "/usr/X11R6/lib/X11/fonts/75dpi": skipping, no write access
   fc-cache: "/usr/X11R6/lib/X11/fonts/100dpi": skipping, no write access
   fc-cache: "/home/poonlap/.fonts": caching, 8 fonts, 0 dirs
   fc-cache: "/usr/X11R6/lib/X11/fonts/truetype": skipping, no such directory
   fc-cache: succeeded
   ```

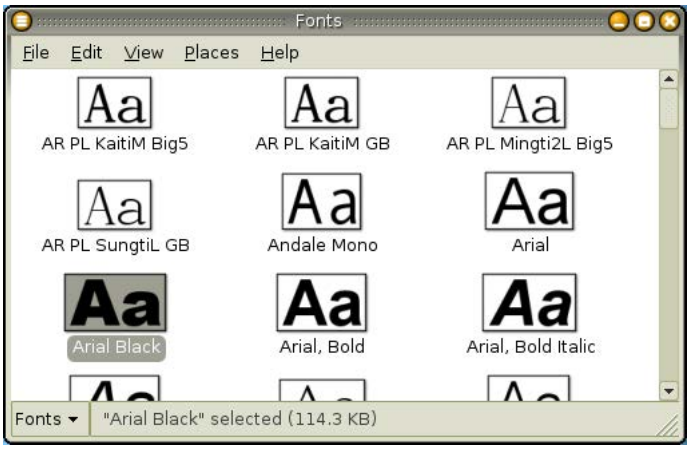
   จากตัวอย่างจะเห็นว่าตัวโปรแกรมไม่สามารเข้าถึงไดเรกทอรีบางตัวได้สามารถแก้ไขโดยการรันคำสั่งด้วยยูสเซอร์ที่มีสิทธิ์เขียนไฟล์ในไดเรกทอรีนั้นเช่น root.

สำหรับสภาพแวดล้อมเดส์กทอป GNOME, วิธีที่ง่าย ๆอีกวิธีหนึ่งคือใช้โปรแกรม `nautilus` ซึ่งเป็นไฟล์เบราเซอร์เปิดไปที่ `fonts://` เพื่อแสดงฟอนต์ต่าง ๆในระบบ. หลังจากนั้นผู้ใช้สามารถดึงและลากฟอนต์ตัวใหม่เข้าไปในหน้าต่างนั้น, จะถือว่าเป็นการติดตั้งฟอนต์ซึ่งให้ผลเหมือนกันก็อปปี้ฟอนต์ไปที่ไดเรกทอรี `~/.fonts`.

6.10 ปรับแต่งแป้นพิมพ์

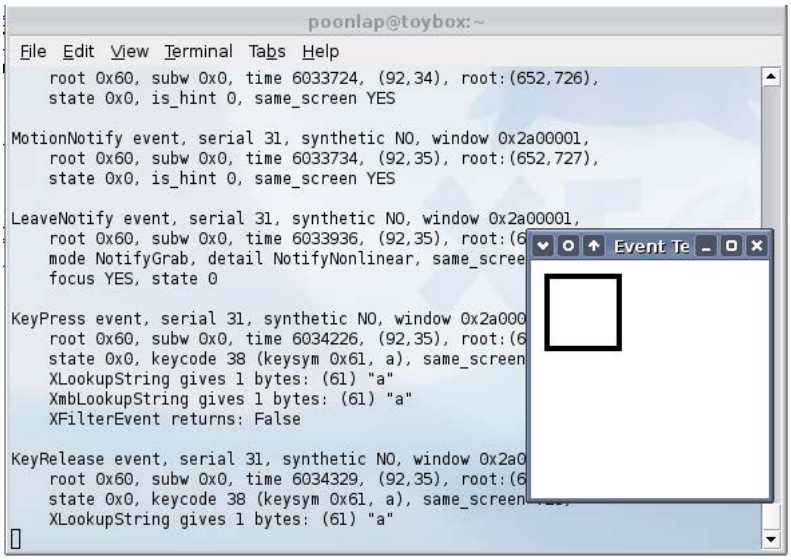
เวลาเรากดคีย์ต่าง ๆบนแป้นพิมพ์, X เซิร์ฟเวอร์จะรับรู้คีย์ที่กดเป็นคีย์โค้ดและแปลงคีย์โค้ดนั้นเป็นอักขระ (คีย์ซิม) ตามตารางที่เตรียมไว้. ตารางสำหรับแปลงคีย์โค้ดเป็นคีย์ซิมนี้เรียกว่า*คีย์แมป (keymap)*.

คีย์แต่ละคีย์บนแป้นพิมพ์ที่กดจะมีค่าคีย์โค้ดและคีย์ซิม. ค่าเหล่านี้สามารถสำรวจได้จากโปรแกรม `xev`. โปรแกรม `xev` เป็นโปรแกรมมาตรฐานในระบบ X วินโดว์สำหรับ



รูปที่ 6.29: หน้าจอ nautilus แสดงฟอนต์ต่าง ๆในระบบ.

แสดงอิเวนต์ (event) ต่าง ๆในระบบ X วินโดว์. ตัวโปรแกรมจะแสดงหน้าต่างว่าง ๆ, ถ้ามีอิเวนต์ต่าง ๆเกิดขึ้นเช่นการกดคีย์บนแป้นพิมพ์, คลิ้กเมาส์ ก็จะแสดงรายละเอียดของอิเวนต์ที่เกิดขึ้นทางเทอร์มินอลที่รันโปรแกรมนั้น.



รูปที่ 6.30: สำรวจค่าคีย์โค้ดและคีย์ซิมด้วย xev.

ตัวอย่างที่ 6.49: สำรวจค่าคีย์โค้ดและคีย์ซิมด้วยคำสั่ง xev.

```
$ xev &↵
--- แสดงผลต่อไปเรื่อยๆ ---
KeyPress event, serial 31, synthetic NO, window 0x2c00001,          ← เกิดการกดคีย์
    root 0x60, subw 0x2c00002, time 10005895, (24,44), root:(1116,842),
    state 0x0, keycode 38 (keysym 0x61, a), same_screen YES,  ← ค่าคีย์โค้ดและคีย์ซิม
    XLookupString gives 1 bytes: (61) "a"
    XmbLookupString gives 1 bytes: (61) "a"
```

```
    XFilterEvent returns: False

KeyRelease event, serial 31, synthetic NO, window 0x2c00001,    ← เกิดการปล่อยคีย์
    root 0x60, subw 0x2c00002, time 10005975, (24,44), root:(1116,842),
    state 0x0, keycode 38 (keysym 0x61, a), same_screen YES,
    XLookupString gives 1 bytes: (61) "a"
--- แสดงผลต่อไปเรื่อยๆ ---
```

ถ้ามีการกดคีย์ “a”, คำสั่ง xev ก็จะแสดงอิเวนต์ตอนกด (KeyPress event) และอิเวนต์ตอนปล่อยคีย์ (KeyRelease event) พร้อมกับค่าคีย์โค้ดและคีย์ซิมของคีย์ที่กดตามตัวอย่างที่ 6.49. เมื่อรู้ค่าคีย์โค้ดของคีย์ที่ต้องการ, ผู้ใช้สามารถปรับแต่งคีย์โค้ดให้สัมพันธ์กับคีย์ซิมตามที่ต้องการด้วยคำสั่ง xmodmap.

ถ้าคลิ้กเมาส์ปุ่มต่าง ๆก็จะแสดงเลขปุ่มของเมาส์ที่คลิ้กเช่นปุ่มซ้ายมีหมายเลขเป็น 1 ฯลฯ.

คำสั่ง xmodmap เป็นคำสั่งที่ใช้ปรับแต่งคีย์ต่าง ๆและแสดงคีย์แมปของแป้นพิมพ์ได้ด้วย. ตัวอย่างต่อไปนี้เป็นผลลัพธ์ของคำสั่ง xmodmap กับตัวเลือก -pke แสดงค่าคีย์โค้ดและคีย์ซิมของแป้นพิมพ์ภาษาไทย.

ตัวอย่างที่ 6.50: แสดงคีย์โค้ดและคีย์ซิมด้วยคำสั่ง xmodmap.

```
$ xmodmap -pke↵
keycode   8 =
keycode   9 = Escape
keycode  10 = 1 exclam Thai_baht Thai_lakkhangyao
keycode  11 = 2 at slash Thai_leknung
keycode  12 = 3 numbersign minus Thai_leksong
keycode  13 = 4 dollar Thai_phosamphao Thai_leksam
keycode  14 = 5 percent Thai_thothung Thai_leksi EuroSign
--- แสดงผลต่อไปเรื่อยๆ ---
```

คีย์โค้ดที่แสดงจะเป็นตัวเลขฐานสิบและคีย์ซิมเช่น Escape, 1, exclam, Thai_baht ฯลฯ เป็นสายอักขระที่กำหนดไว้ในไฟล์ /usr/include/X11/keysymdef.h.

จะเห็นได้ว่าคีย์โค้ดหนึ่งสามารถมีค่าคีย์ซิมได้มากกว่าหนึ่งตัว. ถ้าเป็นแป้นพิมพ์ภาษาอังกฤษอย่างเดียวคีย์โค้ดหนึ่งตัวจะมีคีย์ซิมที่ตั้งค่าไว้ไม่เกิน 2 ตัว. คีย์ซิมตัวแรกจะเป็นค่าอักขระเมื่อกดคีย์นั้นและคีย์ซิมตัวที่สองจะเป็นค่าอักขระที่กดคีย์นั้นพร้อมกับคีย์ Shift. ในกรณีที่เป็นแป้นพิมพ์ภาษาอักกฤษกับภาษาที่สองอื่น ๆเช่นแป้นพิมพ์ภาษาไทย, คีย์โค้ดหนึ่งตัวจะมีคีย์ซิมที่ตั้งค่าไว้ 2 กลุ่ม, แต่ละกลุ่มจะมีคีย์ซิมอยู่ 2 ค่า. กลุ่มแรกจะเป็นกลุ่มคีย์ซิมที่เกี่ยวข้องกับภาษาอังกฤษและกลุ่มที่สองจะเป็นคีย์ซิมที่เกี่ยวข้องกับภาษาไทย. การปรับแต่งแป้นพิมพ์ให้ใช้ภาษาไทยสามารถใช้คำสั่ง xmodmap ปรับแต่งได้, แต่จะยุ่งยากและถือเป็นวิธีที่ล้าสมัยไปแล้ว. ในปัจจบันมักจะนิยมใช้คุณสมบัติของ X เซิร์ฟเวอร์ได้แก่ XKB ปรับแต่งแป้นคีย์บอร์ดภาษาไทยแทน.

ตัวอย่างการใช้งานจริงของคำสั่ง xmodmap เช่นการสลับตำแหน่งสลับตำแหน่งของคีย์ Control กับตำแหน่งของ Caps Lock. ในกรณีนี้จะต้องเตรียมไฟล์ที่รวมคำสั่งที่ใช้กับ xmodmap และใช้ชื่อไฟล์นั้นเป็นอาร์กิวเมนต์ของคำสั่ง xmodmap.

ตัวอย่างที่ 6.51: สลับตำแหน่งคีย์ Control และ Caps Lock โดยใช้ xmodmap.

```
$ cat ~/.Xmodmap↵                    ← ไฟล์ที่เตรียมคำสั่งสำหรับ xmodmap
```

```
remove Lock = Caps_Lock          ← เอาคีย์โมดิไฟเออร์ Lock ออกจากคีย์ซิม Caps_Lock
remove Control = Control_L     ← เอาคีย์โมดิไฟเออร์ Control ออกจากคีย์ซิม Control_L
keysym Control_L = Caps_Lock
keysym Caps_Lock = Control_L          ← สลับคีย์ซิมระหว่าง Control_L กับ Caps_Lock
add Lock = Caps_Lock                 ← เพิ่มโมดิไฟเออร์คีย์ Lock ให้คีย์ซิม Caps_Lock
add Control = Control_L            ← เพิ่มโมดิไฟเออร์คีย์ Control ให้คีย์ซิม Control_L
$ xmodmap ~/.Xmodmap↵
```

ในตัวอย่างจะเตรียมไฟล์ชื่อ `~/.Xmodmap` ซึ่งดิสโทรส่วนใหญ่จะใช้คำสั่ง `xmodmap` เรียกอ่านไฟล์นี้โดยอัตโนมัติตอนเริ่ม X เซิร์ฟเวอร์.

ผู้อ่านสามารถอ่านรายละเอียดไวยกรณ์ของคำสั่ง `xmodmap` ได้จาก `man xmodmap` ซึ่งจะไม่อธิบายในหนังสือเล่มนี้.

6.10.1 ปรับแต่งแป้นฟิมพ์ด้วย XKB

การปรับแต่งแป้นพิมพ์ด้วย `xmodmap` เป็นวิธีแบบดั้งเดิม, ปัจจุบันจะใช้ XKB ปรับแต่งแป้นพิมพ์แทนซึ่งจะง่ายกว่าแบบเดิม. การปรับแต่งแป้นพิมพ์ด้วย XKB สามารถทำได้ในระบบโดยรวมด้วยการปรับแต่งไฟล์ `xorg.conf` ซึ่งเป็นไฟล์ตั้งค่าเริ่มต้นของ X เซิร์ฟเวอร์หรือใช้คำสั่งที่เกี่ยวข้องกับ XKB ปรับแต่งแป้นพิมพ์.

ปรับแต่งแป้นพิมพ์ภาษาไทยสำหรับระบบโดยรวม

การปรับแต่งแป้นพิมพ์ภาษาไทยสำหรับระบบโดยรวมทำได้โดยเพิ่มบรรทัดต่อไปนี้ในไฟล์ `xorg.conf` ในเซกชัน “InputDevice”.

```
Option    "XkbLayout"     "us,th_tis"
Option    "XkbOptions"    "grp:alt_shift_toggle,grp_led:scroll,lv3:ralt_switch"
```

ความหมายของพารามิเตอร์ที่เกี่ยวข้องได้อธิบายไปแล้วในหน้าที่ 268.

ปรับแต่งแป้นพิมพ์ภาษาไทยเฉพาะบุคคล

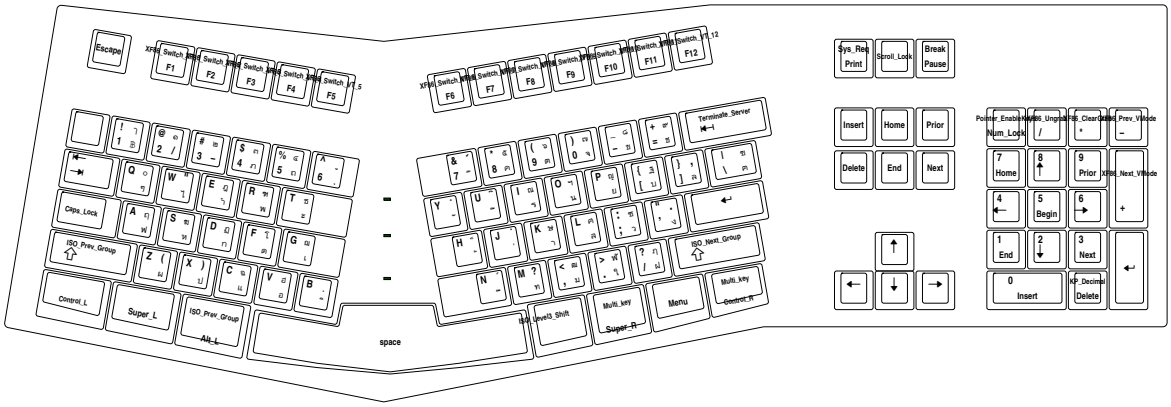
โปรแกรมคำสั่งที่เกี่ยวข้องกับ XKB มีหลายคำสั่ง, ส่วนคำสั่งที่ใช้ปรับแต่งแป้นพิมพ์โดยตรงซึ่งยูสเซอร์ทั่วไปสามารถใช้ได้ด้วยได้แก่ `setxkbmap`. คุณสมบัติสำคัญ ๆที่ปรับแต่งได้ด้วยคำสั่ง `setxkbmap` ได้แก่

- ประเภทของแป้นพิมพ์เชิงฮาร์ดแวร์ (model)
  ถ้าแบ่งประเภทของแป้นพิมพ์ในเชิงฮาร์ดแวร์แล้วสามารถแยกออกได้เป็นหลายประเภท. แป้นพิมพ์บางรุ่นมีปุ่มพิเศษเช่นปุ่มวินโดวส์, บ้างก็ไม่มีปุ่มนี้ ฯลฯ. XKB แบ่งประเภทของแป้นพิมพ์เชิงฮาร์ดแวร์ไว้หลาย ๆแบบโมเดล (model). ประเภทของแป้นพิมพ์ที่ใช้กันทั่วไปคือ pc101 ซึ่งหมายถึงแป้นพิมพ์ทั่วไปที่มีคีย์ 101 คีย์ เช่นตัวอย่างในรูปที่ 6.16. นอกจากนี้ยังมีแป้นพิมพ์ประเภทอื่น ๆอีกเช่น pc102, pc104, pc105, jp106, microsoft (Microsoft Natural keyboard) เป็นต้น.

  คำสั่ง `setxkbmap` สามารถใช้เลือกประเภทของแป้นด้วยตัวเลือก `-model` ตามด้วยชื่อประเภทแป้นพิมพ์ต่าง ๆในไฟล์ `/etc/X11/xkb/rules/xorg.lst`.

ตัวอย่างที่ 6.52: ปรับแต่งประเภทแป้นพิมพ์.

```
$ setxkbmap -model microsoft↵
```



รูปที่ 6.31: แป้นพิมพ์ Microsoft Natural ซึ่งจำนวนปุ่มและรูปร่างต่างจากแป้นพิมพ์อื่น ๆ.

- ผังแป้นพิมพ์ (layout)
  ผังแป้นพิมพ์คือประเภทของแป้นพิมพ์ในเชิงซอฟต์แวร์, ได้แก่การมอบหมายให้คีย์ต่าง ๆแทนอักขระที่กำหนดไว้. ผังแป้นพิมพ์ยังเกี่ยวข้องกับภาษาเช่นแป้นพิมพ์แบบ pc101 สามารถเป็นได้ทั้งแป้นพิมพ์ภาษาอังกฤษและแป้นพิมพ์ภาษาไทย. ในบางกรณีแป้นพิมพ์ภาษาเดียวกันยังสามารถมีผังแป้นพิมพ์ได้หลายแบบ. สำหรับภาษาอังกฤษมีผังแป้นพิมพ์หลัก ๆได้แก่ qwerty และ dvorak. สำหรับแป้นพิมพ์ภาษาไทยมีผังแป้นพิมพ์แบบเกษมณี, ปัตตโชติ และ มอก.820.

  คำสั่ง `setxkbmap` ใช้เลือกผังแป้นพิมพ์ด้วยตัวเลือก `-layout`. ผู้ใช้สามารถระบุชื่อผังแป้นพิมพ์ได้มากกว่าหนึ่งแบบโดยใช้เครื่องหมาย , คั่นระหว่างชื่อผังแป้นพิมพ์.

ตัวอย่างที่ 6.53: ปรับแต่งผังแป้นพิมพ์.

```
$ setxkbmap -layout us,th_tis,fr↵ ← ตั้งผังแป้นพิมพ์ภาษาอังกฤษ, ไทยและฝรั่งเศส
```

สลับเปลี่ยนผังแป้นพิมพ์

ในเวลาขณะใดขณะหนึ่ง, ผู้ใช้สามารถเลือกใช้ผังแป้นพิมพ์ได้อย่างเดียวเท่านั้น. การเปลี่ยนผังแป้นพิมพ์มักจะใช้การกดคีย์พิเศษที่กำหนดไว้สลับเปลี่ยนผังแป้นพิมพ์ไปมา เช่นถ้าการกดคีย์ `Alt+Shift` สำหรับสลับเปลี่ยนผังแป้นพิมพ์. ผู้ใช้สามารถกำหนดคีย์สำหรับเปลี่ยนผังแป้นพิมพ์ได้โดยใช้ตัวเลือก `-option` กับคำสั่ง `setxkbmap`.

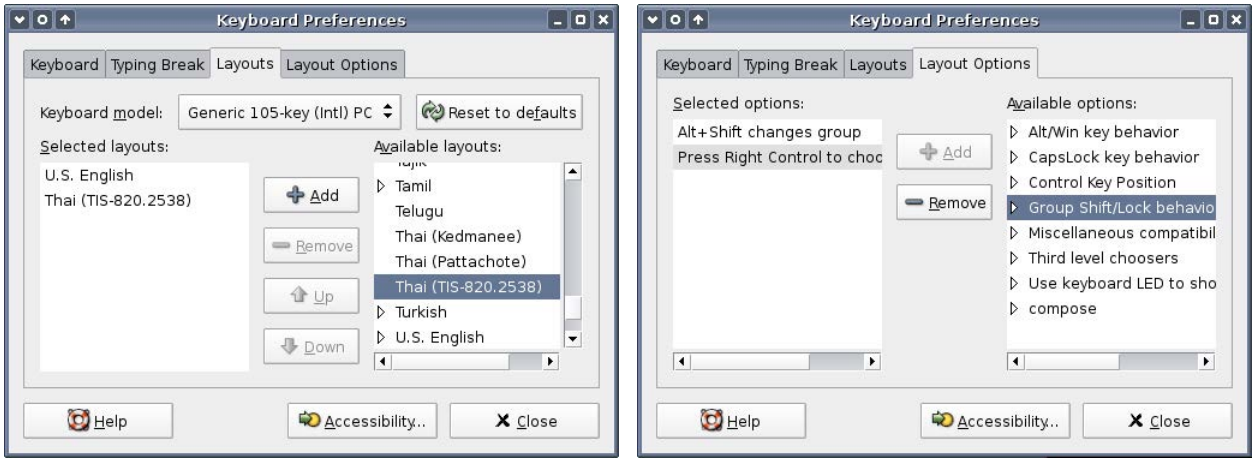
ตัวอย่างที่ 6.54: ระบุวิธีการเปลี่ยนผังแป้นพิมพ์.

```
$ setxkbmap -option grp:alt_shift_toggle↵
```

ในตัวอย่างเป็นการกำหนดค่า `grp:alt_shift_toggle` หมายถึงการกดคีย์ `Alt` ค้างไว้แล้วกดคีย์ `+Shift` สำหรับเปลี่ยนผังแป้นพิมพ์. ระบบ XKB ยังมีวิธีการเปลี่ยนผังแป้นพิมพ์อื่น ๆอีกเช่น `grp:menu_toggle` กดคีย์เมนู, `grp:lwin_toggle` กดคีย์วินโดวส์เป็นต้น. ค่าอื่น ๆเหล่าที่สามารถกำหนดได้และคำอธิบายมีเขียนไว้อยู่ในไฟล์ `xorg.lst`.

การปรับแต่งแป้นพิมพ์เฉพาะบุคคลในสภาพแวดล้อมเดสก์ท็อป GNOME

ในสภาพแวดล้อมเดสก์ท็อปเช่น GNOME ผู้ใช้สามารถปรับแต่งแป้นพิมพ์ให้มีคุณสมบัติต่าง ๆรวมถึงการปรับแต่งแป้นพิมพ์ให้ใช้ภาษาไทยได้จากเมนูปรับแต่งระบบที่เตรียมไว้ด้วยโปรแกรม `gnome-keyboard-properties`. ในสภาพแวดล้อม GNOME จะใช้ XKB ปรับแต่งแป้นพิมพ์ต่างหากแยกจากการปรับแต่งแป้นพิมพ์ด้วยไฟล์ `xorg.conf`. ถ้ามีการปรับแต่งแป้นพิมพ์ด้วย XKB จากไฟล์ `xorg.conf` และปรับแต่งแป้นพิมพ์อีกทีด้วย GNOME, ในระบบจะใช้การปรับแต่งแป้นพิมพ์จาก GNOME เป็นหลัก.

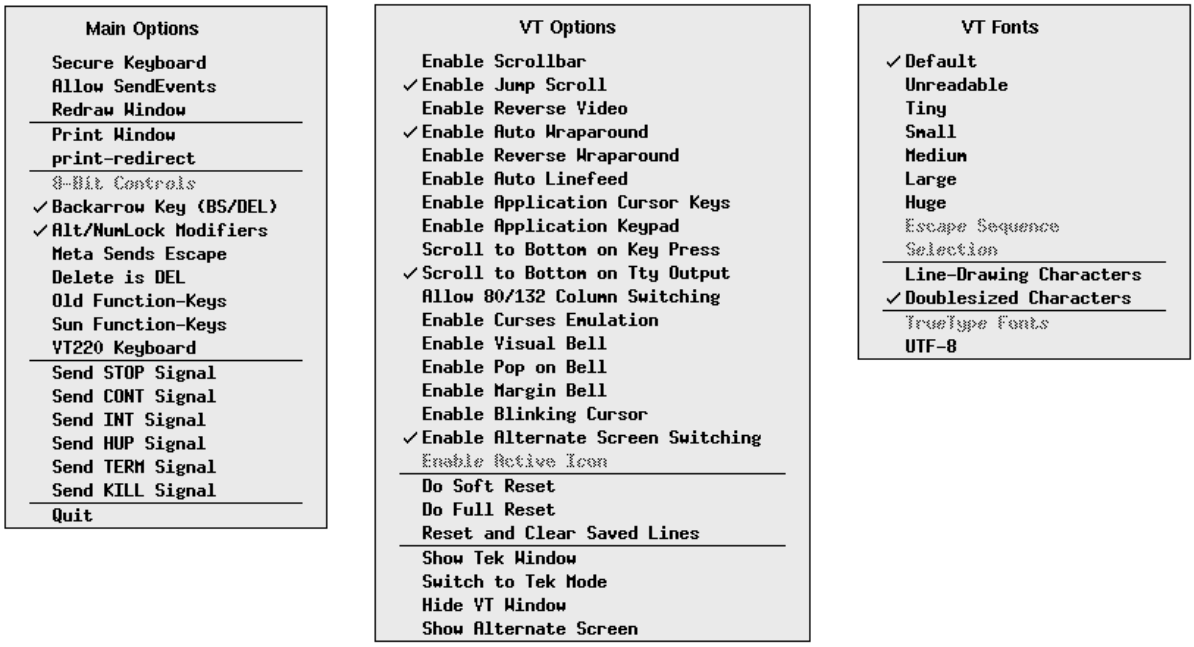


รูปที่ 6.32: โปรแกรม `gnome-keyboard-properties` สำหรับปรับแต่งแป้นพิมพ์.

6.11 เทอร์มินอลเอมิวเลเตอร์

เทอร์มินอลเอมิวเลเตอร์เป็นโปรแกรมที่จำลองเทอร์มินอลสำหรับสั่งคำสั่งด้วยเชลล์. ในระบบ X วินโดว์มีโปรแกรมเทอร์มินอลเอมิวเลเตอร์มาตรฐานได้แก่ `xterm`. โปรแกรม `xterm` เป็นโปรแกรมที่เรียบง่ายไม่มีเมนูบาร์เหมือนโปรแกรมสมัยใหม่ทั่วไป. ถ้าผู้ใช้กดคีย์ `Ctrl` ค้างไว้แล้วกดปุ่มซ้าย, กลาง, หรือขวาของเมาส์, ตัวโปรแกรมจะแสดงเมนูปรับแต่งค่าต่าง ๆที่แสดงในรูปที่ 6.33. ค่าที่ปรับแต่งจากเมนูบางตัวสามารถกำหนดจากตัวเลือกของโปรแกรมได้ด้วย.

สิ่งที่ผู้ใช้ควรรู้เกี่ยวกับการใช้เทอร์มินอลได้แก่การเลื่อนดูหน้าจอที่แสดงผ่านไปแล้ว. ผู้ใช้สามารถใช้เมาส์เลื่อนเนื้อหาขึ้นลงได้, หรือถ้าสะดวกใช้แป้นพิมพ์สามารถใช้คีย์ `Shift+PgUp`



รูปที่ 6.33: เมนูของ xterm เมื่อกดคีย์ Ctrl และเมาส์ปุ่มต่าง ๆ.

เพื่อเลื่อนหน้าจอขึ้น, ใช้คีย์ Shift+PgDn เลื่อนหน้าจอลงแทนการใช้เมาส์ได้. การเลื่อนหน้าจอด้วยแป้นพิมพ์นี้สามารถใช้ได้กับเทอร์มินอลอื่น ๆที่ไม่ใช่ xterm ได้ด้วย.

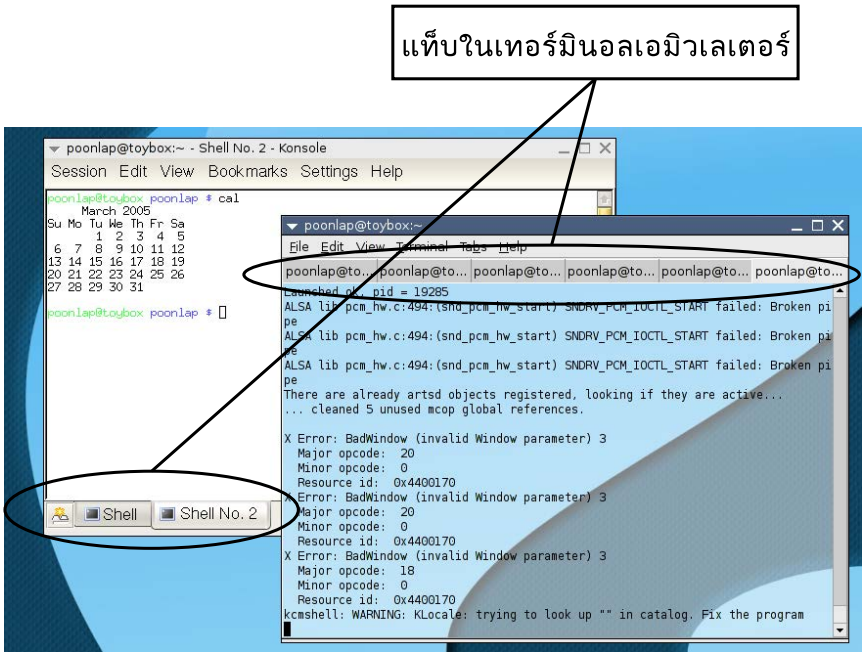
เมื่อเปรียบเทียบ xterm กับเทอร์มินอลสมัยใหม่อื่น ๆเช่น gnome-terminal, konsole ฯลฯ ในแง่ของการใช้งานบรรทัดคำสั่งไม่มีความแตกต่าง. ในแง่ของความใช้ง่ายและการตกแต่ง, เทอร์มินอลสมัยอำนวยความสะดวกเกี่ยวกับการปรับแต่งตัวเทอร์มินอลเช่น ฟอนต์, ฉากหลัง ฯลฯ ปรับแต่งง่ายกว่า xterm. ข้อแตกต่างอีกประการคือเทอร์มินอลสมัยใหม่สามารถแสดงหน้าจอเชลล์ได้หลายตัวภายในหน้าต่างเดียวโดยใช้แท็บ (tab) ไม่ต้องเปิดเทอร์มินอลหลายบาน. ตัวอย่างเช่นใน gnome-terminal สามารถสร้างแท็บใหม่ด้วยการกด Ctrl+Shift+t หรือเลือกสร้างแท็บจากเมนู. ผู้ใช้สร้างมารถเลือกแท็บที่ต้องการโดยใช้เมาส์คลิ้กหรือกดคีย์ Ctrl กับคีย์ PnUp, PnDn เปลี่ยนหน้าจอแท็บไปมาได้.

6.11.1 บันทึกหน้าจอเทอร์มินอล

เทอร์มินอลเป็นอินเทอร์เฟสพื้นฐานที่ใช้อักขระโต้ตอบกับผู้ใช้, ดังนั้นการถ่ายทอดสิ่งที่ทำในบรรทัดคำสั่งจึงง่ายกว่าโปรแกรมระบบ GUI คือผู้ใช้สามารถก็อปปี้สิ่งที่แสดงทางหน้าจอลงในไฟล์แล้วส่งต่อให้ผู้อื่นรู้ได้.

การบันทึกสิ่งที่พิมพ์และสิ่งที่แสดงผลทางเทอร์มินอลแบบพื้นฐานสามารถทำได้โดยใช้เมาส์ก็อปปี้ส่วนแสดงผลที่ต้องการแล้วแปะบันทึกลงไฟล์. วิธีนี้เหมาะกับการบันทึกสิ่งที่กระทำในเทอร์มินอลช่วงเวลาสั้น ๆ. ถ้าต้องการบันทึกการทำงานทั้งหมดในเทอร์มินอลเป็นระยะเวลานาน, ให้ใช้คำสั่ง script. คำสั่ง script จะเริ่มเชลล์ตัวใหม่และเก็บ

script อ้างอิงหน้า 415



รูปที่ 6.34: แท็บในเทอร์มิวนอลเอมิวเลเตอร์สมัยใหม่.

บันทึกที่แสดงในเทอร์มินอลทั้งหมดลงในไฟล์ที่กำหนด. วิธีนี้เหมาะสำหรับเก็บบันทึกการกระทำทุกอย่างในเทอร์มินอลแล้วเปิดดูภายหลัง.

ตัวอย่างที่ 6.55: บันทึกหน้าจอเทอร์มินอลด้วยคำสั่ง `script`.

```
$ script typescript.txt↵          ← บันทึกสิ่งที่แสดงบนเทอร์มินอลในไฟล์ typescript.txt
Script started, file is typescript.txt
$ fortune↵
"In matters of principle, stand like a rock; in matters of taste, swim with
the current."
-- Thomas Jefferson
$ exit↵                                                          ← จบการบันทึก
Script done, file is typescript.txt
$ cat typescript.txt↵                                            ← แสดงสิ่งที่บันทึก
Script started on Thu Mar 24 01:03:10 2005
$ fortune
"In matters of principle, stand like a rock; in matters of taste, swim with
the current."
-- Thomas Jefferson
$ exit

Script done on Thu Mar 24 01:03:39 2005
```

6.12 สภาพแวดล้อมเดสก์ท็อป GNOME

GNOME (GNU Network Object Model Environment) เป็นสภาพแวดล้อมเดสก์ท็อปหนึ่งที่นิยมใช้ในระบบปฏิบัติการลินุกซ์มีจุดมุ่งหมายหลักได้แก่การเสนอสภาพ

แวดล้อมเดสก์ท็อปที่ง่ายต่อการใช้งานสำหรับผู้ใช้ทั่วไป, และเสนอแพลตฟอร์มการพัฒนาแอพพลิเคชันให้สามารถเข้ากันได้ในเดสก์ท็อป. ก่อนหน้าที่มีแนวคิดเรื่องสภาพแวดล้อมเดสก์ท็อป, โปรแกรม GUI ต่าง ๆที่มีใช้ไม่มีรูปแบบที่แน่นอนเช่นบางโปรแกรมมีเมนูบาร์, บางโปรแกรมไม่มี. บางโปรแกรมทีหน้าต่างเอกสารวิธีใช้งาน, บางโปรแกรมไม่มี. สภาพแวดล้อมเดสก์ท็อปเป็นเหมือนมาตรฐานให้โปรแกรมแบบ GUI ต่าง ๆที่ใช้งานบนเดสก์ท็อปมีความเข้ากันได้, มีรูปแบบที่เหมือนกัน. โปรแกรมแต่ละตัวจะดูคล้ายเหมือนกันเพราะสร้างด้วยไลบรารีร่วมที่เหมือนกันและยึดรูปแบบเดียวกัน. แอพพลิเคชันหรือโปรแกรมทุกตัวที่ถือว่าเป็นส่วนหนึ่งในสภาพแวดล้อมเดสก์ท็อป GNOME จะใช้ไลบรารี Gtk+ เหมือนกัน, มีเมนูมาตรฐานได้แก่เมนู File, Edit, Help ฯลฯ, มีตัวเลือกของบรรทัดคล้ายเหมือนกัน เป็นต้น. ในที่นี้จะแนะสภาพแวดล้อมเดสก์ท็อปโดยอ้างอิงจาก GNOME รุ่น 2.6 หรือ 2.8.

สภาพแวดล้อมเดสก์ท็อป GNOME เป็นโครงการที่เริ่มสร้างในราวเดือนสิงหาคม ปีค.ศ. 1997 โดยมี Miguel de Icaza และ Federico Mena ชาวเมกซิกันขณะที่ยังเป็นนักศึกษา. ในช่วงเวลานั้นมีโครงการเด่น ๆได้แก่ KDE ซึ่งเป็นสภาพแวดล้อมเดสก์ท็อปใหม่อีกโครงการเช่นกันแต่มีปัญหาเกี่ยวกับสิทธิ์อนุญาตการใช้. KDE ใช้ไลบรารี Qt ซึ่งในตอนนั้นยังมีสิทธิ์อนุญาตการใช้งานที่ไม่ใช่ GPL ทำให้มีปัญหาว่าจะขัดกับการแก้ไขรหัสต้นฉบับของ KDE หรือไม่. ต่อมาไม่นาน Miguel และ Federoco ก็ได้เริ่มโครงการ GNOME อย่างจริงจังโดยใช้ไลบรารี Gtk+ ที่พัฒนาสำหรับใช้กับโปรแกรม `gimp` เป็นไลบรารีหลักในการสร้างสภาพแวดล้อมเดสก์ท็อป GNOME และได้รับความนิยมในเวลาต่อมาจนถึงปัจจุบัน.

ในสภาพแวดล้อมเดสก์ท็อปจะประกอบด้วยแอพพลิเคชันหรือโปรแกรมต่าง ๆเข้าด้วยกันเป็นระบบ. ส่วนประกอบที่สำคัญ ๆของสภาพแวดล้อมได้แก่

6.12.1 พาเนล (penel)

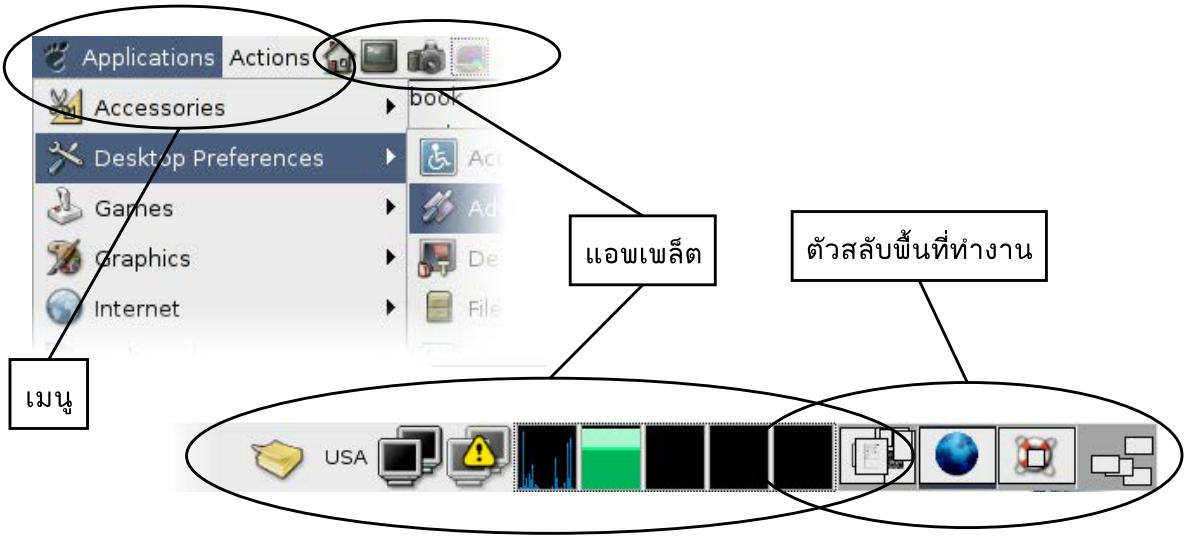
พาเนล (โปรแกรม `gnome-panel`) คือแถบที่อยู่บริเวณขอบหน้าจอเป็นพื้นที่สำหรับใส่วัตถุต่าง ๆเช่นเมนูในสภาพแวดล้อมเดสก์ท็อป. พาเนลสามารถอยู่บริเวณขอบหน้าจอด้านไหนก็ไหนและมีจำนวนได้มากกว่าหนึ่งตัว. ผู้ใช้สามารถลากไปไว้บริเวณขอบที่ต้องการ. การปรับแต่งพาเนลทำได้โดยคลิ้กเมาส์ปุ่มขวาแล้วเลือกเพิ่มแอพเพล็ต, เพิ่มเมนูจากเมนูที่เตรียมไว้ให้.

6.12.2 เมนู (menu)

เมนูเป็นปุ่มแสดงรายการแอพพลิเคชัน (applications menu) หรือการกระทำต่าง ๆ (actions menu) ในพาเนลให้ผู้ใช้เลือกใช้.

6.12.3 หน้าต่าง (window)

หน้าต่างของแอพพลิเคชันหรือโปรแกรมต่างจะควบคุมโดยวินโดว์แมนเนเจอร์เช่น การย่อขยายหน้าต่าง, การเลื่อนหน้าต่าง ฯลฯ. ผู้ใช้สามารถใช้วินโดว์แมนเนเจอร์อะไรก็ได้



รูปที่ 6.35: พาเนลในสภาพแวดล้อมเดสก์ท็อป GNOME.

ปัจจุบันวินโดว์แมนเนเจอร์ปริยายของ GNOME คือ `metacity`.

ที่มีคุณสมบัติของวินโดว์แมนเนเจอร์พื้นฐานและเข้ากันได้กับ GNOME เช่น `metacity`, `sawfish` ฯลฯ. เวลา GNOME เรียกใช้วินโดว์แมนเนเจอร์จะใช้โปรแกรม `gnome-wm` เป็นตัวกลางโดยไม่เรียกใช้โปรแกรมวินโดว์แมนเนเจอร์โดยตรง.

6.12.4 พื้นที่ทำงาน (workspace)

ในสภาพแวดล้อม GNOME สามารถมีพื้นที่ทำงานเป็นหน้าจอได้หลายบาน. โดยส่วนใหญ่ระบบจะตั้งจำนวนพื้นที่ทำงานไว้ให้โดยปริยาย 4 หน้าจอ. ผู้ใช้สามารถสลับเปลี่ยนพื้นที่ทำงานจากหน้าจอหนึ่งไปอีกหน้าจอหนึ่งด้วยการคลิ้กตัวสลับพื้นที่ทำงานหรือใช้แป้นพิมพ์ลัด (shortcut) กด `Ctrl+Alt` พร้อมกับคีย์ลูกศรซ้าย, ขวา, บน, หรือล่าง.

6.12.5 โปรแกรมจัดการแฟ้มข้อมูล (file manager)

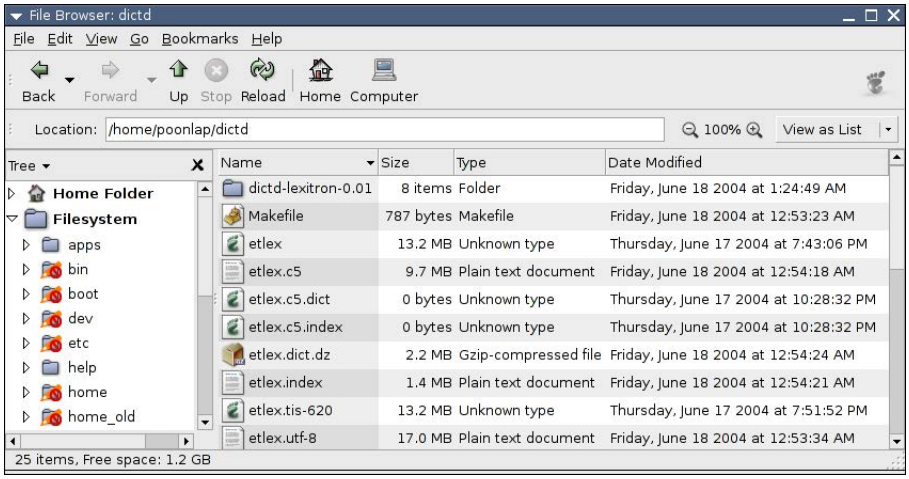
โปรแกรม `nautilus` เป็นโปรแกรมจัดการไฟล์ต่าง ๆในสภาพแวดล้อมเดสก์ท็อป GNOME, มีหน้าที่แสดงและจัดการไฟล์หรือไดเรกทอรีต่าง ๆด้วย GUI. นอกจากนี้ยังทำหน้าที่เป็นพื้นเดสก์ท็อปที่เห็นบนหน้าจอ, และเป็นโปรแกรมศูนย์กลางสำหรับใช้เดสก์ท็อปโดยรวม.

การใช้งาน `nautilus` อาจแบ่งออกได้เป็น 3 ประเภทได้แก่

ไฟล์เบราเซอร์ (file browser)

การใช้ `nautilus` แบบไฟล์เบราเซอร์ให้เรียกใช้โปรแกรมจากเมนู Applications → Browse Filesystem หรือสั่งคำสั่ง “`nautilus --browser`” จากบรรทัดคำสั่ง. หน้าต่าง `nautilus` แบบนี้ทางเทคนิคเรียกว่า *Navigation Metaphor* คือการมองไฟล์เบราเซอร์เป็นโปรแกรม, เมื่อเปิดไดเรกทอรีไม่มีการเปิดหน้าจอใหม่. คล้ายกับการใช้งานจาก

เชลล์ที่ผู้ใช้ต้องรู้ว่าขณะนี้กำลังอยู่ในไดเรกทอรีอะไร. การใช้งานแบบนี้เหมาะสำหรับผู้ที่คุ้นเคยกับการใช้ระบบปฏิบัติการวินโดวส์มาก่อน.



รูปที่ 6.36: ใช้ `nautilus` เป็นไฟล์เบราเซอร์.

ในหน้าต่างของ `nautilus` แบบเบราเซอร์จะมีพื้นที่กรอกชื่อสถานที่ “Location:” ซึ่งผู้ใช้สามารถใส่ชื่อไดเรกทอรีที่ต้องการจัดการไฟล์หรือที่อยู่พิเศษในตารางที่ 6.4.

ไฟล์เบราเซอร์เชิงวัตถุ (object)

บนพื้นหน้าจอเดสก์ท็อปจะมีไอคอน (icon) อำนวยความสะดวกเช่น “Computer”, “Home”, “Start Here” และ “Trash”. ถ้าดับเบิ้ลคลิ้กไอคอนเหล่านี้จะเป็นการเรียกใช้ `nautilus` เป็นไฟล์เบราเซอร์เชิงวัตถุ.

เวลาดับเบิ้ลคลิ้กไดเรกทอรีที่แสดงในหน้าต่างจะเปิดหน้าต่างใหม่ทุกครั้ง. การเปิดหน้าต่างใหม่ทุกครั้งหลังจากที่ดับเบิ้ลคลิ้กตัววัตถุนี้เริ่มใช้ใน GNOME 2.6 เรียกว่า *spatial view* หรือในทางเทคนิคเรียกว่า *Object Oriented Metaphor*. หน้าต่างของ `nautilus` จะไม่แสดงไอออนบาร์ (icons bar) หรือ Location โดยจะถือว่าไดเรกทอรีที่ดับเบิ้ลคลิ้กนั้นเป็นวัตถุ, เป็นแฟ้มที่ใส่ไฟล์อื่น ๆ. ถ้าดับเบิ้ลคลิ้กไฟล์, `nautilus` จะดูว่าไฟล์นั้นเป็นไฟล์อะไรแล้วเรียกโปรแกรมที่เหมาะสมเปิดไฟล์นั้น. ในทำนองเดียวกันถ้าสิ่งที่ดับเบิ้ลคลิ้กคือไดเรกทอรี, ก็จะเปิดหน้าต่างแอพพลิเคชันที่ใช้ดูไฟล์ที่อยู่ข้างในเสมอเหมือนกับการเปิดไฟล์.

การกระทำของ `nautilus` แบบนี้ดูเหมือนกับย้อนยุคและไม่สะดวกสำหรับผู้ที่ใช้คอมพิวเตอร์คล่องแล้ว. แต่ในมุมมองของผู้ที่ไม่เคยใช้คอมพิวเตอร์มาก่อนจะเข้าใจกับการทำงานในลักษณะนี้ได้ง่ายกว่าการทำงานแบบไฟล์เบราเซอร์. หลังจากที่เบิ้ลคลิ้กแล้ว, หน้าต่างเดิมจะไม่มีการเปลี่ยนแปลง, ไม่สูญหาย, ทำให้ผู้ที่ไม่เคยใช้คอมพิวเตอร์มาก่อนไม่หลงว่าอะไรอยู่ที่ไหน. ถ้าไม่ต้องการเปิดหน้าต่างใหม่ทุกครั้ง, ผู้ใช้สามารถดับเบิ้ลคลิ้กด้วยปุ่มกลางแทนปุ่มซ้ายของเมาส์. หรือดับเบิ้ลคลิ้กตามปรกติขณะที่กดคีย์ `Shift` ค้างไว้. ถ้าต้องการเปลี่ยนเป็นแบบไฟล์เบราเซอร์ก็สามารถคลิ้กปุ่มเมาส์ขวาแล้วเลือก Browse Folder.

ตารางที่ 6.4: ชื่อสถานที่พิเศษสำหรับ nautilus.

สถานที่	คำอธิบาย
applications:///	แสดงรายการแอพพลิเคชันในเมนู Applications.
burn:///	แสดงสถานที่เตรียมไฟล์และไดเรกทอรีสำหรับเขียนแผ่นซีดี.
fonts:///	แสดงฟอนต์ที่มีอยู่ในระบบ. ผู้ใช้สามารถติดตั้งฟอนต์ได้โดยการลากฟอนต์เข้าไปในหน้าต่างนี้.
network:///	แสดงเน็ตเวิร์กเช่นไซต์สำหรับ ftp ที่ติดตั้งไว้.
preferences:///	แสดงแอพพลิเคชันสำหรับช่วยปรับแต่งสภาพแวดล้อมเดสก์ท็อปต่าง ๆ. ผู้ใช้สามารถใช้โปรแกรมช่วยปรับแต่งสภาพแวดล้อมเดสก์ท็อปเหล่านี้ได้จากเมนูเช่นกัน.
server-settings:///	แสดงแอพพลิเคชันสำหรับปรับแต่งเซิร์ฟเวอร์.
start-here:///	อำนวยความสะดวกเป็นจุดเริ่มต้นสำหรับการกระทำต่าง ๆ. ในหน้าต่างจะแสดงสถานที่ของแอพพลิเคชัน, ที่ปรับแต่งระบบ ฯลฯ.
system-settings:///	แสดงแอพพลิเคชันำหรับปรับแต่งระบบ.
themes:///	แสดงธีม (theme) ของ GNOME และเป็นที่ผู้ใช้สามารถติดตั้งธีมใหม่ได้ด้วย.

เดสก์ท็อป

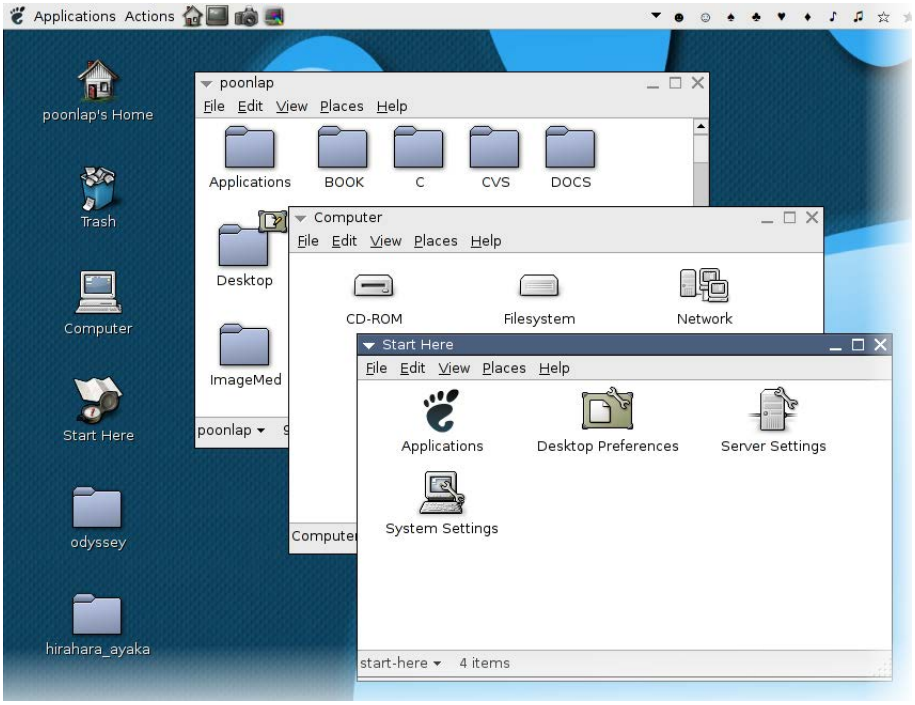
พื้นหลังเดสก์ท็อปของ GNOME จริง ๆแล้วเป็นหน้าต่างหนึ่งของ nautilus ซึ่งรันโดยอัตโนมัติหลังจากที่ผู้ใช้เข้าสู่เซสชันของ GNOME.

6.12.6 เซสชัน

เวลาผู้ใช้ล็อกอินผ่านทางดิสเพลยแมนเนเจอร์เข้าสู่ระบบ X วินโดว์จนจบการทำงานจะเรียกช่วงสภาพแวดล้อมนั้นว่า*เซสชัน (session)*. การจัดการเซสชันเป็นการเปิดโอกาสให้ผู้ใช้สามารถเก็บบันทึกการทำงานในเดสก์ท็อปเพื่อใช้ในคราวหน้าตอนล็อกอินได้และเป็นวิธีที่ผู้ใช้เรียกใช้โปรแกรมต่าง ๆโดยอัตโนมัติหลังจากที่ล็อกอิน.

คำแปลภาษาไทยของ session คือ วาระแต่ในที่นี้จะใช้คำว่าเซสชันเพื่อความสะดวก.

โปรแกรมที่เป็นตัวเริ่มต้นเซสชันในสภาพแวดล้อมเดสก์ท็อป GNOME ได้แก่ gnome-session ซึ่งปรกติจะถูกกระทำการโดยดิสเพลยแมนเนเจอร์เช่น gdm หรือหลังจากสั่งคำสั่ง startx. gnome-session จะเริ่มทำงานโดยสร้างโปรเซสที่สำคัญ ๆสำหรับสภาพแวดล้อมเดสก์ท็อปเช่น gnome-wm, gnome-panel, nautilus ฯลฯ. เวลาเริ่มเซสชันจะมีไฟล์ /usr/share/gnome/default.session เป็นไฟล์ตั้งค่าเริ่มต้นของเซสชันปริยาย. ในไฟล์นี้จะระบุโปรแกรมต่าง ๆที่จำเป็นสำหรับสภาพแวดล้อมเดสก์ท็อป.



รูปที่ 6.37: ใช้ nautilus เป็นไฟล์เบราเซอร์เชิงวัตถุและพื้นเดสก์ท็อป.

ตัวอย่างที่ 6.56: ไฟล์ /usr/share/gnome/default.session.

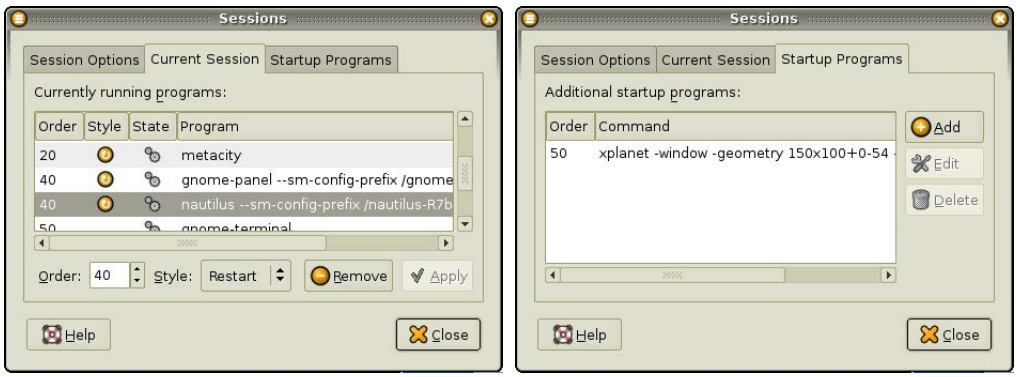
```
# This is the default session that is launched if the user doesn't
# already have a session.
# The RestartCommand specifies the command to run from the $PATH.
# The Priority determines the order in which the commands are started
# (with Priority = 0 first) and defaults to 50.
# The id provides a name that is unique within this file and passed to the
# app as the client id which it must use to register with gnome-session.
# The clients must be numbered from 0 to the value of num_clients - 1.

[Default]                                                    ← ชื่อเซสชัน
num_clients=8                                          ← จำนวนไคลเอ็นต์ในเซสชัน
0,id=default0                                                ← id ของไคลเอ็นต์
0,Priority=0                                               ← ค่าลำดับความสำคัญ
0,RestartCommand=gnome-smproxy --sm-client-id default0     ← โปรแกรมที่ต้องการรัน
1,id=default1
1,Priority=10
1,RestartCommand=gnome-wm --sm-client-id default1
2,id=default2
2,Priority=40
2,RestartCommand=gnome-panel --sm-client-id default2
3,id=default3
3,Priority=40
3,RestartCommand=nautilus --no-default-window --sm-client-id default3
4,id=default4
4,Priority=60
4,RestartCommand=gnome-cups-icon --sm-client-id default4
```

```
5,id=default5
5,Priority=40
5,RestartCommand=gnome-volume-manager --sm-client-id default5
6,id=default6
6,Priority=40
6,RestartCommand=magicdev --sm-client-id default6
7,id=default7
7,Priority=50
7,RestartCommand=vino-session --sm-client-id default7
```

จากตัวอย่างเป็นไฟล์ตั้งค่าเริ่มต้นเซสชันปริยายของ GNOME 2.8. ชื่อที่อยู่ในวงเล็บเหลี่ยมคือชื่อเซสชันซึ่งในที่นี้คือ Default. โปรแกรมที่ทำงานในเซสชันจะมีชื่อ id และลำดับความสำคัญ. โปรแกรมต่าง ๆที่ระบุในไฟล์จะถูกกระทำตามลำดับความสำคัญที่ระบุไว้. โปรแกรมที่มีค่า priority น้อยกว่าจะทำงานก่อนตามลำดับ. RestartCommand เป็นการระบุให้เริ่มทำงานใหม่ถ้าตายไป. สมมติว่าเราสั่งคำสั่ง "`killall nautilus`" เพื่อฆ่าโปรเซสของ `nautilus` ทุกตัว, ตัวจัดการเซสชันจะรันโปรแกรม `nautilus` ใหม่แทนตัวที่ตายไปโดยอัตโนมัติ.

ไฟล์ `default.session` เป็นไฟล์ที่ระบบเตรียมไว้ให้อยู่แล้ว, ผู้ใช้ไม่ต้องสร้างเอง. และผู้ใช้สามารถปรับแต่ง, ควบคุมโปรแกรมในเซสชันเฉพาะของตัวเองด้วยโปรแกรม `gnome-session-properties`. โปรแกรมนี้สามารถเลือกจากเมนู Applications → Desktop Preferences → Advanced → Sessions.



รูปที่ 6.38: โปรแกรม `gnome-session-properties` สำหรับปรับแต่งเซสชัน.

ผู้ใช้เอาโปรเซสที่กำหนดในเซสชันปริยายเช่น `nautilus` ออกได้ถ้าไม่ต้องการโดยใช้แท็บ Current Session ในรูปที่ 6.38. ตอนล็อกเอาท์ออกจากระบบจะมีคำถามถามว่าต้องการบันทึกเซสชันไว้ใช้คราวหน้าหรือไม่ให้ตอบว่าใช่. หรือถ้าต้องการบันทึกเซสชันในขณะใดขณะหนึ่งก็สามารถสั่งคำสั่ง `gnome-session-save` ได้ทันที. เซสชันที่บันทึกจะเป็นข้อมูลที่เก็บไว้ในไฟล์ `~/.gnome2/session` ซึ่งมีเนื้อหาคล้ายกับไฟล์ `default.session`. ถ้าผู้ใช้มีไฟล์ `~/.gnome2/session` อยู่, เวลาล็อกอินเข้าสู่สภาพแวดล้อมเดสก์ท็อป GNOME จะใช้ไฟล์นี้เริ่มเซสชันแทนไฟล์ `defatul.session`.

นอกจากการปรับแต่งเซสชันแล้ว, ผู้ใช้สามารถกำหนดโปรแกรมที่ต้องการรันหลังจากเข้าเซสชัน GNOME ทางแท็บ Startup Program ของ `gnome-session-properties` ด้วย.

6.12.7 การปรับแต่งสภาพแวดล้อมเดสก์ท็อป GNOME

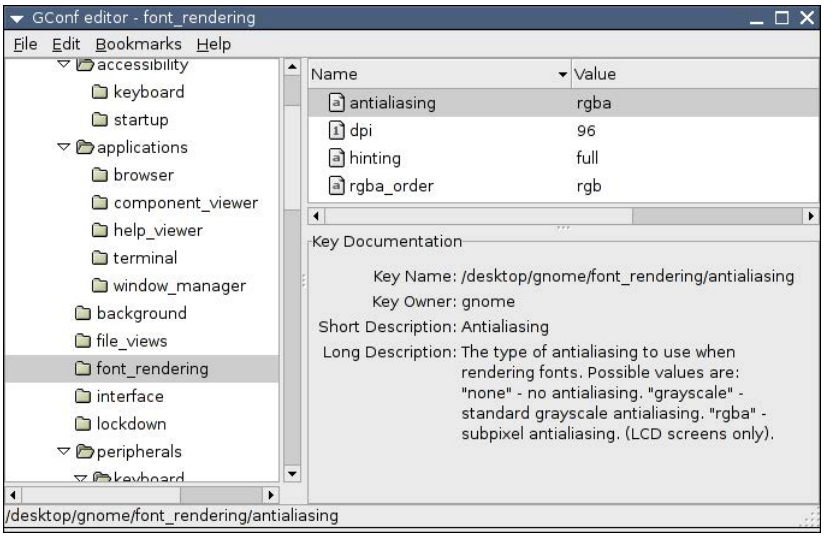
การปรับแต่งสภาพแวดล้อมเดสก์ท็อป GNOME โดยทั่วไปจะทำโดยการเลือกโปรแกรมปรับแต่งจากเมนู Applications → Desktop Preferences. สิ่งที่ผู้ใช้สามารถปรับแต่งได้เช่น ฟอนต์ที่ใช้ในเดสก์ท็อป, เซสชัน, เมาส์ ฯลฯ. โปรแกรมช่วยปรับแต่งเหล่านี้มักมีชื่อเริ่มต้นด้วย `gnome-` และลงท้ายด้วย `-properties` เช่นโปรแกรมสำหรับปรับแต่งความละเอียดหน้าจอสามารถเรียกใช้จากบรรทัดคำสั่งด้วยชื่อ `gnome-display-properties` เป็นต้น.

โปรแกรมช่วยปรับแต่งเดสก์ท็อปเหล่านี้จริง ๆแล้วเป็นอินเทอร์เฟสระหว่างผู้ใช้กับเดมอน `gconfd-2` ซึ่งทำหน้าที่จัดการปรับแต่งระบบ, บันทึกการปรับแต่งต่าง ๆเก็บลงไฟล์ในฟอร์แมต XML. ไฟล์ที่เก็บค่าการปรับแต่งเหล่านี้จะอยู่ในไดเรกทอรี `~/.gconf` ซึ่งในไดเรกทอรีนี้จะมีไดเรกทอรีแยกย่อยต่อออกไปสำหรับเก็บค่าปรับแต่งเฉพาะบุคคลของเดสก์ท็อปและแอพพลิเคชันต่าง ๆ. วิธีการปรับแต่งเดสก์ท็อปและแอพพลิเคชันในสภาพแวดล้อมเดสก์ท็อป GNOME มีชื่อเรียกว่า *GConf*.

ไฟล์โปรแกรม `gconfd-2` จะอยู่ในไดเรกทอรี `/usr/lib/gconf2`.

โดยปรกติผู้ใช้สามารถปรับแต่งเดสก์ท็อปจากเมนู Desktop Preferences ด้วยโปรแกรมช่วยปรับแต่งเดสก์ท็อป. ถ้าเป็นการปรับแต่งแอพพลิเคชัน, ผู้ใช้สามารถปรับแต่งแอพพลิเคชันได้จากเมนูในหน้าต่างของแอพพลิเคชันนั้น ๆ. ในกรณีที่สิ่งที่ต้องการปรับแต่งไม่สามารถทำได้จากโปรแกรมช่วยปรับแต่งหรือจากเมนูที่เตรียมไว้, ผู้ใช้สามารถใช้โปรแกรม `gconf-editor` (จากเมนูเลือก System Tools → Configuration Editor) ใช้ปรับแต่งค่าคุณสมบัติต่างได้โดยตรงโดยที่ไม่ต้องแก้ไขไฟล์ที่อยู่ใต้ไดเรกทอรี `.gconf` ด้วยตัวเอง.

การปรับแต่งเดสก์ท็อปและแอพพลิเคชันด้วย GConf จะเก็บค่าคุณสมบัติต่าง ๆไว้ในฐานข้อมูลเป็นไฟล์ XML. ในเชิงตรรกะจะแจกแจงเก็บค่าคุณสมบัติที่ต้องการปรับแต่งเป็นโค้งสร้างคล้ายเหมือนกับไดเรกทอรีและไฟล์ที่ใช้ในระบบไฟล์ทั่วไป. ค่าที่เก็บในฐานข้อมูลจะมีชื่อที่เรียกว่าคีย์ (key) และค่า (value) ของคุณสมบัติที่ต้องการปรับแต่งควบคู่กันไป. ตัวอย่างเช่น `/apps/gnome-terminal/profiles/Default/font` เป็นชื่อคีย์สำหรับปรับแต่งฟอนต์ที่ใช้แสดงผลของโปรแกรม `gnome-terminal`. เวลาอ้างอิงชื่อคีย์จะระบุเป็นชื่อเต็ม ๆรวมถึงไดเรกทอรีตำแหน่งที่เก็บคีย์ซึ่งได้แก่ `/apps/gnome-terminal/profiles/Default` ด้วย. ในไดเรกทอรีหนึ่งสามารถเก็บคีย์ได้หลายตัวเช่นในไดเรอทอรี `/apps/gnome-terminal/profiles/Default` ยังมีคีย์อื่น ๆอีกเช่น `use_theme_colors`, `palette`, `exit_action` ฯลฯ. เราสามารถสำรวจโครงสร้างของไดเรกทอรีและคีย์ต่าง ๆหรือตั้งค่าได้จากโปรแกรม `gconf-editor`. ตัวโปรแกรมยังช่วยแสดงคำอธิบายคีย์ต่าง ๆและบางครั้งจะแสดงค่าต่างที่สามารถกำหนดให้ด้วย. ใน GNOME รุ่น 2.8, ผู้ดูแลระบบสามารถใช้ `gconf-editor` กำหนดค่าปริยาย (default) หรือค่าบังคับ (mandatory) ที่จะใช้ในสภาพแวดล้อมเดสก์ท็อปสำหรับผู้



รูปที่ 6.39: โปรแกรม gconf-editor ปรับแต่งเดสก์ท็อปและแอพพลิเคชัน.

ใช้ทั่วไปได้ด้วย.

วิธีปรับแต่งเดสก์ท็อปอีกวิธีหนึ่งคือใช้โปรแกรมบรรทัดคำสั่ง gconftool-2. เมื่อเทียบกับโปรแกรม gconf-editor แล้วจะใช้ยากกว่าและเหมาะสำหรับใช้ในเชลล์สคริปต์. เช่นจากตัวอย่างที่ 6.57, ผู้ใช้สามารถสร้างเชลล์สคริปต์สำหรับเปลี่ยนรูปพื้นเดสก์ท็อปทุก 5 นาทีจากอัลบันรูปที่มีอยู่ได้ด้วยคำสั่ง gconftool-2.

ตัวอย่างที่ 6.57: การใช้ gconftool-2

```
$ gconftool-2 --list-dirs /↵                    ← สำรวจไดเรกทอรีในฐานข้อมูล GConf
 /desktop
 /apps
 /system
 /schemas
$ gconftool-2 --list-entries /desktop/gnome/background↵          ← สำรวจคีย์และค่า
 color_shading_type = vertical-gradient
 secondary_color = #7F7F7F
 primary_color = #7F7F7F
 picture_filename = /usr/share/pixmaps/backgrounds/gnome/branded/GNOME-Aqua.jpg
 picture_options = stretched
 picture_opacity = 100
 draw_background = true
$ gconftool-2 --type string -s /desktop/gnome/background/picture_filename \ ↵
> /home/poonlap/wallpaper.jpg↵                          ← เปลี่ยนรูปของพื้นเดสก์ท็อป
$ gconftool-2  -g /desktop/gnome/background/picture_filename↵        ← ดูค่าของคีย์
/home/poonlap/wallpaper.jpg
```

เป็นที่ยอมรับกันว่าสภาพแวดล้อมเดสก์ท็อป GNOME ใช้ง่ายกว่าเชลล์. ผู้ใช้ที่ไม่คุ้นเคยกับอินเทอร์เฟสแบบบรรทัดคำสั่งสามารถใช้คอมพิวเตอร์ได้ง่ายขึ้นเพียงแต่มีทักษะเบื้องต้นในการใช้เมาส์และแป้นพิมพ์เท่านั้น. สำหรับผู้ที่ต้องเรียนรู้เกี่ยวกับสภาพแวดล้อมเดสก์ท็อป GNOME เพิ่มเติมสามารถอ่านเอกสารที่มาพร้อมกับ GNOME ได้จาก

โปรแกรม `yelp` (จากเมนู Applications → Help). สำหรับสภาพแวดล้อมเดสก์ท็อปอื่น ๆเช่น KDE ที่ไม่ได้แนะนำให้หนังสือเล่มนี้ผู้ใช้สามารถอ่านการใช้เบื้องต้นจากโปรแกรม `khelpcenter` และสิ่งที่สำคัญที่สุดคือการลองใช้ด้วยตนเอง.

6.13 สรุปท้ายบท

- ระบบ X วินโดว์เป็นระบบแบบไคลเอ็นต์เซิร์ฟเวอร์. เซิร์ฟเวอร์เป็นโปรแกรมแยกจากระบบปฏิบัติการมีหน้าที่แสดงผลกราฟิกทางหน้าจอเมื่อได้รับการร้องของจากไคลเอ็นต์.
- โปรแกรม X เซิร์ฟเวอร์สามารถทำงานได้หลายตัวพร้อมกันโดยระบุใช้หน้าจอคนละตัว. และประเภทของเซิร์ฟเวอร์มีหลายชนิดเช่น `Xnest`, `Xvfb`, `Xvnc` เป็นต้น.
- X เซิร์ฟเวอร์เปิดโอกาสให้ไคลเอ็นต์ติดต่อกับเซิฟร์เวอร์โดยการใช้ไลบรารี Xlib ซึ่งเป็น API สำหรับแสดงผลกราฟิกขั้นพื้นฐาน. ในความเป็นจริงไคลเอ็นต์จะใช้ไลบรารีอื่น ๆที่ใช้ไลบรารี Xlib อีกทีในการสร้างโปรแกรมเช่น GTK+, Qt เป็นต้น.
- การจัดการหน้าต่างของโปรแกรมเช่นการย่อขยาย, ย้ายตำแหน่งหน้าต่างเป็นหน้าที่ของโปรแกรมที่เรียกว่าวินโดว์แมนเนเจอร์.
- ระบบการจัดการฟอนต์แบ่งเป็น 2 ประเภทใหญ่ ๆคือระบบจัดการฟอนต์แบบดั้งเดิมที่เรียกว่า X core font และ fontconfig. เวลาติดตั้งฟอนต์ใหม่ในระบบควรคำนึงถึงปัจจัยต่าง ๆได้แก่ ระบบการจัดการฟอนต์ที่ใช้, ต้องการติดตั้งฟอนต์ในระบบโดยรวมหรือเฉพาะบุคคล เป็นต้น.
- การปรับแต่งแป้นพิมพ์แบ่งเป็น 2 ประเภทใหญ่ ๆคือใช้โปรแกรม `xmodmap` และ XKB. XKB เหมาะสำหรับการปรับแต่งแป้นพิมพ์ภาษาไทยและในสภาพแวดล้อมเดสก์ท็อปเช่น GNOME สามารถปรับแต่งแป้นพิมพ์เฉพาะบุคคลได้.
- สภาพแวดล้อมเดสก์ท็อปช่วยอำนวยความสะดวกให้ผู้ที่เริ่มใช้คอมพิวเตอร์ใช้คอมพิวเตอร์ได้ง่ายขึ้น. โปรแกรมต่าง ๆที่เป็นส่วนหนึ่งในสภาพแวดล้อมเดสก์ท็อปจะมีรูปร่างหน้าตา, วิธีการใช้งานคล้ายเหมือนกัน, มีเอกสารช่วยเหลือผู้ใช้, ใช้เมาส์ควบคุมการทำงานเป็นหลัก เป็นต้น.

บทที่ 7
ภาษาไทยกับลินุกซ์

ในอดีต, คอมพิวเตอร์สร้างขึ้นมาสำหรับการใช้งานในกลุ่มประเทศที่ใช้ภาษาอังกฤษ. ปัจจุบันคอมพิวเตอร์ใช้กันอย่างแพร่หลายทั่วโลกไม่จำกัดอยู่ในประเทศที่ใช้ภาษาอังกฤษอีกต่อไป. ในแต่ละประเทศมีเอกลักษณ์ของตัวเอง, บางประเทศมีภาษาของตัวเอง, บางประเทศใช้ภาษาเดียวกันประเทศอื่นแต่มีการใช้ภาษามีความแตกต่างกันในรายละเอียดปลีกย่อย. และสิ่งเหล่านี้เองเป็นปัจจัยที่ทำให้โปรแกรมที่ใช้ในคอมพิวเตอร์ต้องเปลี่ยนจากการสนับสนุนภาษาอังกฤษอย่างเดียวมาสนับสนุนภาษาต่าง ๆ.

วิธีการทำให้โปรแกรมสนับสนุนภาษาอื่นนอกจากภาษาอังกฤษที่เข้าใจง่ายที่สุดคือการแก้ไขไฟล์ต้นฉบับของโปรแกรมเช่นถ้าเป็นโปรแกรมแบบ GUI ก็อาจจะเปลี่ยนข้อความที่ใช้แสดงในเมนูเป็นภาษาท้องถิ่นหรือปรับรหัสต้นฉบับส่วนอื่น ๆให้เหมาะสมกับภาษาที่ต้องการใช้. การกระทำในลักษณะนี้เรียกว่าการ *localization* หรือเรียกย่อ ๆว่า *L10N* เป็นหลักการที่เข้าใจง่ายแต่มีข้อเสียบางอย่างเช่นถ้าต้องการทำให้โปรแกรมรองรับภาษาไทย 10 ภาษาก็จะได้โปรแกรมที่ปรับแก้ไขแล้ว 10 โปรแกรมเป็นต้น.

เลข 10 ที่อยู่ใน L10N คือจำนวนอักษรที่อยู่ระหว่าง l กับ n ในคำว่า localization.

กลวิธีอีกอย่างที่สามารถทำให้โปรแกรมรองรับภาษานานาชาติได้โดยส่งผลกระทบรหัสต้นฉบับเดิมให้น้อยที่สุดคือการ *Internationalization* หรือเขียนย่อ ๆว่า *I18N*. แนวความคิดของ I18N จะต่างจาก L10N ที่ว่าจะแยกส่วนที่จัดการหรือเกี่ยวข้องภาษาท้องถิ่นเช่นข้อความที่ต้องแสดง ฯลฯ ออกจากตัวรหัสต้นฉบับ. แต่รหัสต้นฉบับนั้นต้องเขียนอยู่ในมาตรฐานที่กำหนดที่สามารถรองรับภาษานานาชาติได้. วิธีนี้เป็นวิธีที่สะอาดกว่าการ L10N สามารถแยกงานของผู้พัฒนาซอฟต์แวร์กับผู้แปลภาษาออกจากกันได้ง่าย.

เลข 18 ที่อยู่ใน I18N หมายถึงจำนวนอักษรที่อยู่ระหว่าง I และ N.

การรองรับภาษาไทยไม่จำกัดเฉพาะข้อมความที่ใช้แสดงในโปรแกรมเท่านั้น. การประมวลผลของโปรแกรมที่เกี่ยวกับข้องกับภาษายังมีอีกหลายส่วนเช่น

- การเข้ารหัสอักขระ (character encoding) คือการนำเสนอข้อมูลอักขระเป็นสายข้อมูลในหน่วยไบต์. ในแต่ละภาษาใช้จำนวนบิตแสดงข้อมูลต่างกัน. ข้อมูล 7 บิต สามารถแทนอักขระทั่วไปที่ใช้ในภาษาอังกฤษได้ทุกตัวในขณะที่ภาษาไทยต้องใช้ข้อมูล 8 บิต (1 ไบต์) เพื่อที่จะแสดงทั้งข้อมูลภาษาอังกฤษและภาษาไทย.

คำว่า encryption ก็แปลงเป็นไทยว่าการเข้ารหัสเช่นกัน.

- การแสดงวันเดือนปี. โปรแกรมที่รองรับ I18N จะแสดงวันเดือนปีให้เหมาะสมกับสภาพแวดล้อมของผู้ใช้เช่นสำหรับคนไทยจะคุ้นเคยกับปี พ.ศ. มากกว่าปี ค.ศ.

เป็นต้น.

- การแสดงตัวเลขและสกุลเงินตรา. ในบางประเทศจะมีวิธีการแสดงจำนวนจุดทศนิยมไม่เหมือนกันเช่นในประเทศฝรั่งเศสจะนิยมเขียนจุดทศนิยมเป็นเครื่องหมายลูกน้ำ.

- การรองรับภาษานานาชาติโดยตัวโปรแกรมอย่างเดียวบางครั้งไม่เพียงพอ. ในสภาพแวดล้อมที่รองรับภาษานานาชาติต้องมีฟอนต์เพื่อแสดงผลอักขระต่าง ๆให้ถูกต้องด้วย.

- สภาพแวดล้อมที่เปิดให้ป้อนข้อมูลได้หลายภาษาจะสามารถตั้งผังแป้นพิมพ์ได้หลายภาษาพร้อม ๆกัน.

7.1 ความรู้เบื้องต้นเกี่ยวกับอักขระไทย

ก่อนที่จะแนะนำเรื่องเกี่ยวกับการประมวลผลข้อมูลภาษาไทย, เราจะมาทำความเข้าใจกับอักขระไทยและศัพท์เทคนิคที่เกี่ยวข้องก่อนดังนี้.

- อักขระ (character) —— สัญลักษณ์ที่มีความหมายในตัวเช่น “a” คืออักขระที่แสดงถึงตัวอักษร “a”. อักขระมีความหมายคลอบคลุมถึงตัวอักษร, ตัวเลข, เครื่องหมายพิเศษ และเครื่องหมายอื่น ๆ.

- รหัส (code) —— ค่าตัวเลขที่ใช้แทนอักขระ.

- ชุดอักขระ (character set) —— กลุ่มของอักขระ. โดยทั่วไปจะถือว่ามีความหมายเดียวกันกับชุดรหัสอักขระ.

- ชุดรหัสอักขระ (coded character set) —— บางครั้งเรียกย่อ ๆว่า charset, code set หมายถึงชุดอักขระที่อักขระทุกตัวมีรหัสประจำตัวเช่น “a” มีค่ารหัสเป็น 0x61 (เลขฐานสิบหก) ฯลฯ.

- การเข้ารหัสอักขระ (character encoding) —— วิธีการ, รูปแบบการนำเสนอข้อมูลรหัสอักขระเป็นสายข้อมูลไบต์.

- ยูนิโค้ด (unicode) —— ชุดรหัสอักขระที่รวมอักขระของภาษานานาชาติไว้ในชุดเดียวและใช้ค่าหมายเลขขนาด 16 บิตในการนำเสนอ.

- กลีฟ (glyph) —— รูปอักขระที่ใช้ในการแสดงผล.

7.1.1 รหัสอักขระ TIS-620

อักขระภาษาไทยที่ใช้ในคอมพิวเตอร์นิยามไว้ใน มอก.620-2533 (TIS-620) [49] โดยสำนักงานมาตรฐานผลิตภัณฑ์อุตสาหกรรม (สมอ.) มีอยู่ 87 ตัวประกอบด้วย

- พยัญชนะไทย (46 ตัว) — ก ข ฃ ค ฅ ฆ ง จ ฉ ช ซ ฌ ญ ฎ ฏ ฐ ฑ ฒ ณ ด ต ถ ท ธ น บ ป ผ ฝ พ ฟ ภ ม ย ร ล ว ศ ษ ส ห ฬ อ ฮ
- สระ (19 ตัว) — ฤ ฦ ะ ั า ำ ิ ี ึ ื ุ ู ฺ (พินทุ) เ แ โ ใ ไ ๋ (นิคหิต)
- วรรณยุกต์และทัณฑฆาต (5 ตัว) — ่ ้ ๊ ๋ ์ (ทัณฑฆาต)
- ตัวเลขไทย (10 ตัว) — ๐ ๑ ๒ ๓ ๔ ๕ ๖ ๗ ๘ ๙
- เครื่องหมายพิเศษ (7 ตัว) — ฯ ฿ ๆ ็ (ยามักการ) ๏ (ฟองมัน) ๚ (อังคั่นคู่) ๛ (โคมูตร)

เราเรียกชุดรหัสอักขระนี้ว่า TIS-620 (รูปที่ 7.1) ประกาศใช้สาธารณะเมื่อปีพ.ศ. 2529 และได้รับการปรับปรุงอีกทีเมื่อปีพ.ศ. 2533.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	`	p				ฐ	ภ	ะ	เ	๐
1			!	1	A	Q	a	q			ก	ฑ	ม	ั	แ	๑
2			"	2	B	R	b	r			ข	ฒ	ย	า	โ	๒
3			#	3	C	S	c	s			ฃ	ณ	ร	ำ	ใ	๓
4			$	4	D	T	d	t			ค	ด	ฤ	ิ	ไ	๔
5			%	5	E	U	e	u			ฅ	ต	ล	ี	ๅ	๕
6			&	6	F	V	f	v			ฆ	ถ	ฦ	ึ	ๆ	๖
7			'	7	G	W	g	w			ง	ท	ว	ื	็	๗
8			(	8	H	X	h	x			จ	ธ	ศ	ุ	่	๘
9			)	9	I	Y	i	y			ฉ	น	ษ	ู	้	๙
A			*		J	Z	j	z			ช	บ	ส	ฺ	๊	๚
B			+	;	K	[	k	{			ซ	ป	ห		๋	๛
C			,	<	L	\	l	|			ฌ	ผ	ฬ		์	
D			-	=	M	]	m	}			ญ	ฝ	อ		๎	
E			.	>	N	^	n	~			ฎ	พ	ฮ		๏	
F			/	?	O	_	o				ฏ	ฟ	ฯ	฿	๐	

รูปที่ 7.1: ตารางรหัสอักขระ TIS-620.

วิธีการดูตารางรหัสอักขระจะเริ่มจากการอ่านตัวเลขที่อยู่ข้างบนตารางก่อนและตามด้วยตัวเลขที่อยู่ด้านซ้ายของตาราง. ตัวเลขเหล่านี้เป็นตัวเลขฐาน 16 มีค่าตั้งแต่ 0 ถึง F. ตัวอย่างเช่น “a” มีค่ารหัสเป็น 0x61 หรือแปลงเป็นเลขฐานสองเป็น 1100001. อักขระ

ASCII (ภาษาอังกฤษ) จะอยู่ในช่วง 0x00 - 0x7F (0000000 - 1111111) และอักขระภาษาไทยจะอยู่ในช่วง 0xA1 - 0xFB (10100001 - 11111011). จะเห็นได้ว่าในระบบที่รองรับภาษาอังกฤษอย่างเดียวสามารถนำเสนอข้อมูลได้โดยใช้ข้อมูลอักขระขนาด 7 บิต. ถ้าต้องการเพิ่มการนำเสนอข้อมูลภาษาไทยด้วยต้องใช้ข้อมูลอักขระขนาด 8 บิตหรือ 1 ไบต์.

เราสามารถแบ่งข้อมูลในตารางที่แสดงชุดรหัสอักขระแบบ 8 บิตเป็นส่วนต่าง ๆได้ดังนี้

- C0 —— ส่วนอักขระควบคุม (control characters) ที่ตำแหน่งบิตตัวที่ 8 (หลักตัวเลขที่อยู่ซ้ายสุด) มีค่าเป็นศูนย์, 0x00 (00000000) - 0x1F (00011111) และ 0x7F (01111111).
- G0 หรือ GL —— ได้แก่ชุดอักขระที่ตำแหน่งบิตตัวที่ 8 มีค่าเป็นศูนย์, 0x20 (00100000) - 0x7E (01111110). GL ย่อมาจากคำว่า *Graphics Left*.
- C1 —— ส่วนอักขระควบคุมที่ตำแหน่งบิตตัวที่ 8 มีค่าเป็นหนึ่ง, 0x80 (1000000) - 0x9F (10011111).
- G1 หรือ GR —— ได้แก่ชุดอักขระที่ตำแหน่งบิตตัวที่ 8 มีค่าเป็นหนึ่ง, 0xA0 (10100000) - 0xFF (11111111). GR ย่อมาจากคำว่า *Graphics Right*.

ISO 8859-1 เป็นชุดรหัสอักขระภาษาที่ใช้ในยุโรปตะวันตก. มีชื่อเรียกอีกอย่างว่า Latin1.

อักขระภาษาไทยจะอยู่ในตำแหน่ง GR ซึ่งชุดรหัสอักขระอื่น ๆเช่น ISO 8859-1, ISO 8859-2 ฯลฯ กำหนดรหัสของอักขระในภาษาของตนเองไว้ในช่วง GR เช่นกัน. คอมพิวเตอร์รับรู้ข้อมูลต่าง ๆเป็นไบนารีดังนั้นเวลาที่คอมพิวเตอร์รับข้อมูลเท็กซ์เช่น 0xA1, คอมพิวเตอร์ไม่สามารถตัดสินได้ว่า 0xA1 คืออักขระภาษาไทย “ก” หรือภาษาลาติน. วิธีแก้ปัญหานี้มีอยู่ 2 ทางคือบอกให้ระบบปฏิบัติรับรู้ว่าสภาพแวดล้อมภาษาที่ใช้อยู่, หรือเปลี่ยนไปใช้ชุดรหัสอักขระยูนิโค้ดซึ่งช่วงรหัสอักขระของภาษาไทยจะไม่ซ้ำกับ latin1.

7.1.2 ชุดรหัสอักขระ ISO 8859-11

ชุดรหัสอักขระ ISO 8859-11 เป็นชุดรหัสอักขระภาษาไทยที่ขยายต่อจากมาตรฐาน ISO 8859 ซึ่งเป็นมาตรฐานรหัสอักขระที่สามารถแสดงได้ด้วยข้อมูล 8 บิต (1 ไบต์) และนิยมใช้กันในหมู่ประเทศทางยุโรป. ค่ารหัสของชุดอักขระนี้เหมือนกับรหัสอักขระ TIS-620 ทุกประการและเป็นมาตรฐานที่องค์กรมาตรฐานโลก (ISO, International Organization for Standardization) ประกาศใช้เมื่อปี พ.ศ. 2544.

7.1.3 ชุดรหัสอักขระยูนิโค้ดและ ISO 10646

ชุดรหัสอักขระ*ยูนิโค้ด (Unicode, Universal chracter encoding)* เป็นชุดรหัสอักขระที่รวมอักขระของภาษาต่าง ๆเข้าไว้ด้วยกันเป็นชุดเดียว. รหัสที่มอบหมายให้อักขระพื้นฐานมีค่าตั้งแต่ 0x0000 ถึง 0xFFFD จะเรียกว่า *Basic Multilingual Plane (BMP)* ซึ่งมีภาษาหลัก ๆที่ใช้กันทั่วโลกรวมอยู่ในนี้. รหัสอักขระภาษาไทยจะอยู่ในช่วง 0x0E00 - 0x0E7F แต่ค่ารหัสที่ใช้จริงจะอยู่ในช่วง 0x0E01 - 0x0E5B ในรูปที่ 7.3.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	`	p				°	À	Ð	à	ð
1			!	1	A	Q	a	q			¡	±	Á	Ñ	á	ñ
2			"	2	B	R	b	r			¢	²	Â	Ò	â	ò
3			#	3	C	S	c	s			£	³	Ã	Ó	ã	ó
4			$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
5			%	5	E	U	e	u			¥	µ	Å	Õ	å	õ
6			&	6	F	V	f	v			¦	¶	Æ	Ö	æ	ö
7			'	7	G	W	g	w			§	·	Ç	×	ç	÷
8			(	8	H	X	h	x			¨	¸	È	Ø	è	ø
9			)	9	I	Y	i	y			©	¹	É	Ù	é	ù
A			*	:	J	Z	j	z			ª	º	Ê	Ú	ê	ú
B			+	;	K	[	k	{			«	»	Ë	Û	ë	û
C			,	<	L	\	l	|			¬	¼	Ì	Ü	ì	ü
D			-	=	M	]	m	}			-	½	Í	Ý	í	ý
E			.	>	N	^	n	~			®	¾	Î	Þ	î	þ
F			/	?	O	_	o				¯	¿	Ï	ß	ï	ÿ

รูปที่ 7.2: ตารางรหัสอักขระ ISO-8859-1 (Latin1).

ชุดอักขระที่รวมภาษาต่าง ๆเข้าด้วยกันอีกตัวคือ *ISO 10646* ซึ่งในปัจจุบันได้ทำการตกลงกับยูนิโค้ดรวมเนื้อหาเข้าด้วยกันเพื่อความไม่สับสนของผู้ใจแต่ยังใช้ชื่อแยกกันอยู่. ชุดรหัสอักขระยูนิโค้ดแบบ BMP จะใช้ข้อมูลขนาด 2 ไบต์ (16 บิต) แทนอักขระหนึ่งตัวซึ่งเพียงพอต่อภาษาหลัก ๆที่มีอยู่ในโลก. และสามารถขยายเพิ่มขึ้นอีกมีขนาดสูงสุด 4 ไบต์ (32 บิต) ซึ่งเป็นยูนิโค้ดชุดสมบูรณ์.

7.1.4 การเข้ารหัสอักขระ

คอมพิวเตอร์จะรับรู้การนำข้อมูลอักขระในรูปของสายข้อมูลไบต์ซึ่งเรียกว่า*การเข้ารหัสอักขระ (character encoding)*. วิธีการเข้ารหัสอักขระที่ง่ายที่สุดคือใช้รหัสที่กำหนดไว้ในชุดรหัสอักขระตรง ๆเช่นข้อมูลของอักขระ “ก” เมื่อเข้ารหัสอิงตามชุดรหัสอักขระ TIS-620 แล้วจะมีค่าเป็น 0xA1 เป็นต้น. การใช้รหัสแสดงข้อมูลแบบนี้มีผลเสียที่ว่าคอมพิวเตอร์ไม่สามารถรับรู้ประเภทของอักขระว่าเป็นภาษาไทยหรือภาษาลาติน. ถ้าเป็นการส่งข้อความผ่านทางเมลและไม่มีการบอกให้โปรแกรมอ่านเมลรู้ว่ารหัสที่ใช้เป็นภาษาไทย, ตัวโปรแกรมจะถือว่าเป็นรหัส ISO 8859-1 แทนที่จะเป็น TIS-620 เพราะรหัส ISO 8859-1 ใช้กันแพร่หลายมากกว่าภาษาไทยเป็นผลทำโปรแกรมอ่านเมลแสดงรูปทรงอักขระไม่ถูก

รูปที่ 7.3: ตารางรหัสอักขระยูนิโค้ดช่วงภาษาไทย.

ต้อง.

สำหรับชุดอักขระยูนิโค้ดแบบ BMP ถ้าจะใช้ค่ารหัสลงรหัสอักขระตรง ๆจะใช้พื้นที่ 2 ไบต์ต่อหนึ่งอักขระเช่น “ก” เมื่อเข้ารหัสอักขระแล้วมีค่าเป็น 0x0E01. วิธีการเข้ารหัสอักขระแบบนี้มีชื่อเรียกว่า *UCS-2*. ในทำนองเดียวกัน, ชุดอักขระยูนิโค้ดแบบสมบูรณ์จะใช้พื้นที่ 4 ไบต์ต่อหนึ่งอักขระในการเก็บข้อมูลเช่น “ก” เมื่อเข้ารหัสอักขระแล้วมีค่าเป็น 0x00000E01. วิธีการเข้ารหัสอักขระแบบนี้มีชื่อเรียกว่า *UCS-4*.

วิธีการเข้ารหัสอักขระแบบ UCS-2 และ UCS-4 มีข้อเสียคือใช้เนื้อที่เก็บข้อมูลเกินความจำเป็นทำให้มีการคิดค้นวิธีเข้ารหัสที่มีประสิทธิภาพยิ่งขึ้นโดยการแปลงค่ารหัสอักขระที่เรียกว่า *UTF (Unicode Transformation Format)*. วิธีเข้ารหัสอักขระ UTF มีวิธีหลายวิธีแยกย่อยไปอีกได้แก่ UTF-8, UTF-16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE และ UTF-32LE. วิธีเข้ารหัสอักขระที่นิยมใช้กันได้แก่ *UTF-8* ซึ่งมีข้อดีหลายประการและมีคุณสมบัติเด่น ๆคือ.

LE ย่อมาจาก Little Endian. BE ย่อมาจาก Big Endian

- ค่ารหัสที่เข้ารหัสแล้วของอักขระในช่วง 0x0000 - 0x007F จะเหมือนกับค่ารหัสอักขระ ASCII. ทำให้วิธีการประมวลผลข้อมูลภาษาอังกฤษไม่เปลี่ยนแปลงถึงแม้ข้อมูลที่ใช้จะเป็นยูนิโค้ด.

ตารางที่ 7.1: ค่ารหัสอักขระยูนิโค้ดช่วงต่าง ๆหลังจากเข้ารหัสแบบ UTF-8.

ช่วงรหัสยูนิโค้ด BMP	ค่าหลังจากที่เข้ารหัส UTF-8 (แสดงด้วยเลขฐานสอง)
0x0000 - 0x007F	0*xxxxxxx*
0x0080 - 0x07FF	110*xxxxx* 10*xxxxxx*
0x0800 - 0xFFFF	1110*xxxx* 10*xxxxxx* 10*xxxxxx*

- รหัสอักขระยูนิโค้ดตั้งแต่ 0x0080 จะถูกแปลงเป็นข้อมูลในหน่วย 1 ไบต์เรียงต่อกัน 2 หรือ 3 ตัวขึ้นอยู่กับช่วงของอักขระ. อักขระลาตินหนึ่งตัวจะถูกแปลงเป็นข้อมูล 2 ไบต์. ส่วนอักขระภาษาไทยหนึ่งตัวจะถูกแปลงเป็นข้อมูล 3 ไบต์. ตัวอย่างเช่น “ก” ซึ่งมีค่ารหัสยูนิโค้ด 0x0E01 เวลาเข้ารหัสแบบ UTF-8 จะเป็น 0xE0 0xB8 0x81 (11100000 10111000 10000001).
- ค่าที่เข้ารหัสแล้วมีกฎเกณฑ์ที่แน่นอนสามารถบอกจำนวนไบต์ที่ต้องแปลงกลับเป็นรหัสยูนิโค้ดได้.

ในที่นี้จะไม่อธิบายวิธีการเอ็นโค้ดแบบ UTF-8 แต่สามารถหาอ่านได้จาก RFC 2279 [50].

ปัจจุบันโปรแกรมที่ต้องรองรับภาษานานาชาติมักจะเก็บข้อมูลในรูป UTF-8 เช่นเว็บเพจ, ไฟล์ XML. ผลดีคือสามารถแสดงภาษาหลายภาษาได้ด้วยชุดอักขระตัวเดียว. สำหรับภาษาไทยมีข้อเสียเล็กน้อยที่ขนาดของข้อมูลที่เก็บจะเพิ่มขึ้น 3 เท่าเนื่องจากการเอ็นโค้ดแบบ UTF-8. และนี่เป็นเหตุผลหนึ่งที่คนทั่วไปยังใช้เอ็นโค้ดแบบ TIS-620 อยู่เพราะกินพื้นที่ต่อหนึ่งอักขระแค่ 1 ไบต์. อย่างไรก็ตาม, ยูนิโค้ดและการเข้ารหัสแบบ UTF-8 เป็นที่ยอมรับและนิยมใช้กันอย่างกว้างขวางในปัจจุบันและอาจเรียกได้ว่าเป็นมาตรฐานไปแล้ว.

7.1.5 การเปลี่ยนการเข้ารหัสด้วย `iconv`

การเข้ารหัสของภาษาไทยที่นิยมใช้กันมีอยู่สองแบบคือ TIS-620 และ UTF-8. ในกรณีที่ต้องการเปลี่ยนการเข้ารหัสของข้อมูลจะใช้คำสั่ง `iconv`.

เนื่องจากการเข้ารหัสแบบ ISO-8859-11 ให้ผลเหมือนกับ TIS-620 ในที่นี้จะเรียกการเข้ารหัสสองแบบนี้รวมกันเป็น TIS-620.

`iconv` อ้างอิงหน้า 388

ตัวอย่างที่ 7.1: แปลงข้อมูล TIS-620 เป็น UTF-8.

```
$ iconv -f TIS-620 -t UTF-8 -o thai_utf8.txt thai_tis620.txt↵
```

ตัวเลือก `-f` (from) ใช้ระบุการเข้ารหัสของข้อมูลนำเข้าและตัวเลือก `-t` (to) ใช้ระบุการเข้ารหัสของผลลัพธ์ที่ต้องการ. คำสั่ง `iconv` จะรับข้อมูลจาก stdin ถ้าไม่ระบุชื่อไฟล์และจะส่งผลลัพธ์ออกทาง stdout ถ้าไม่ระบุชื่อไฟล์ที่ต้องการเก็บผลลัพธ์. ชื่อการเข้ารหัสที่คำสั่ง `iconv` สามารถแปลงได้ดูได้จากตัวเลือก `-l`.

ตัวอย่างที่ 7.2: ชื่อการเข้ารหัสต่าง ๆที่รองรับในคำสั่ง `iconv`

```
$ iconv -l↵
--- แสดงผลต่อไปเรื่อยๆ ---
```

```
    TCVN5712-1, TCVN5712-1:1993, TIS-620, TIS620-0, TIS620.2529-1, TIS620.2533-0,
    TIS620, TS-5881, TSCII, UCS-2, UCS-2BE, UCS-2LE, UCS-4, UCS-4BE, UCS-4LE,
    UCS2, UCS4, UHC, UJIS, UK, UNICODE, UNICODEBIG, UNICODELITTLE, US-ASCII, US,
    UTF-7, UTF-8, UTF-16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE, UTF-32LE, UTF7,
    UTF8, UTF16, UTF16BE, UTF16LE, UTF32, UTF32BE, UTF32LE, VISCII, WCHAR_T,
--- แสดงผลต่อไปเรื่อยๆ ---
```

ชื่อการเข้ารหัสที่เกี่ยวข้องกับภาษาไทยได้แก่ TIS-620, TIS620-0, TIS620.25259-1, TIS620.2533-0, TIS620 และ CP874. การเข้ารหัสแบบ TIS-620 มีหลายชื่อให้เลือก แต่ผลที่ได้ไม่มีความแตกต่างกันและสามารถระบุชื่อการเข้ารหัสด้วยตัวอักษรตัวเล็กเช่น tis-620 ได้ด้วย. CP874 เป็นการเข้ารหัสอักขระภาษาไทยในระบบปฏิบัติการวินโดวส์ซึ่งคล้ายเหมือนกับ TIS-620.

คำสั่งที่คล้ายกับ `iconv` อีกตัวคือคำสั่ง `convmv` ใช้สำหรับแปลงการเข้ารหัสอักขระของชื่อไฟล์.

7.2 โลแคล

โลแคล (locale) คือแนวคิดการกำหนดสภาพแวดล้อมภาษารวมถึงวัฒนธรรมของผู้ใช้ในระบบคอมพิวเตอร์. แนวคิดนี้มีเริ่มใช้ในมาตรฐาน ISO 9899 ซึ่งเป็นมาตรฐานสากลของภาษา C. แนวคิดเรื่องโลแคลมีผลดีต่อผู้พัฒนาซอฟต์แวร์และผู้ใช้. สำหรับผู้พัฒนาซอฟต์แวร์สามารถพัฒนาซอฟต์แวร์โดยไม่ต้องคำนึงถึงการประมวลผลที่เกี่ยวกับภาษา. ฟังก์ชันไลบรารีมาตรฐานในภาษา C เช่น `isalnum`, `isdigit`, `tolower` ฯลฯ จะรับรู้ข้อมูลเกี่ยวกับภาษาที่ต้องการประมวลผล. ผู้พัฒนาไม่ต้องเขียนโค้ดเพื่อประมวลผลสิ่งเหล่านี้เองเมื่อสภาพแวดล้อมของภาษาเปลี่ยนไป. ในแง่ของผู้ใช้โปรแกรม, สามารถเลือกภาษาที่ใช้ในการแสดงผลตามต้องการ ฯลฯ.

รูปที่ 7.4: การแสดงข้อความในโปรแกรมเมื่อสภาพแวดล้อมโลแคลต่างกัน.

7.2.1 ชื่อโลแคล

ในระบบที่รองรับการใช้โลแคลจะมีฐานข้อมูลโลแคลของภาษาต่าง ๆแยกตามชื่อโลแคล. ชื่อโลแคลมีรูปแบบทั่วไปดังนี้.

```
language[_territory][.codeset]
```

ตัวอย่างชื่อโลแคลเช่น th, th_TH, th_TH.TIS-620, th_TH.UTF-8. ส่วนที่ประกอบเป็นชื่อโลแคลได้แก่.

- *language* — ชื่อภาษาเช่นภาษาไทย th, ภาษาอังกฤษ en, ภาษาญี่ปุ่น ja เป็นต้น. ชื่อภาษานี้เป็นชื่ออักษรภาษาอังกฤษ 2 ตัวตามมาตรฐาน ISO639.

- *territory* — ชื่อประเทศที่ใช้ภาษานั้น. ภาษาเช่นภาษาอังกฤษใช้กันแพร่หลายในประเทศ. ประเทศสหราชอณาจักรอังกฤษใช้ภาษาอังกฤษ, ประเทศสหรัฐอเมริกาก็ใช้ภาษาอังกฤษ. ถึงแม้ว่าจะใช้ภาษาอังกฤษเหมือนกันแต่ภาษาหรือวัฒนธรรมที่ใช้ในแต่ละท้องถิ่นมีความแตกต่างกันในรายละเอียด. และส่วนที่เรียกว่า territory นี้เป็นตัวบ่งบอกสถานที่ที่ใช้ภาษานั้น. ตัวอย่างเช่น TH (Thailand), GB (Great Britain), US (United State of America), JP (Japan) ฯลฯ. ชื่อ territory จะเป็นอักษรภาษาอังกฤษตัวใหญ่สองตัวตามมาตรฐาน ISO3166. สำหรับภาษาที่มีใช้ในประเทศเดียวเท่านั้นเช่นภาษาไทยสามารถละเลยชื่อโลแคลส่วนนี้ได้. ถ้าต้องการใช้ส่วน territory ให้เขียนใช้เครื่องหมาย _ ต่อจากชื่อภาษา. ตัวอย่างชื่อโลแคลที่มีส่วน territory เช่น th_TH, en_GB, en_US, ja_JP เป็นต้น.

- *codeset* — ส่วนสุดท้ายในชื่อโลแคลได้แก่การเข้ารหัสที่ใช้ในโลแคล.

คำสั่ง locale เป็นคำสั่งสำหรับแสดงฐานข้อมูลโลแคลในระบบเช่นแสดงชื่อโลแคลในระบบ (ตัวเลือก -a) หรือแสดงชื่อการเข้ารหัสอักขระต่าง ๆ (ตัวเลือก -m). ถ้าสั่งคำสั่งโดยไม่ระบุอาร์กิวเมนต์ใด ๆจะแสดงโลแคลของผู้ใช้.

locale อ้างอิงหน้า 413

ตัวอย่างที่ 7.3: แสดงชื่อโลแคลในฐานข้อมูลที่มีในระบบ.

```
$ locale -a↵
C
POSIX
aa_DJ
aa_DJ.iso88591
--- แสดงผลต่อไปเรื่อยๆ ---
tg_TJ.koi8t
th_TH
th_TH.tis620                                ← หรือ th_TH.TIS620, th_TH.TIS-620
th_TH.utf8                                      ← หรือ th_TH.UTF8, th_TH.UTF-8
thai
ti_ER
--- แสดงผลต่อไปเรื่อยๆ ---
```

ชื่อโลแคล C หรือ POSIX เป็นชื่อโลแคลพิเศษซึ่งไม่เกี่ยวข้องกับภาษาใด ๆ. หมายถึงให้ระบบประมวลผลข้อมูลโดยที่ไม่ต้องคำนึงถึงเรื่องเกี่ยวกับภาษาและวัฒนธรรม.

สำหรับโลแคลบางตัวเช่นภาษาไทยสามารถมีการเข้ารหัสอักขระได้มากกว่าหนึ่งแบบ ดังนั้นชื่อโลแคลควรระบุให้ชัดเจนเช่น th_TH.TIS-620 หรือ th_TH.UTF-8. ชื่อโลแคลเช่น th_TH และ thai ที่ไม่ระบุการเข้ารหัสอักขระเป็นชื่อโลแคลแบบย่อ (alias) ที่กำหนดในไฟล์ /usr/share/locale/locale.alias หรือ /usr/lib/X11/locale/locale.alias.

7.2.2 สร้างโลแคล

localedef อ้างอิงหน้า 406

ในระบบบางระบบอาจจะไม่มีฐานข้อมูลของโลแคลทุกตัว. ผู้ดูแลระบบสามารถสร้างฐานข้อมูลโลแคลเพิ่มเติมได้ด้วยคำสั่ง localedef. การสร้างฐานโลแคลใหม่ต้องการข้อมูลเกี่ยวกับโลแคลอย่างน้อยสองอย่างคือข้อมูลดิบเกี่ยวกับตัวโลแคลและข้อมูลเกี่ยวกับการเข้ารหัสอักขระ. ในระบบที่รองรับการใช้โลแคลจะมีข้อมูลดิบของโลแคลเป็นไฟล์ชื่อโลแคลต่าง ๆเก็บไว้ในไดเรกทอรี /usr/share/i18n/locales. ข้อมูลเกี่ยวกับการเข้ารหัสอักขระจะเก็บไว้ในไดเรกทอรี /usr/share/i18n/charmaps ในรูปของไฟล์ที่บีบอัดด้วย gzip.

ตัวอย่างต่อไปนี้เป็นการสร้างโลแคล th_TH.TIS-620 ด้วยคำสั่ง localedef. หลังจากที่สร้างโลแคลแล้วสามารถตรวจสอบโลแคลที่สร้างใหม่ด้วยคำสั่ง locale -a. ในบางดิสโทรมาคำสั่งเฉพาะสำหรับสร้างโลแคลเช่น Debian มีคำสั่ง locale-gen ซึ่งเป็นเชลล์สคริปต์และเรียกใช้คำสั่ง localedef เช่นกัน.

ตัวอย่างที่ 7.4: สร้างโลแคลในระบบ.

```
# gzip -dc /usr/share/i18n/charmaps/TIS-620.gz > /tmp/TIS-620↵
# localedef -f /tmp/TIS-620 -i /usr/share/i18n/locales/th_TH th_TH.TIS-620↵
```

7.2.3 ตัวแปรสภาพแวดล้อมที่เกี่ยวกับโลแคล

การทำงานของโลแคลจะอิมพลีเมนต์ (implement) อยู่ในไลบรารีภาษา C ของ GNU (GNU C library). ผู้ใช้สามารถกำหนดสภาพแวดล้อมภาษาของตนเองที่ต้องการใช้ด้วยตัวแปรสภาพแวดล้อมต่าง ๆดังต่อไปนี้.

- LC_CTYPE —— ตัวแปรสภาพแวดล้อมเกี่ยวกับการแยกแยะประเภทของอักขระ. เช่นในภาษาอังกฤษ, โปรแกรมสามารถแปลงอักษร “A” ให้เป็นตัวอักษรตัวเล็ก “a” ได้ซึ่งมีความหมายเดียวกัน. แต่ในสภาพแวดล้อมภาษาไทยเช่น “ก” ไม่มีแนวคิดเรื่องอักษรตัวใหญ่หรือตัวเล็ก.

- LC_COLLATE —— ตัวแปรสภาพแวดล้อมเกี่ยวกับการจัดลำดับข้อมูล. ถ้าค่าของตัวแปรไม่เหมาะสมกับภาษาที่ใช้, โปรแกรมจัดลำดับข้อมูลอาจจะทำงานผิดพลาดได้. เช่นการใช้คำสั่ง sort เรียงลำดับคำภาษาไทยในสภาพแวดล้อมภาษาอังกฤษจะให้ผลไม่ถูกต้องเพราะวิธีเรียงลำดับข้อมูลแตกต่างตามภาษาที่ใช้.

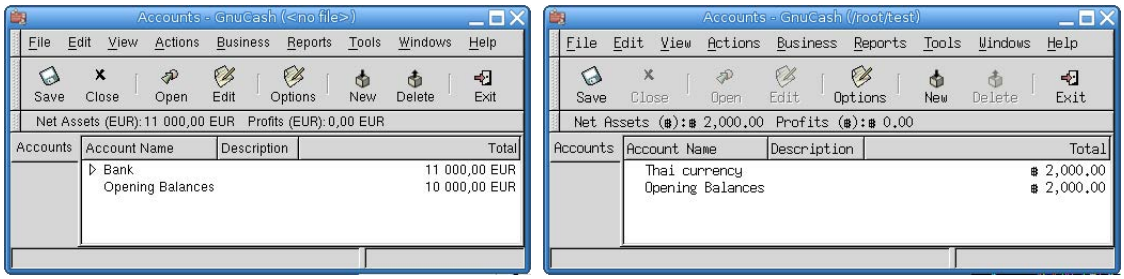
ตัวอย่างที่ 7.5: ผลกระทบของภาษากับการเรียงลำดับข้อมูล.

```
$ cat unsorted.txt↵
zebra
ไก่
กา
ช้าง
กบ
ant
$ LC_COLLATE=C sort unsorted.txt↵        ← เรียงลำดับข้อมูลโดยไม่คำนึงถึงภาษา
ant
zebra
กบ
กา
ช้าง                                                    ← เรียงลำดับผิด
ไก่
$ LC_COLLATE=th_TH.TIS-620 sort unsorted.txt↵
ant
zebra
กบ
กา
ไก่
ช้าง
```

- LC_MESSAGES — โปรแกรมสามารถแสดงข้อความด้วยภาษาที่ระบุในตัวแปรสภาพแวดล้อมนี้เช่นในรูปที่ 7.4.

- LC_MONETARY — ตัวแปรสภาพแวดล้อมสำหรับแสดงสกุลเงินตรา.

- LC_NUMERIC — ตัวแปรสภาพแวดล้อมเกี่ยวกับการแสดงตัวเลขเช่นจุดทศนิยม.



รูปที่ 7.5: การแสดงตัวเลขและสกุลเงินตราตามโลเคล.

- LC_TIME — แสดงวันเดือนปีตามภาษาที่กำหนด. เช่นประเทศแถบยุโรปมักจะแสดงเวลาแบบ 24 ชั่วในขณะที่สหรัฐอเมริการแสดงเวลาเป็น 12 ชั่วโมง (AM/PM).

- LC_ALL — ตัวแปรสภาพแวดล้อมพิเศษสำหรับบังคับการกำหนดค่าโลแคลของตัวแปรสภาพแวดล้อมที่ขึ้นต้นด้วย LC_ ตัวอื่น ๆ. สมมติว่าเดิมกำหนดค่า LC_MESSAGES เป็น th_TH แต่มีการกำหนดค่า LC_ALL เป็น en. ค่าตัวแปลสภาพแวดล้อมที่

ขึ้นต้นด้วย LC_ รวมถึง LC_MESSAGES ด้วยจะใช้ค่าที่กำหนดด้วยตัวแปรสภาพแวดล้อม LC_ALL.

- LANG —— ตัวแปรสภาพแวดล้อมพิเศษคล้ายกับ LC_ALL แต่จะมีผลกับตัวแปรสภาพแวดล้อมที่ขึ้นต้นด้วย LC_ ที่ไม่ได้ตั้งค่าไว้หรือมีค่าเป็น en หรือ POSIX เท่านั้น. ตัวอย่างเช่น LC_MESSAGES ไม่มีค่าตั้งไว้แต่มีการตั้งค่า LANG เป็น th_TH. ระบบจะถือว่า LC_MESSAGES มีค่าเป็น th_TH โดยปริยาย.

หลังจากที่รู้จัดตัวแปรสภาพแวดล้อมต่าง ๆที่เกี่ยวข้องกับโลแคลแล้วเราสามารถปรับแต่งสภาพแวดล้อมได้อย่างมีประสิทธิภาพขึ้น. ตัวอย่างเช่นเราต้องการใช้สภาพแวดล้อมที่เป็นภาษาไทยเท่าที่เป็นไปได้แต่ไม่ต้องการแสดงวันเวลาเป็นภาษาไทยก็สามารถตั้งค่าตัวแปรสภาพแวดล้อมได้ดังนี้.

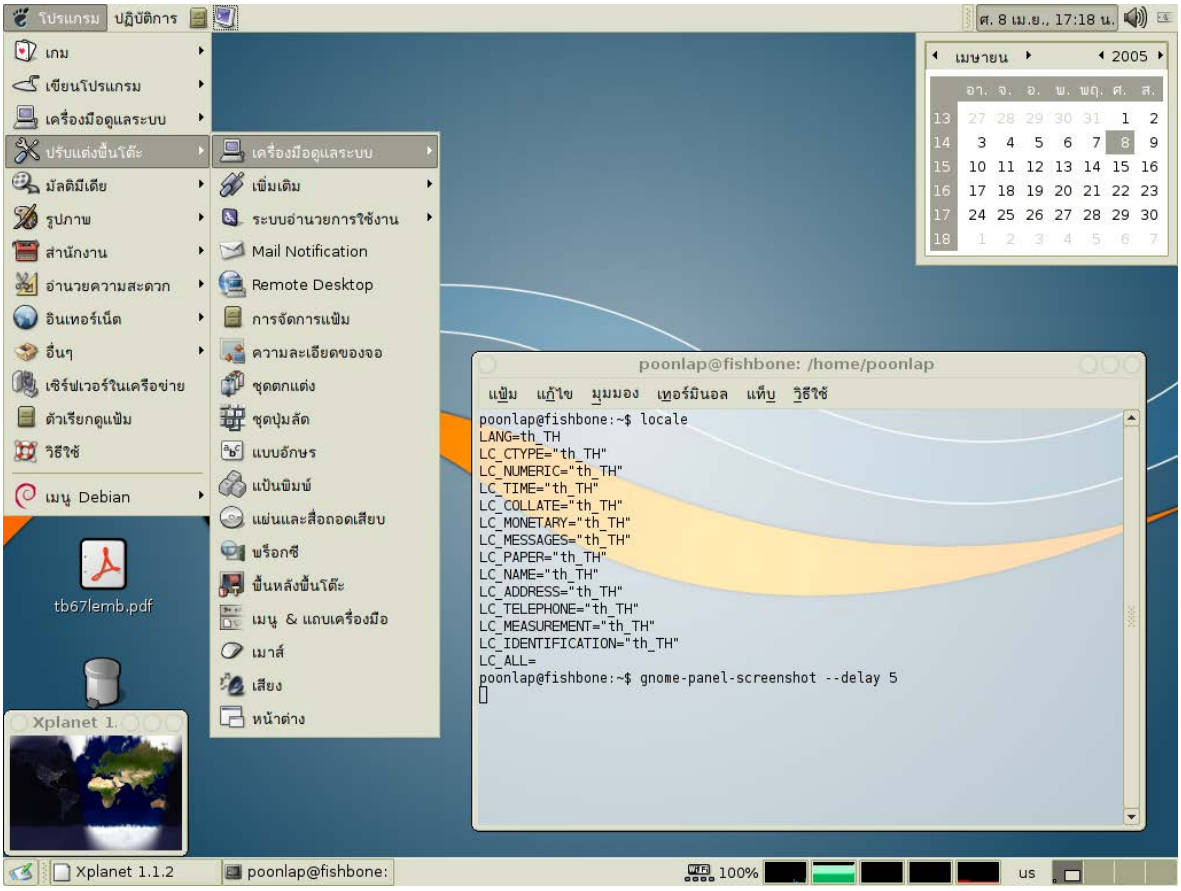
ตัวอย่างที่ 7.6: การปรับแต่งค่าโลแคล.

```
$ export LC_TIME=en_US↵
$ export LANG=th_TH↵
$ locale↵
LANG=th_TH
LC_CTYPE="th_TH"
LC_NUMERIC="th_TH"
LC_TIME=en_US                          ← ยังเป็นภาษาอังกฤษหลังจากตั้งค่า LANG
LC_COLLATE="th_TH"
LC_MONETARY="th_TH"
LC_MESSAGES="th_TH"
LC_PAPER="th_TH"
LC_NAME="th_TH"
LC_ADDRESS="th_TH"
LC_TELEPHONE="th_TH"
LC_MEASUREMENT="th_TH"
LC_IDENTIFICATION="th_TH"
LC_ALL=
```

7.3 การแสดงอักขระภาษาไทย

จากช่วงที่ผ่านมาเราได้ทำความรู้จักกับอักขระภาษาไทย, การเข้ารหัสอักขระข้อมูล, ตลอดจนวิธีการระบุสภาพแวดล้อมที่ใช้งานด้วยโลแคล. สิ่งที่เราควรจะทำความรู้จักต่อไปคือการแสดงผลอักขระภาษาไทยทางหน้าจอ. การแสดงผลภาษาไทยสามารถทำได้ทั้งในเท็กซ์โหมด (คอนโซล) และในระบบ X วินโดว์. ในที่นี้จะเน้นอธิบายการแสดงอักขระภาษาไทยในระบบ X วินโดว์เป็นหลัก.

ฟอนต์แบบดั้งเดิมที่ใช้ในระบบ X วินโดว์เป็นฟอนต์แบบบิตแมปและมีความสัมพันธ์กับชุดรหัสอักขระที่ต้องการใช้.



รูปที่ 7.6: สภาพแวดล้อมเดสก์ท็อป GNOME ในโลแคลไทย.

7.3.1 ประวัติฟอนต์ภาษาไทยที่ใช้ในลินุกซ์

ฟอนต์ภาษาไทยในยุคแรก ๆสร้างขึ้นมาเพื่อใช้กับระบบ X วินโดว์บนระบบปฏิบัติการยูนิกซ์. เท่าที่ผู้เขียนสืบประวัติได้ฟอนต์ในยุคแรกสร้างขึ้นในปี พ.ศ.2535 โดย คุณวรเดช เย็นบุตร มหาวิทยาลัยวอชิงตัน. ฟอนต์นี้เป็นฟอนต์บิตแมป BDF ชื่อ thai8x13 และ thai9x13 มีรูปทรงแบบเดียวคือรูปตัวตรงและไม่ใช้ชื่อฟอนต์ตามหลัก XFLD. ฟอนต์นี้ต่อมาเผยแพร่ทางเว็บไซต์ของกลุ่มนักเรียนไทยในมหาวิทยาลัย Tokyo Institute of Techonology (TIT) นำโดยคุณมานพ วงศ์สายสุวรรณและคุณวุฒิชัย อัมพรอร่ามเวทย์.

ในปี พ.ศ.2537 คุณมานพเพิ่มแก้ไขฟอนต์บิตแมปของโครงการบรรณาธิกรณ์ `mule` โดยเพิ่มสร้างฟอนต์ส่วนที่เป็นภาษาไทยเพิ่มเข้าไปแล้วตั้งชื่อฟอนต์ใหม่สำหรับใช้กับภาษาไทย 3 ฟอนต์ได้แก่ฟอนต์ thai6x14, thai7x18 และ thai8x20. ฟอนต์เหล่านี้มีชื่อตาม XFLD และใช้ชุดรหัสอักขระเป็น tis620.25259-1 และสิทธิ์อนุญาตการใช้เป็น public domain. ในขณะเดียวกันกลุ่มนักเรียนไทยใน TIT ได้แปลงฟอนต์ทรูไทป์ภาษาไทย DBThaiText ให้เป็นฟอนต์บิตแมปแบบ BDF เพื่อใช้กับระบบ X วินโดว์อีกด้วย. รูปทรงฟอนต์ DBThaiText ที่แปลงมาจากฟอนต์ทรูไทป์ไม่ค่อยสวยงามนักเพราะเป็นการแปลงฟอนต์โดยการใช้โปรแกรมแปลงฟอนต์แต่มีข้อดีที่ว่ามีฟอนต์หลายขนาดและมีรูปทรงที่

บรรณาธิกรณ์ `mule` เป็นบรรณาธิกรณ์ที่สร้างต่อเติมจาก `emacs` รองรับการใช้ภาษานานาชาติเช่น ภาษาญี่ปุ่น, ภาษาไทย ฯลฯ. ปัจจุบันได้รวมกลับเข้ามาในรหัสต้นฉบับของ `emacs` แล้ว.

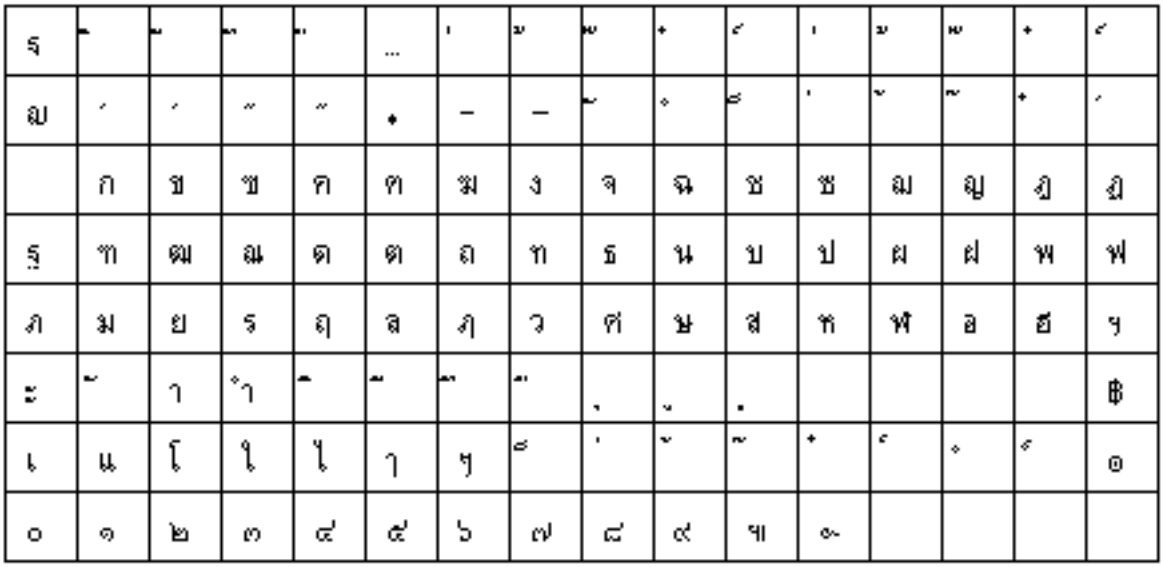
รูปที่ 7.7: ฟอนต์ thai9x13.

สำคัญ ๆครบถ้วนได้แก่ตัวหนาและตัวเอียง.

รูปที่ 7.8: ฟอนต์ thai6x14.

ในปี พ.ศ.2539 กลุ่มนักเรียนในมหาวิทยาลัย The University of Electro-communications, Tokyo เริ่มโครงการ “สื่อไทย” (ZzzThai) มีจุดประสงค์ส่งเสริมการใช้ภาษาไทยกับคอมพิวเตอร์ในแพล็ตฟอร์มหลัก ๆได้แก่ยูนิกซ์, วินโดวส์ และแมคอินทอช. โครงการนี้รวบรวมฟอนต์และจัดทำเอกสารในรูปของเว็บไซต์อธิบายการติดตั้งและใช้งานภาษาไทยเป็นขั้นเป็นตอนพร้อมรูปประกอบ. โครงการที่สืบต่อมาจากโครงการสื่อไทยและเกี่ยวข้องกับลินุกซ์ได้แก่โครงการ *Thai Extension* หรือเรียกย่อ ๆว่า TE เป็นโครงการรวมฟอนต์, โปรแกรมต่าง ๆที่เกี่ยวกับภาษาไทยรวมเป็นแพ็กเกจสำหรับติดตั้งเพิ่มในลินุกซ์ดิสทริบิวชันที่มีอยู่แล้วให้ใช้ภาษาไทยได้ทันที. โครงการนี้ร่วมกันทำโดยคุณ ไพศาล เตชะจารุวงศ์. และผู้เขียน. มีการแก้ไขฟอนต์ต่าง ๆให้เช่นชื่อ XFLD ของฟอนต์ที่มีมาให้ถูกต้องและเผยแพร่ฟอนต์บิตแมปใหม่ที่สร้างโดยคุณไพศาลเช่นฟอนต์ phaisarn-sanserif, phaisarn-thaismall, phaisarn-thaicom, phaisarn-thaimedia ฯลฯ. ฟอนต์ไพศาลบางตัวแตกต่างจาก

ฟอนต์ที่เคยมีมาตรงที่มีอักขระพิเศษเพิ่มเติมเช่นมีวรรณยุกต์ 3 ชุดสำหรับการแสดงผลในตำแหน่งต่าง ๆให้สวยขึ้นเป็นต้น.

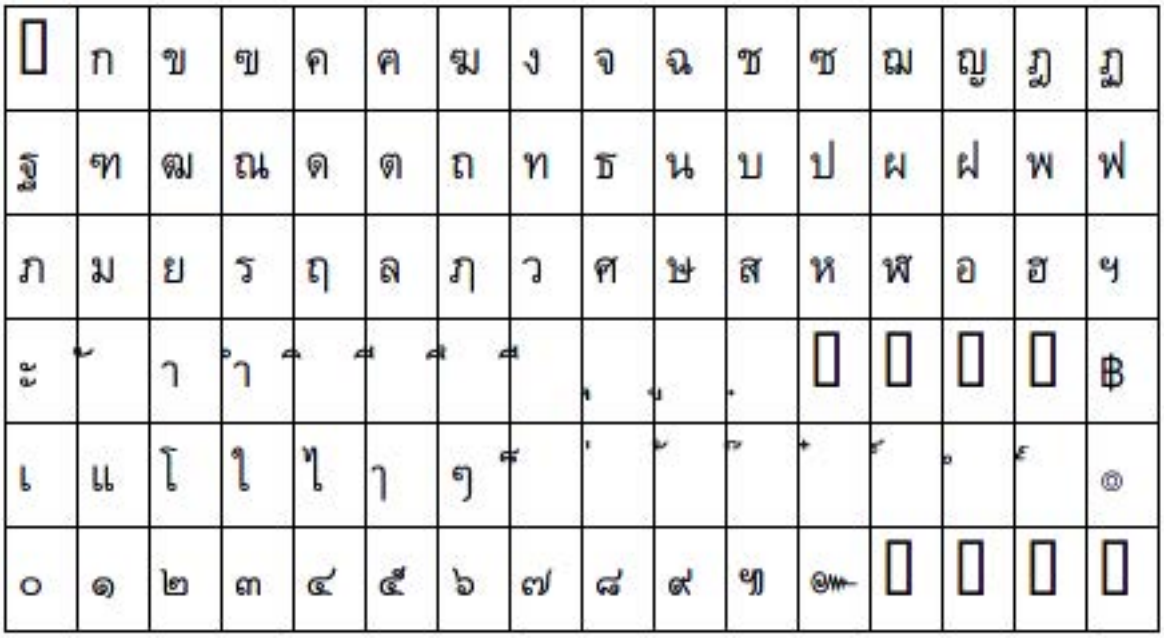


รูปที่ 7.9: ฟอนต์ -phaisarn-sanserif-medium-r-normal—14-140-100-100-p-80-tis620-2.

ในปี พ.ศ.2542 ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติหรือ NECTEC ได้เริ่มโครงการฟอนต์แห่งชาติและเผยแพร่ฟอนต์ที่สำคัญ ๆได้แก่ฟอนต์ nectec-fixed หรือเรียกสั้น ๆว่า nectec18 เป็นฟอนต์บิตแมปที่ความกว้างของอักขระทุกตัวเท่ากัน, ยกเว้นสระและวรรณยุกต์บางตัวที่มีความกว้างเป็นศูนย์. รูปทรงอักขระของฟอนต์นี้อ้วน, อ่านง่ายและเป็นฟอนต์ที่มีรูปร่างครบถ้วนได้แก่ตัวธรรมดา, ตัวหนา, ตัวเอียงและตัวหนาเอียง. ฟอนต์นี้นิยมใช้กับเทอร์มินอลเอมิวเลเตอร์ที่รองรับอักขระไทยเช่น `xiterm` ที่ได้รับการแพชท์แล้ว.

รูปที่ 7.10: ฟอนต์ nectec-fixed.

นอกจากฟอนต์บิตแมปแล้วเนคเทคยังเผยแพร่ฟอนต์แห่งชาติแบบทรูไทป์สำหรับใช้กับระบบ X วินโดว์บนลินุกซ์ได้แก่ฟอนต์ นรสีห์ (Norasi), ครุฑ (Garuda) และกินรี (Kinnari). ฟอนต์เหล่านี้มีตัวหนา, เอียงครบทุกชุด, ช่วยให้การแสดงผลอักขระภาษาไทยในระบบ X วินโดว์สวยงามขึ้นโดยเฉพาะระบบ X วินโดว์ที่รองรับการแสดงอักขระแบบแอนติอเลียส.



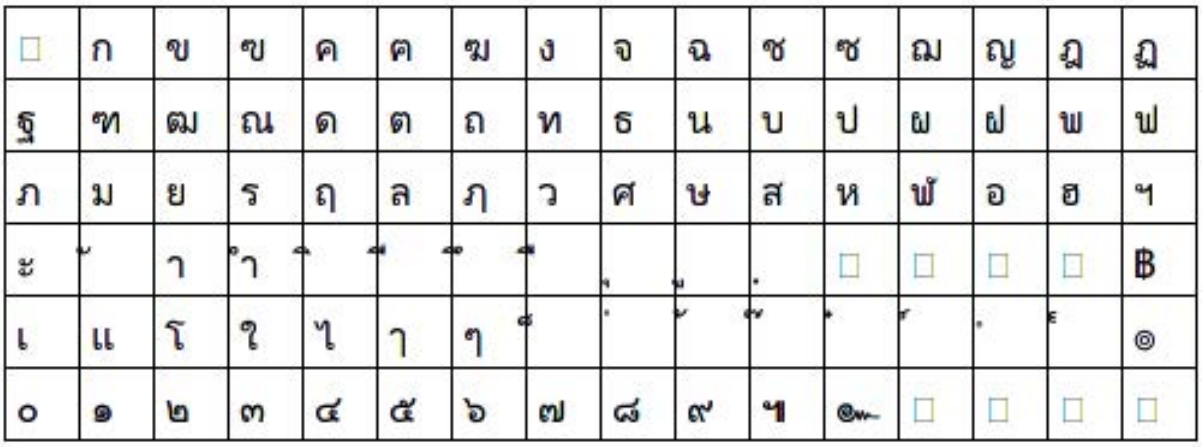
รูปที่ 7.11: ฟอนต์ Garuda.

หลังจากที่มีโครงการลินุกซ์ดิสทริบิวชันภาษาไทยลินุกซ์ทะเล (Linux TLE), หนึ่งในทีมพัฒนาลินุกซ์ทะเลคือคุณศิริชัย เลิศวรวุฒิ ได้สร้างฟอนต์ทรูไทป์ตัวใหม่ชื่อฟอนต์ โลมา (Loma) ในปี พ.ศ.2546. ฟอนต์นี้รวมอยู่ในลินุกซ์ทะเลเผยแพร่ด้วยสิทธิ์การอนุญาตแบบ GPL. ฟอนต์นี้รูปทรงอ้วนกลมอ่านง่ายและเป็นฟอนต์โดยปริยายในลินุกซ์ทะเล.

ฟอนต์ทรูไทป์ภาษาไทยที่ใช้กันในลินุกซ์มักจะเป็นฟอนต์ที่อักขระมีความกว้างไม่คงที่และฟอนต์ภาษาไทยยังขาดฟอนต์ทรูไทป์ที่อักขระมีความกว้างคงที่อยู่. ในปี พ.ศ.2546 หลังจากที่ทีมงานลินุกซ์ทะเลเริ่มสร้างฟอนต์โลมาได้ไม่นาน, ผู้เขียนได้สร้างฟอนต์ทรูไทป์ที่อักขระทุกตัวมีความกว้างเท่ากันชื่อ TlwgMono และเผยแพร่ด้วยสิทธิ์การใช้แบบ GPL. ฟอนต์นี้นำส่วนอักขระภาษาอังกฤษมาจากฟอนต์ FreeMono และสร้างส่วนที่เป็นอักขระภาษาไทยด้วยโปรแกรม `fontforge` ซึ่งเป็นซอฟต์เสรี.

โปรแกรม `fontforge` เดิมชื่อ `pfaedit`.

ปัจจุบัน, ฟอนต์ทรูไทป์ภาษาไทยที่แนะนำไปแล้วรวมไว้อยู่ในโครงการ thaifonts-scalable ได้รับการดูแลและพัฒนาต่อเติมเช่นเพิ่มตาราง OpenType, ปรับแต่งรูปร่างของอักขระ โดยอาสาสมัครกลุ่ม Thai Linux Working Group มี คุณเทพพิทักษ์ การุณบุญญานันท์ เป็นแกนนำ.



รูปที่ 7.12: ฟอนต์ Loma.

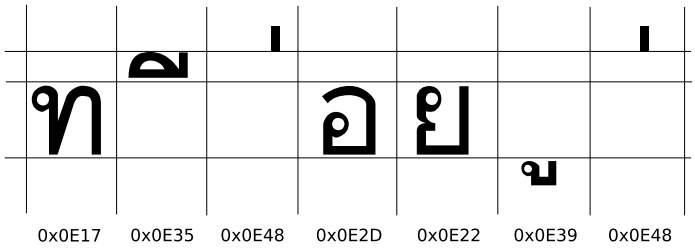
รูปที่ 7.13: ฟอนต์ TlwgMono.

7.3.2 กลีฟ

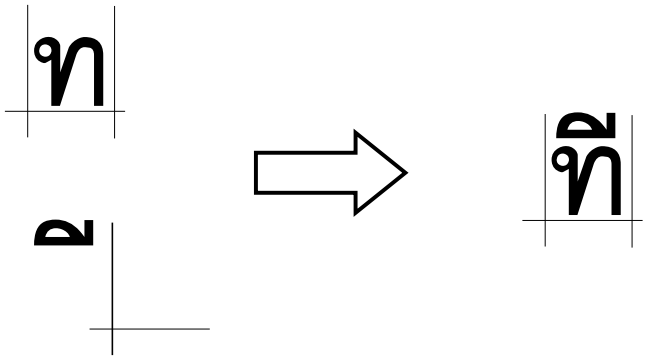
กลีฟคือรูปทรงอักขระในฟอนต์สำหรับแสดงผลทางหน้าจอ. โดยปรกติ, กลีฟจะมีความกว้างเฉพาะแต่ละตัว. แต่ถ้าอักขระภาษาไทยทุกตัวมีความกว้าง, เวลาแสดงผลตำแหน่งของสระและวรรณยุกต์บางตัวจะผิดปรกติตามที่แสดงในรูปที่ 7.14. โปรแกรมที่ต้องการฟอนต์ที่อักขระทุกตัวมีความกว้างเท่ากันเช่น `xterm` และโปรแกรมเทอร์มินอลเอมิวเลเตอร์ต่าง ๆและตัวโปรแกรมจะจัดการแสดงผลเอง. ฟอนต์ภาษาไทยที่อักขระทุกตัวมีความกว้างเท่ากันหมดได้แก่ TlwgMono, -misc-fixed-medium-r-normal--18-120-100-100-c-90-iso10646-1 เป็นต้น.

ฟอนต์ misc-fixed ที่มีอักขระภาษาไทยเป็นฟอนต์ที่รวมอยู่ในแพ็กเกจของ X เซิร์ฟเวอร์.

สระและวรรณยุกต์บางตัวในฟอนต์ภาษาไทยส่วนใหญ่มักจะมีความกว้างเป็นศูนย์และตำแหน่งของกลีฟที่แสดงผลจะเยื้องไปหาพยัญชนะที่แสดงผลไปแล้ว. ฟอนต์ที่มีกลีฟ



รูปที่ 7.14: ฟอนต์ที่กลีฟแต่ละตัวมีความกว้าง.

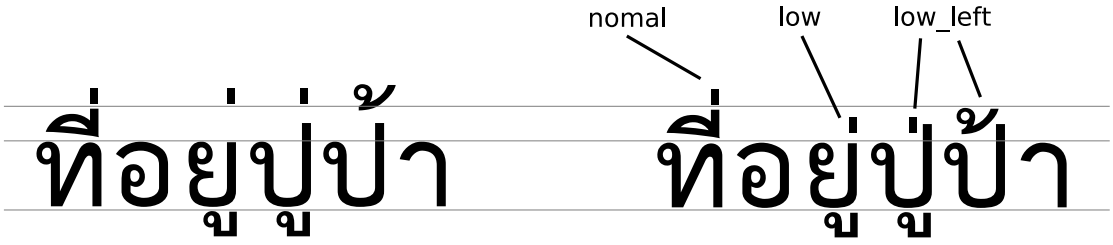


รูปที่ 7.15: กลีฟสระภาษาไทยที่มีความกว้างเป็นศูนย์.

ในลักษณะนี้ช่วยให้โปรแกรมแสดงผลอักขระได้ทันทีโดยที่ตัวโปรแกรมไม่ต้องจัดลำดับอักขระเอง. ฟอนต์ภาษาไทยที่กลีฟแบบนี้เช่น ฟอนต์แห่งชาติ, Loma, Tlwg Typewriter, phaisarn-sanserif, ฟอนต์ทรูไทป์ในระบบปฏิบัติการวินโดวส์ เป็นต้น.

กลีฟในฟอนต์ภาษาไทยบางตัวจะมีสระและวรรณยุกต์บางตัวมากกว่าหนึ่งชุดเพื่อการแสดงผลที่สวยงาม. ตัวอย่างเช่นไม่เอกในรูปที่ 7.16 มีตำแหน่งที่เป็นไปได้สามที่คือ ตำแหน่งปรกติ, low และ low_left. โปรแกรมที่รับรู้กลีฟเหล่านี้และสามารถจัดระดับสระและวรรณยุกต์ได้ถูกต้องจะไม่เกิดปัญหาที่เรียกว่าสระลอย. การจัดระดับสระวรรณยุกต์ไม่จำกัดเฉพาะสระที่อยู่บนพยัญชนะเท่านั้นการจัดระดับนี้รวมถึงสระอุ, สระอูตำแหน่งที่ต่ำกว่าปรกติสำหรับใช้กับพยัญชนะที่มีหางยาว “ฎ”, “ฏ”. ตัว “ญ” และ “ฐ” ที่ลดรูปเช่นคำว่า “กตัญญู” เป็นต้น.

ฟอนต์ภาษาไทยแบบดั้งเดิมได้แก่ฟอนต์บิตแมปจะมีส่วนขยายชื่อฟอนต์ XFLD เป็น tis620-0 หรือ tis620-2. จากชื่อฟอนต์เช่น -thai-fixed-medium-r-normal--16-114-100-100-p-100-tis620-0 สามารถบอกได้ว่าเป็นฟอนต์ที่ใช้กับชุดรหัสอักขระ TIS-620 และไม่มีกลีฟพิเศษสำหรับจัดระดับสระวรรณยุกต์. ฟอนต์เช่น -phaisarn-sanserif-medium-r-normal--14-140-100-100-p-80-tis620-2 เป็นฟอนต์ที่ใช้กับชุดรหัสอักขระ TIS-620 เช่นกันแต่มีกลีฟพิเศษสำหรับการจัดระดับสระวรรณยุกต์ในตำแหน่งรหัส 0x80 - 0x9F. ฟอนต์ภาษาไทยในระบบ fontconfig ซึ่งเป็นฟอนต์ทรูไทป์และเป็นฟอนต์แบบยูนิโค้ดจะมีกลีฟพิเศษในทำนองเดียวกันแต่เก็บไว้ในช่วงรหัสที่เรียกว่า *PUA (Private Use Area)*. PUA เป็นช่วงรหัสในฟอนต์ที่ใช้เก็บกลีฟพิเศษไม่มีกฎเกณฑ์ที่แน่นอน. สำหรับฟอนต์



รูปที่ 7.16: การจัดระดับตำแหน่งสระและวรรณยุกต์.

ภาษาไทยจะนิยมเก็บกลีฟพิเศษต่าง ๆไว้ในช่วงรหัส 0xF700 - F71A. โปรแกรมสมัยใหม่ที่สามารถใช้ความสามารถของ OpenType จะไม่มีกลีฟพิเศษอีกต่อไปแต่จะใช้ข้อมูลการจัดระดับอักขระต่างจากตารางที่เรียกว่า *GPOS (Glyph Positioning)* และ *GSUB (Glyph Substitution)* ในฟอนต์แบบ OpenType.

การจัดระดับสระวรรณยุกต์และแสดงผลภาษาไทยในสภาพแวดล้อมเดสก์ท็อป GNOME จะใช้ไลบรารีที่เรียกว่า *Pango*. ไลบรารี Pango เป็นไลบรารีที่เกี่ยวกับกับการแสดงผลอักขระภาษานานาชาติและมีเกี่ยวข้องโดยตรงกับ GTK+, Xft และ fontconfig. ในส่วนที่เกี่ยวข้องกับการแสดงผลภาษาไทยเริ่มแรกเป็นการร่วมพัฒนาของ คุณชูกิจ วนาธรรม และต่อมามี คุณเทพพิทักษ์ พัฒนาต่อในปัจจุบัน.

7.4 การป้อนข้อมูล

ถ้าเราพิจารณาการแสดงผลอักขระภาษาไทยเช่นคำว่า “ที่” จะสังเกตเห็นว่าเกิดจากการรวมตัวของอักขระ 3 ตัวซึ่งถ้าใช้เครื่องพิมพ์ดีดพิมพ์สามารถพิมพ์ได้สองวิธีและให้ผลเหมือนกันได้แก่ ท ◌ี ◌่ หรือ ท ◌่ ◌ี. การแสดงผลของทั้งสองวิธีให้ผลเหมือนกันเนื่องจากกลีฟของสระอีและไม้เอกไม่มีความกว้าง. แต่ในแง่ของการป้อนข้อมูลให้กับคอมพิวเตอร์ ท ◌ี ◌่ (0x0E17 0x0E35 0x0E48) และ ท ◌่ ◌ี (0x0E17 0x0E48 0x0E35) เป็นสายข้อมูลที่แตกต่างกันอย่างสิ้นเชิง. ดังนั้นการป้อนข้อมูลให้ถูกต้องจึงมีความสำคัญสำหรับคอมพิวเตอร์โดยเฉพาะอย่างยิ่งการค้นหาข้อมูลจะเป็นปัญหาใหญ่ถ้าการป้อนข้อมูลผิดพลาดไม่เป็นมาตรฐานทำให้หาข้อมูลไม่เจอทั้ง ๆที่มีข้อมูลจริงแต่เรียงลำดับผิด. ปัญหาการป้อนข้อมูลผิดพลาดอย่างอื่นเช่นการป้อนข้อมูลซ้ำ. คำว่า “ที่” ถ้ามีไม้เอกสองตัวก็ยังเห็นเป็นคำว่า “ที่” เหมือนมีไม้เอกตัวเดียว.

กลวิธีการป้อนข้อมูลหรือในทางเทคนิคเรียกว่า *Input Method (IM)* เป็นกลวิธีในระบบ X วินโดว์ที่ใช้รับข้อมูลที่ป้อนเข้ามาทางแป้นพิมพ์ส่งต่อให้โปรแกรมหรือเซิร์ฟเวอร์ประมวลผลก่อนที่จะแปลงเป็นข้อมูลใช้งานจริง. การป้อนข้อมูลในลักษณะมักจะใช้ในการป้อนข้อมูลภาษาที่ซับซ้อนและไม่สามารถป้อนข้อมูลโดยตรงจากแป้นพิมพ์เช่น ภาษาญี่ปุ่น, ภาษาจีน และภาษาเกาหลี. กลวิธีของ IM เองสามารถแบ่งได้เป็น 2 แบบคือ

- XIM (X Input Method) —— เป็นกลวิธีการป้อนข้อมูลดั้งเดิมที่ใช้ในระบบ X วินโดว์และยังใช้กันในปัจจุบันได้. ไลบรารทูลคิด GUI ทั่วไปรับรู้การป้อนข้อมูลแบบนี้. การใช้งานทำได้โดยการตั้งค่าตัวแปรสภาพแวดล้อม `XMODIFIERS`.

- IMmodule —— เป็นกลวิธีการป้อนข้อมูลสมัยใหม่ในเชิงโมดูลใช้กับทูลคิดสมัยใหม่เช่น GTK+. สามารถเลือกใช้งานได้โดยการตั้งค่าตัวแปรสภาพแวดล้อม GTK_IM_MODULE หรือตั้งค่าในแอพพลิเคชันเฉพาะราย. การใช้งานภาษาไทยแบบ IMmodule ยังไม่สมบูรณ์.

7.4.1 X Input Method

XIM (X Input Method) ช่วยการป้อนข้อมูลภาษาไทยให้ถูกต้องโดยกรองข้อมูลที่ได้จากแป้นพิมพ์, ตรวจสอบประเภทของอักขระว่าสามารถเป็นข้อมูลนำเข้าได้หรือไม่ก่อนที่จะส่งข้อมูลให้โปรแกรมต่อไป. ก่อนใช้ XIM กับภาษาไทย, ผู้ใช้ต้องตั้งแค่โลแคลของตัวแปรสภาพแวดล้อม LC_CTYPE ให้เป็นโลแคลภาษาไทยด้วยเพราะมีการตรวจสอบประเภทของอักขระที่ป้อนข้อมูลเข้าจากแป้นพิมพ์. ตามที่ได้แนะนำไปแล้วว่าค่าของ LC_CTYPE สามารถตั้งค่าทับด้วยตัวแปรสภาพแวดล้อม LANG หรือ LC_ALL ก็ได้.

การใช้ XIM ทำได้โดยการตั้งค่าตัวแปรสภาพแวดล้อม XMODIFIERS โดยมีรูปแบบดังต่อไปนี้.

```
export XMODIFIERS="@im=method"
```

ผู้ใช้จะระบุ *method* ซึ่งเป็นชื่อวิธีการใช้ XIM แบบต่างๆ. สำหรับการใช้ XIM ภาษาไทยมีค่าที่ระบุได้ดังต่อไปนี้.

- Passthrough —— XIM จะไม่ตรวจสอบอักขระที่ป้อนทางแป้นพิมพ์, จะส่งอักขระที่ได้รับจากแป้นพิมพ์ให้โปรแกรมต่อไป. ผู้ใช้สามารถป้อนข้อมูลที่ไม่เหมาะสมได้เช่นพิมพ์วรรณยุกต์โดยไม่มีพยัญชนะหรือสระที่เหมาะสมนำหน้าเป็นต้น. การตั้งค่าตัวแปร XMODIFIERS="@im=none" ให้ผลเช่นเดียวกับ Passthrough.

- BasicCheck —— XIM จะตรวจสอบลำดับการป้อนข้อมูลเบื้องต้นว่าเหมาะสมหรือไม่. ตัวอย่างเช่น

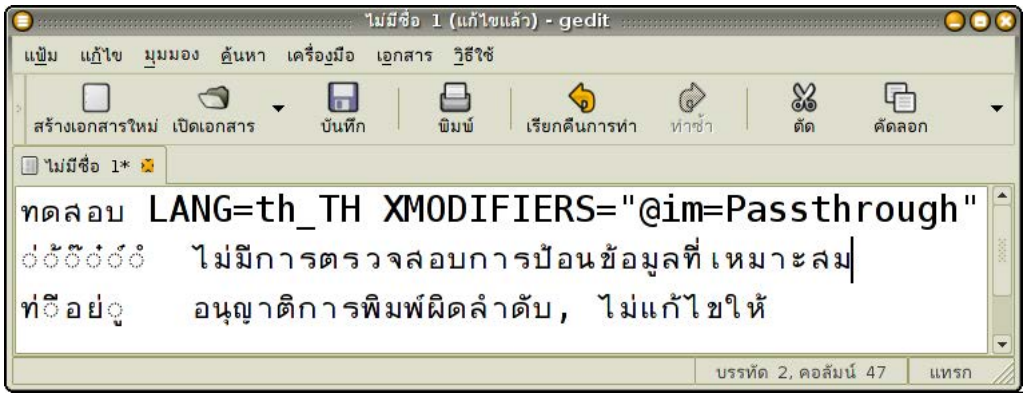
  – ปฏิเสธการป้อนข้อมูลที่ไม่เหมาะสม. เช่นไม่สามารถพิมพ์ไม่เอกตามหลังสระอา เป็นต้น.

  – แก้ไขลำดับอักขระถ้าเป็นไปได้. เช่นถ้าพิมพ์ ท ◌ี ◌่ ก็จะแปลงลำดับอักขระให้ถูกต้องเป็น ท ◌่ ◌ี โดยอัตโนมัติ.

- Strict —— เพิ่มกฎที่ช่วยให้การพิมพ์ภาษาไทยถูกต้องยิ่งขึ้นเช่น XIM แบบ BasicCheck สามารถพิมพ์อักขระที่ไม่มีความหมายเช่น “กๆ” แต่ใน XIM แบบ Strict จะสามารถพิมพ์ “ๆ” ตามหลัง “ฤ” หรือ “ฦ” ได้เท่านั้น.

รูปที่ 7.17 แสดงตัวอย่างโปรแกรม gedit ที่รันด้วยคำสั่ง LANG=th_TH XMODIFIERS="@im=Passthrough" gedit จะเห็นได้ว่าผู้ใช้สามารถพิมพ์ผิดลำดับได้เช่น ท ◌่ ◌ี. ผู้ใช้สามารถสังเกตเห็นสิ่งปรกติได้เพราะไลบรารี Pango ช่วย

เติมเครื่องหมาย dotted circle บ่งบอกว่าหน้าสระอีต้องการอักขระที่เหมาะสม. ถ้าใช้ XIM แบบ BasicCheck หรือ Strict ตัวโปรแกรมจะแก้ลำดับให้ถูกต้องโดยอัตโนมัติ.

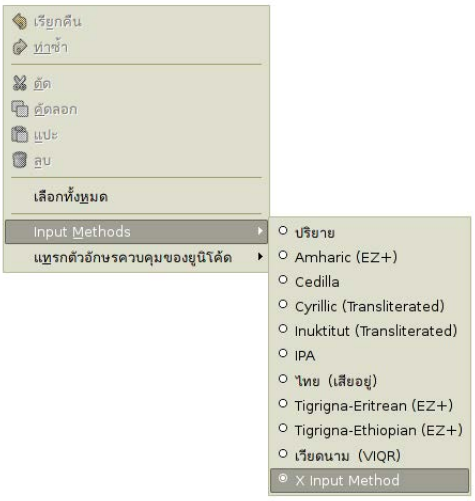


รูปที่ 7.17: โปรแกรม gedit ใช้ XIM แบบ Passthrough.

ในสภาพแวดล้อมโลแคลภาษาไทยถ้าไม่มีการตั้งค่า XMODIFIERS จะถือว่าตัวแปรสภาพแวดล้อมนี้ใช้ XIM แบบ BasicCheck โดยปริยาย.

7.4.2 IM module

การป้อมข้อมูลแบบ IM module ใช้ในแอพพลิเคชันที่ใช้ไลบรารี GTK+ เช่นโปรแกรมต่าง ๆในสภาพแวดล้อมเดสก์ท็อป GNOME. การเลือกโมดูลสามารถทำได้โดยการตั้งค่าตัวแปรสภาพแวดล้อม GTK_IM_MODULE หรือเลือกจากแอพพลิเคชันเช่นในรูปที่ 7.18.



รูปที่ 7.18: การเลือก IM module ในโปรแกรม gedit.

โมดูลการป้อนข้อมูลภาษาไทยยังไม่สมบูรณ์และไม่สามารถใช้ได้. ซึ่งในขณะที่เขียนหนังสือนี้คุณเทพพิทักษ์จาก Thai Linux Working Group พยายามจะแก้ไขประสานงาน

กับนักพัฒนาของ GNOME อยู่. อย่างไรก็ตาม IM module มีโมดูลสำหรับเชื่อมต่อใช้ XIM ตามปรกติ. ถ้าเลือก X Input Method จากเมนูก็จะเหมือนการใช้ XIM ตามปรกติ.

7.5 สรุปท้ายบท

- รหัสอักขระที่เป็นมาตรฐานและเกี่ยวข้องกับภาษาไทยได้แก่ TIS-620, ISO8859-11 และ ยูนิโค้ด.
- การเข้ารหัสอักขระมีหลายวิธี. การเข้ารหัสแบบ TIS-620 เป็นการใช้ค่ารหัสอักขระตรง ๆในการเข้ารหัสอักขระ, ทำให้ซ้ำกับการเข้ารหัสอักขระภาษาอื่น ๆเช่น ISO8859-1.
- การเข้ารหัสแบบยูนิโค้ดเช่น UCS-2, UCS-4, UTF-8 ฯลฯ มีข้อดีที่รหัสอักขระของภาษาต่าง ๆไม่ซ้ำกัน.
- โลแคลเป็นกลวิธีที่ใช้ปรับแต่งสภาพแวดล้อมการใช้งานโปรแกรมต่าง ๆให้เป็นภาษาและวัฒนธรรมที่ต้องการ. โลแคลภาษาไทยที่ใช้กันทั่วไปได้แก่ `th_TH.TIS-620` และ `th_TH.UTF-8`.
- การติดปรับแต่งค่าโลแคลจะใช้ตัวแปรสภาพแวดล้อมที่ขึ้นต้นด้วย `LC_`, `LC_ALL` หรือ `LANG`.
- Input Method หรือ IM เป็นวิธีที่จัดการข้อมูลอักขระที่ได้รับจากแป้นพิมพ์, ตรวจสอบข้อมูล, ประมวลผลแล้วส่งต่อให้โปรแกรมต่อไป.
- IM ที่เกี่ยวข้องกับการป้อนข้อมูลภาษาไทยในปัจจุบันคือ XIM (X Input Method) สามารถเรียงลำดับอักขระที่ป้อนเข้ามาจากแป้นพิมพ์ให้ถูกต้อง, หรือปฏิเสธการป้อนข้อมูลที่ไม่เหมาะสม.
- วิธีการใช้ XIM ทำได้โดยการตั้งค่าตัวแปรสภาพแวดล้อม `XMODIFIERS`.

บทที่ 8
แนะนำโปรแกรมใช้งาน

ปัจจุบันสภาพแวดล้อมเดสก์ท็อปเช่น GNOME, KDE พัฒนาไปอย่างรวดเร็ว, โปรแกรมต่าง ๆใช้งานง่ายขึ้น, มีอินเทอร์เฟสที่ทันสมัย. การใช้งานโปรแกรมบนลินุกซ์ไม่ได้จำกัดเฉพาะงานเดสก์ท็อป, โปรแกรมที่มีอินเฟสเป็นบรรทัดคำสั่งยังเป็นที่นิยมเพราะทำงานได้รวดเร็วและสามารถเขียนสคริปต์ให้ทำงานตามโดยที่ไม่ต้องโต้ตอบกับผู้ใช้ได้. อย่างไรก็ตามโปรแกรมต่างสามารถใช้ร่วมกันได้เช่นในสภาพแวดล้อมเดสก์ท็อป GNOME สามารถใช้โปรแกรมของ KDE ได้หรือใช้โปรแกรมที่ไม่ได้ใช้ไลบรารี GTK+ ได้เช่นกัน. สิ่งที่สำคัญคือการรู้จักเลือกใช้โปรแกรมให้เหมาะสมกับงาน, สภาพแวดล้อม และตามความถนัดของผู้ใช้.

โปรแกรมหรือแอพพลิเคชันต่าง ๆที่ใช้ในลินุกซ์ส่วนใหญ่จะเป็นโปรแกรมซอฟต์แวร์เสรี. ตอนที่ติดตั้งลินุกซ์ดิสทริบิวชันผู้ใช้สามารถเลือกแอพพลิเคชันที่ใช้ติดตั้งในระบบหรือติดตั้งหลังติดตั้งตัวระบบปฏิบัติการ. วิธีการติดตั้งโปรแกรมจะอธิบายในบทที่เกี่ยวกับการดูแลระบบหน้า ??. ในบทนี้จะแนะนำโปรแกรมต่าง ๆตามประเภทการใช้งานเพื่อให้เห็นภาพพจน์และเป็นประโยชน์ในการใช้งานจริง. ในช่วงแรกเป็นเนื้อหาเกี่ยวกับบรรณาธิกรณ์ซึ่งจะอธิบายละเอียดกว่าโปรแกรมอื่น ๆ. ส่วนแอพพลิเคชันแบบ GUI จะไม่เจาะลึกในรายละเอียดเนื่องจากผู้ใช้สามารถเรียนรู้จากการใช้แอพพลิเคชันเหล่านั้นด้วยตัวเองหรืออ่านจากหนังสือเล่มอื่น ๆ.

8.1 บรรณาธิกรณ์

บรรณาธิกรณ์ (editor) เป็นชื่อทั่วไปที่ใช้เรียกโปรแกรมสำหรับสร้างไฟล์หรือแก้ไขข้อมูลที่มีอยู่. บรรณาธิกรณ์ต่างจากโปรแกรมเวิร์ดโปรเซสเซอร์เช่น OpenOffice ตรงที่บรรณาธิกรณ์เน้นการประมวลผลข้อมูลเท็กซ์เช่นแก้ไขข้อมูล, และไม่มีความสามารถในการจัดแต่งหน้าเอกสารสำหรับงานพิมพ์. ในระบบจำเป็นต้องมีบรรณาธิกรณ์เพื่อแก้ไขไฟล์ต่าง ๆ, ปรับแต่งระบบ. ผู้ใช้ทั่วไปโดยเฉพาะผู้ดูและระบบควรรู้จักวิธีใช้บรรณาธิกรณ์ที่ทำงานในเทอร์มินอลอย่างน้อยหนึ่งตัว.

แก้ไข (edit) ในที่นี้หมายถึงการพิมพ์ข้อมูลเพิ่ม, ลบแก้ไขข้อมูล.

บรรณาธิกรณ์ที่ใช้กันในปัจจุบันสามารถแบ่งออกเป็นสองกระแสตามความนิยมได้แก่ `vi` และ `emacs`. นอกจากนี้ยังมีบรรณาธิกรณ์อื่น ๆเช่น `nano`, `gedit`, `kedit` ฯลฯ.

vi และ emacs เป็นบรรณาธิกรณ์ที่ได้รับความนิยมสูงเพราะเป็นโปรแกรมที่มีมานานแล้ว, มีความสามารถต่าง ๆมากมาย, ใช้ได้กับงานทั่วไปจนถึงเป็นบรรณาธิกรณ์สำหรับเขียนโปรแกรมพัฒนาซอฟต์แวร์.

8.1.1 vi(m)

บรรณาธิกรณ์ในยูนิกซ์ยุคแรก ๆคือ ed เป็นบรรณาธิกรณ์ที่ประมวลผลข้อมูลเท็กซ์ในไฟล์บรรทัดต่อบรรทัด, ไม่สามารถแสดงข้อมูลต่อเนื่องเป็นหน้าบนเทอร์มินอล. โปรแกรม ed เป็นบรรณาธิกรณ์ที่ใช้ยากและในเวลาต่อมา George Coulouris[51] สร้างบรรณาธิกรณ์ชื่อ em คล้ายกับ ed แต่ใช้งานง่ายขึ้น. ขณะที่ Bill Joy พัฒนา BSD ได้พบกับ George Coulouris และได้เอาบรรณธิกรณ์ em เป็นต้นแบบในการสร้างบรรณาธิกรณ์ตัวใหม่ที่ดีกว่าได้แก่ ex. อย่างไรก็ตาม ex ก็ยังคงเป็นบรรณาธิกรณ์เชิงบรรทัด (line editor) แบบเดิมจนกระทั่ง Bill Joy เขียนบรรณาธิกรณ์ใหม่สามารถแสดงผลเต็มหน้าจอได้ที่เรียกว่า vi. vi รับแนวคิดของบรรณาธิกรณ์ก่อนหน้านั้นเช่น ed มาไม่น้อย. จะเห็นได้จากคำสั่งที่ใช้ใน vi เหมือนกับคำสั่งที่ใช้ใน ed.

vi อ่านออกเสียงว่า vee-eye

ปัจจุบัน vi ที่ใช้กันในลินุกซ์ไม่ใช่โปรแกรม vi แบบดั้งเดิมแต่เป็นโปรแกรม vi โคลนคือเป็นบรรณาธิกรณ์ที่สร้างให้ทำงานเหมือนกับ vi และได้รับการปรับปรุงพัฒนาให้ดีขึ้นจนความสามารถบางอย่างแตกต่างจากของเดิมโดยสิ้นเชิง. โปรแกรม vi ในปัจจุบันคือ vim (Vi IMproved). ในดิสโทรทั่วไปจะสร้างซอฟต์ลิงก์ vi ชี้ไปที่ตัวโปรแกรม vim. โดยปรกติ vim เป็นบรรณาธิกรณ์ที่ใช้อยู่ในเทอร์มินอลเช่นเดียวกับ vi แบบดั้งเดิม. แต่ตอนที่สร้าง vim จากรหัสต้นฉบับสามารถเลือกสร้าง vim ให้แสดงผลแบบ GUI ได้และจะมีโปรแกรมต่างหากเตรียมไว้ชื่อ gvim. gvim จะมีหน้าต่างเป็นของตัวเองใช้ในระบบ X วินโดว์. อย่างไรก็ตามคนส่วนใหญ่ก็ยังนิยมให้ vi ซึ่งรันอยู่ในเทอร์มินอลตามปรกติ.

โปรแกรม vi ที่ใช้ในหนังสือเล่มนี้จะหมายถึง vim.

นอกจากนี้ยังมีโปรแกรมที่อยู่ในตระกูลเดียวกันได้แก่ view คือ vi แบบพิเศษสำหรับเปิดอ่านไฟล์ได้อย่างเดียว, ไม่สามารถแก้ไขไฟล์. ex เป็นบรรณาธิกรณ์ประมวลผลข้อมูลบรรทัดต่อบรรทัดคล้าย ed. โปรแกรมเหล่านี้จริง ๆแล้วก็คือ vim.

การใช้โปรแกรม vim มักจะสั่งคำสั่ง vi หรือ vim จากบรรทัดคำสั่ง.

```
vi [options] [filename]
```

ตัวเลือก	คำอธิบาย
-e	รันโปรแกรมโดยให้ผลเหมือนกับ ex.
-g	รันโปรแกรมโดยใช้อินเทอร์เฟสแบบ GUI. ให้ผลเหมือนกับคำสั่ง gvim.
-R	ไฟล์ที่เปิดด้วย vi และตัวเลือกนี้จะเป็นไฟล์แบบอ่านได้อย่างเดียว, ไม่สามารถแก้ไขได้. ให้ผลเหมือนกับคำสั่ง view.
-C	ระบุให้ vim ทำงานเหมือนกับ vi แบบดั้งเดิมเท่าที่เป็นไปได้.

ไฟล์ที่ระบุเป็นอาร์กิวเมนต์ของคำสั่งเป็นไฟล์ที่ต้องการแก้ไขหรือชื่อไฟล์ใหม่ที่ต้องการสร้าง. ถ้าไม่ระบุชื่อไฟล์และต้องการบันทึกสิ่งที่พิมพ์ต้องระบุชื่อไฟล์ภายหลัง.

โหมด

ก่อนที่จะใช้ `vi` ต้องทำความรู้จักกับตัวโปรแกรมก่อนว่า `vi` มีแนวคิดเรื่องโหมด (mode). โหมดคือสภาพการทำงานแบ่งออกเป็น 2 แบบได้แก่

- โหมดคำสั่ง (command mode) —— หลังจากที่ `vi` เริ่มทำงานจะอยู่ในโหมดคำสั่ง. ขณะที่อยู่ในโหมดคำสั่ง, การกดปุ่มแป้นพิมพ์ถือเป็นการสั่งคำสั่งให้ `vi` กระทำการอะไรบางอย่าง. ปุ่มบางตัวกดแล้วจะมีการโต้ตอบทันทีเช่นการเลื่อนเคอร์เซอร์. คำสั่งบางอย่างต้องพิมพ์เครื่องหมาย : เป็นการเริ่มคำสั่งบอกโปรแกรมรับรู้ว่าคำสั่งนี้เป็นคำสั่งยาวและต้องกด `Enter` ตามจึงจะกระทำการ.
- โหมดแก้ไข (edit mode) —— เวลาที่ต้องการพิมพ์ข้อความต้องเปลี่ยนโหมดให้อยู่ในโหมดแก้ไขก่อนจึงจะพิมพ์ข้อความได้. คำสั่ง (คีย์) ต่าง ๆที่แสดงในตารางที่ 8.1 เป็นวิธีการเปลี่ยนโหมดจากโหมดคำสั่งให้เป็นโหมดแก้ไข. ในทางกลับกันจะใช้การกดปุ่ม `Esc` เพื่อเปลี่ยนโหมดจากโหมดแก้ไขให้กลับเข้าสู่โหมดคำสั่ง.

ตารางที่ 8.1: คำสั่งสำหรับแก้ไขข้อความใน `vi`.

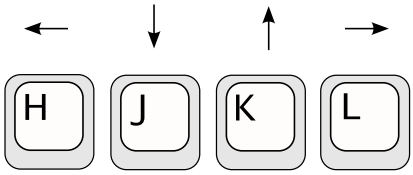
คีย์คำสั่ง	คำอธิบาย
`a`	append —— เริ่มแก้ไขข้อมูลที่ตำแหน่งถัดจากเคอร์เซอร์.
`i`	insert —— เริ่มแก้ไขข้อมูลที่ตำแหน่งเคอร์เซอร์ปัจจุบัน.
`o`	open line —— เปิดบรรทัดใหม่ถัดจากบรรทัดปัจจุบันเริ่มการแก้ไข.
`r`	replace character —— แก้ไขอักขระตำแหน่งเคอร์เซอร์ปัจจุบัน. หลังจากที่เปลี่ยนอักขระแล้วจะกลับเป็นโหมดคำสั่งทันที.
`R`	replace —— เริ่มการแก้ไขจากตำแหน่งเคอร์เซอร์ปัจจุบันและเขียนทับอักขระที่มีอยู่.

การเลื่อนเคอร์เซอร์

การเลื่อนเคอร์เซอร์ใน `vi` ทำได้โดยการกดคีย์ลูกศรต่าง ๆขณะที่อยู่ในโหมดแก้ไขหรือโหมดคำสั่ง. ในโหมดคำสั่งสามารถใช้คีย์ `h j k l` แทนคีย์ลูกศรเลื่อนเคอร์เซอร์ไปทางซ้าย, บรรทัดถัดไป, บรรทัดก่อนหน้า และทางขวาได้ด้วย. ผู้ที่ใช้ `vi` คล่องแล้วจะถนัดใช้คีย์ `hjkl` มากกว่าคีย์ลูกศรเพราะโดยปรกติตำแหน่งของมือขวาที่วางบนแป้นพิมพ์จะตรงกับ `hjkl`.

โปรแกรม `vi` มีคำสั่งสำหรับเลื่อนเคอร์ไปที่ต้นบรรทัดด้วยคีย์ `^` และเลื่อนคำสั่งไปที่ท้ายบรรทัดด้วยคีย์ `$`. อักขระที่ใช้เป็นคำสั่งนี้เหมือนกับอักขระที่ใช้ใน regular expression.

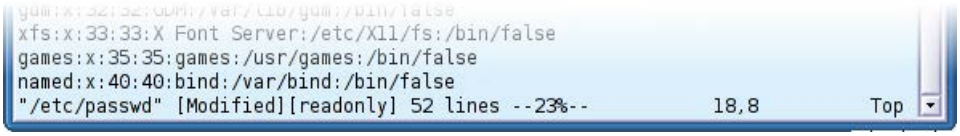
การเลื่อนเนื้อหาทั้งหน้าจอลงจะใช้คีย์ `Ctrl+f` (forward). ในทางกลับกันจะใช้คีย์



รูปที่ 8.1: ตำแหน่งของนิ้วเวลาใช้เลื่อนเคอร์เซอร์.

Ctrl+b (backward) เพื่อเลื่อนหน้าจอขึ้น. ในกรณีที่ต้องการตรวจสอบว่าตำแหน่งของเคอร์เซอร์อยู่ที่ส่วนไหนของไฟล์ให้กดคีย์ Ctrl+g. vi จะแสดงชื่อไฟล์, ข้อมูลของไฟล์ที่แก้ไขและตำแหน่งบรรทัดและคอลัมน์ของเคอร์เซอร์ที่บรรทัดสุดท้ายของเทอร์มินอล.

การย้ายเคอร์เซอร์ไปบรรทัดที่ต้องการให้กดเลขบรรทัดแล้วตามด้วย G (กด Shift+g) เช่นถ้าต้องการกระโดดไปบรรทัดที่สามให้กดคีย์ 3G. ถ้ากด G โดยไม่มีตัวเลขนำหน้าจะเป็นการกระโดดไปที่บรรทัดสุดท้ายของไฟล์.



รูปที่ 8.2: การแสดงตำแหน่งของเคอร์เซอร์ใน vi.

ลบอักขระหรือข้อความ

ขณะที่อยู่ในโหมดแก้ไขสามารถใช้คีย์ Backspace หรือ Del เพื่อลบอักขระที่ไม่ต้องการได้. Backspace จะลบอักขระที่อยู่ก่อนหน้าเคอร์เซอร์, Del จะลบอักขระที่อยู่ตรงตำแหน่งเคอร์เซอร์.

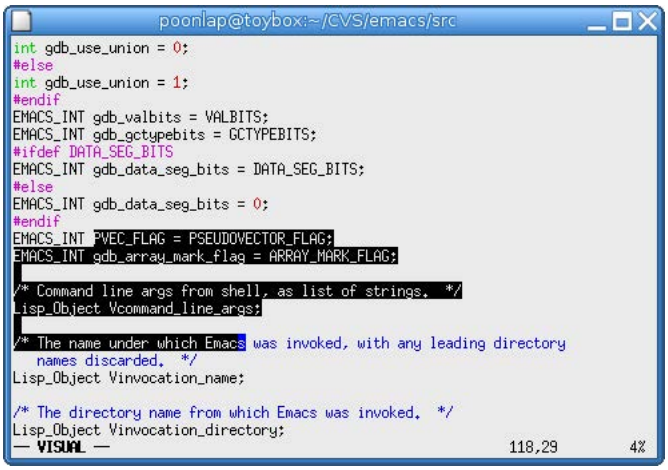
ถ้าอยู่ในโหมดคำสั่งสามารถใช้คำสั่งในตารางที่ 8.2. ถ้าต้องการยกเลิกสิ่งที่กระทำไปให้ใช้กดคีย์ u (undo).

อักขระที่ลบด้วยคำสั่งในตารางข้างบนจะถูกบันทึกไว้ในพื้นที่ชั่วคราว. หลังจากนั้นผู้ใช้สามารถเลื่อนเคอร์เซอร์ไปตำแหน่งที่ต้องการและเรียกอักขระที่อยู่ในพื้นที่ชั่วคราวกลับมาวางในตำแหน่งเคอร์เซอร์ได้ด้วยคีย์ p.

การลบอักขระเหล่าด้วยคำสั่งเหล่านี้สามารถระบุจำนวนที่ต้องการลบโดยพิมพ์จำนวนที่ต้องการลบแล้วตามด้วยคำสั่ง. การระบุจำนวนด้วยตัวเลขมีประโยชน์เมื่อต้องการลบข้อมูลในคราวเดียวเช่น 100dd จะลบข้อมูล 100 บรรทัด, 4x จะลบอักขระ 4 ตัวเป็นต้น. ในทำนองเดียวกันถ้าต้องการใช้เคอร์เซอร์เลือกช่วงที่ต้องการลบ, จะใช้ต้องเปลี่ยนเป็นโหมด VISUAL โดยกดคีย์ v. โปรแกรม vi จะไฮไลต์ส่วนที่เลือก, หลังจากนั้นผู้ใช้สามารถสั่งคำสั่งลบช่วงที่เลือกไว้ได้. ถ้าต้องการยกเลิกโหมด VISUAL ให้กดคีย์ v อีกครั้ง.

ตารางที่ 8.2: คำสั่งสำหรับลบอักขระใน vi.

คีย์คำสั่ง	คำอธิบาย
x	ให้ผลเหมือนกับการกดคีย์ Del ในโหมดแก้ไข. ลบอักขระที่อยู่ในตำแหน่งเคอร์เซอร์.
dd	ลบบรรทัดที่เคอร์เซอร์อยู่ทั้งบรรทัด.
dw	ลบคำ. คำในที่นี้หมายถึงคำภาษาอังกฤษ. เช่นถ้ามีสายอักขระ “How are you?” และเคอร์เซอร์อยู่ในตัวอักษร a, กด dw จะลบ are.
D	ลบอักขระตั้งแต่ตำแหน่งเคอร์เซอร์จนจบบรรทัด.



รูปที่ 8.3: โหมด VISUAL เลือกช่วงที่ต้องการกระทำการ.

เชื่อมต่อบรรทัด

ในบางกรณีเรามีความจำเป็นต้องรวมบรรทัดสองบรรทัดเข้าเป็นบรรทัดเดียวกัน. ในโหมดคำสั่งสามารถใช้คีย์ J (join) รวมบรรทัดที่อยู่จากบรรทัดปัจจุบันต่อท้ายรวมกันเป็นบรรทัดเดียวได้. การใช้คำสั่งนี้จะเร็วกว่าการแก้ไขข้อมูลในโหมดแก้ไข.

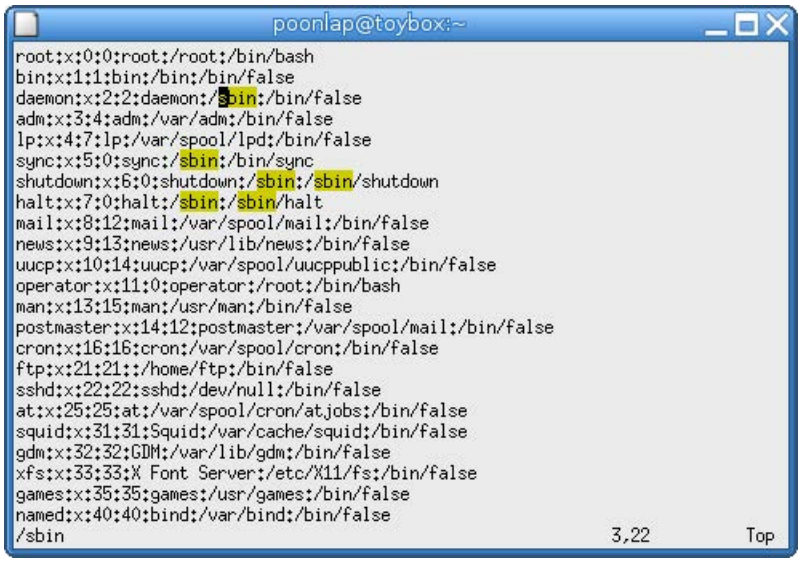
ในทำนองเดียวกัน, ในกรณีที่ต้องการรวมข้อมูลจากไฟล์อื่นเข้ามาในไฟล์ที่แก้ไขอยู่ให้ใช้คำสั่ง :r (กด : แล้วกด r). ต่อจากนั้นให้พิมพ์ชื่อไฟล์ที่ต้องการดึงข้อมูลเข้ามาไว้ด้วยกัน. ในขณะที่พิมพ์ชื่อไฟล์สามารถใช้คีย์แท็บช่วยเติมเต็มชื่อไฟล์ได้ด้วย, เหมือนความสามารถของ bash เชลล์.

ค้นหาคำ

การค้นข้อมูลในไฟล์เป็นงานที่เกิดขึ้นบ่อยครั้ง. ในโปรแกรม vi ผู้ใช้สามารถค้นหาคำด้วย regular expression โดยกดคีย์ / (search forward) ขณะที่อยู่ในโหมดคำสั่งและพิมพ์คำหรือ regular expression ที่ต้องการหาแล้วกด Enter. เคอร์เซอร์จะเลื่อนไปอยู่

รูปที่ 8.4: พรอมต์รอรับคำสั่งใน vi.

ตำแหน่งที่คำค้นหาเจอ. และจะไฮไลต์คำที่หาเจอด้วยถ้าปรับแต่ง vi ไว้. กด n (next match) เพื่อเลื่อนเคอร์เซอร์ไปตำแหน่งที่หาคำนั้นเจอต่อไปเรื่อย ๆ. ในทางกลับกันคำสั่ง ? ใช้หาคำที่อยู่ก่อนหน้าเคอร์เซอร์และกด n เพื่อดูคำที่หาเจอไปเรื่อย ๆ.



รูปที่ 8.5: การหาคำใน vi.

ข้อควรระวังในการหาคำคือการหาคำจะใช้ regular expression. ดังนั้นจะต้องเพิ่มเครื่องหมาย \ หน้าอักขระที่มีความหมายพิเศษสำหรับ regular expression ทุกครั้ง.

บันทึก, อ่านไฟล์, และจบการทำงาน

คำสั่งของ vi ที่เกี่ยวข้องกับการบันทึกและจบการทำงานที่สำคัญ ๆสรุปไว้ในตารางที่ 8.3.

ตารางที่ 8.3: คำสั่งสำหรับจัดการไฟล์ใน vi.

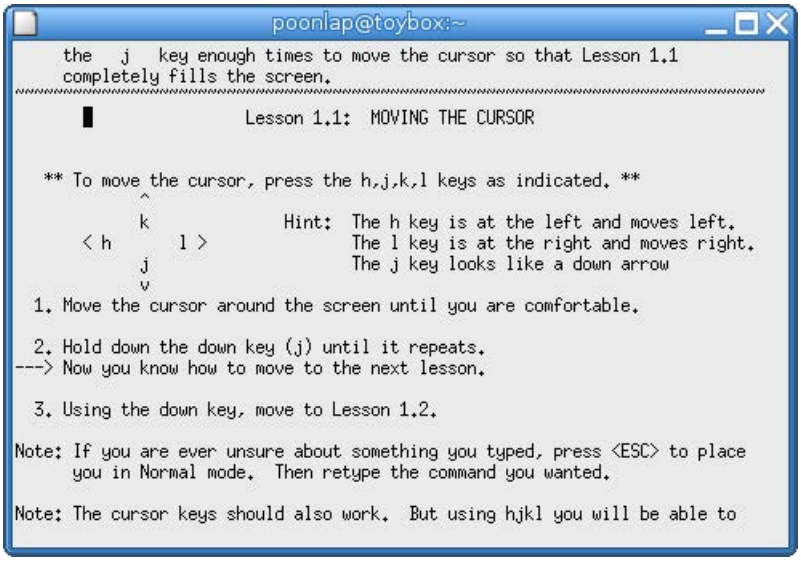
คีย์คำสั่ง	คำอธิบาย
:w [*filename*]	บันทึกข้อมูลที่แก้ไขลงในไฟล์. ถ้าระบุชื่อไฟล์ด้วยจะเป็นการบันทึกข้อมูลลงในไฟล์ที่ระบุ.
:q	quit —— จบการทำงานโดยที่ไม่บันทึกสิ่งที่แก้ไข.

`:q!`	บังคับจบการทำงานโดยที่ไม่บันทึกสิ่งที่แก้ไข.
`ZQ`	ให้ผลเช่นเดียวกับ `:q!`.
`:wq`	บันทึกข้อมูลที่แก้ไขลงในไฟล์แล้วจบการทำงาน.
`ZZ`	ให้ผลเหมือน `wq` แต่จะบันทึกข้อมูลเมื่อมีการเปลี่ยนแปลงเกิดขึ้น.
`:wq!`	บังคับบันทึกข้อมูลที่แก้ไขลงในไฟล์แล้วจบการทำงาน.
`:e` *filename*	เปิดแก้ไขไฟล์ที่ระบุ.

ขอความช่วยเหลือ

ในโปรแกรม `vi` มีคำสั่งสำหรับแสดงเอกสารช่วยเหลือแนะนำการใช้คำสั่งเบื้องต้นได้แก่ `:help`. หรือระบุหัวข้อขอความช่วยเหลือ "`:help [`*subject*`]`". *subject* คือชื่อหัวข้อสำหรับการช่วยเหลือหรือคำสั่งที่ต้องการดูรายละเอียดการใช้งาน. ตัวอย่างคำสั่งเช่น "`:help set`" จะแสดงข้อมูลเกี่ยวกับการใช้คำสั่ง `set`.

นอกจากข้อมูลช่วยเหลือในตัวโปรแกรมยังมีโปรแกรม `vimtutor` ช่วยสอนการใช้ `vi` เบื้องต้น.



รูปที่ 8.6: โปรแกรม `vitutor`.

ปรับแต่ง `vi`

`vim` เป็นบรรณาธิกรณ์ที่สามารถปรับแต่งความสามารถต่าง ๆช่วยอำนวยความสะดวกของผู้ใช้. การปรับแต่งเบื้องต้นสามารถทำได้โดยการสั่งคำสั่ง `:set` และตามด้วยชื่อตัวเลือกต่าง ๆ. ชื่อตัวเลือกนี้อาจจะมีความหมายในตัวและไม่ต้องการอาร์กิวเมนต์เช่น "`:set number`" เป็นการสั้งให้ `vim` แสดงหมายเลขบรรทัดบริเวณซ้ายมือของหน้าจอ. ตัวเลือกบางตัวต้องการอาร์กิวเมนต์ได้แก่ค่าที่เป็นตัวเลขหรือสายอักขระเช่น "`:set`

columns=80” เป็นการตั้งจำนวนคอลัมน์ที่แสดงผลในหน้าหนึ่งให้มีขนาด 80 ตัวอักษร. การใช้คำสั่ง :set มีรูปแบบต่าง ๆดังต่อไปนี้.

- :set —— แสดงชื่อและค่าตัวเลือกที่แตกต่างจากค่าโดยปริยาย.
- :set all —— แสดงชื่อและค่าตัวเลือกทั้งหมด.
- :set *option*? —— ตรวจสอบค่าของตัวเลือกว่ามีค่าเป็นอะไรหรือมีการตั้งตัวเลือกที่ระบุหรือไม่.
- :set [no]*option* —— ใช้สำหรับระบุใช้ตัวเลือกที่ไม่ต้องการอาร์กิวเมนต์. ตัวเลือกพวกนี้สามารถพิมพ์คำว่า no นำหน้าเป็นการระบุยกเลิกตัวเลือกที่ตั้งไปแล้ว. ตัวอย่างเช่น “:set number”, “:set nonumber” เป็นต้น.
- :set *option=value* —— ตั้งค่าให้ตัวเลือกที่ต้องการอาร์กิวเมนต์เช่น “:set columns=80”.

ตารางที่ 8.4 แสดงชื่อตัวเลือกที่ใช้บ่อยและมีประโยชน์. บางกรณีชื่อตัวเลือกจะมีชื่อย่อและชื่อเต็ม.

ตารางที่ 8.4: ตัวเลือกสำหรับการปรับแต่ง vi.

คำสั่ง	คำอธิบาย
[no]autoindent	ย่อหน้าบรรทัดใหม่ให้เหมาะสมโดยอัตโนมัติสำหรับการใช้ vi เขียนโปรแกรม. ตัวเลือกนี้มีชื่อย่อคือ [no]ai.
[no]compatible	ปรับแต่งให้ vim ทำงานเหมือน vi แบบดั้งเดิมเท่าที่เป็นไปได้. ถ้าต้องการใช้ vim เต็มความสามารถควรใช้ตัวเลือก “:set nocompatible”. สิ่งที่แตกต่างระหว่าง vi และ vim ดูได้จากคำสั่ง “:help vi_diff”.
[no]hlsearch	ไฮไลต์คำที่หาเจอเมื่อสั่งคำสั่งค้นหา. ตัวเลือกนี้มีชื่อย่อคือ [no]hls.
[no]incsearch	ค้นหาคำทันทีขณะพิมพ์คำที่ต้องการค้นหา. ตัวเลือกนี้มีชื่อย่อคือ [no]is
[no]ruler	แสดงเลขบรรทัดและคอลัมน์ด้านล่างของหน้าจอ.
[no]showmatch	เลื่อนเคอร์เซอร์จับคู่วงเล็บแบบต่าง ๆเมื่อทำการปิดวงเล็บช่วยให้ผู้ใช้รับรู้ว่าบริเวณข้อมูลที่อยู่ในวงเล็บ. ตัวเลือกนี้มีชื่อย่อคือ [no]sm.
[no]visualbell	กระพริบหน้าจอแทนการใช้เสียงเมื่อเกิดข้อผิดพลาด. ตัวเลือกนี้มีชื่อย่อคือ [no]vb
[no]wrap	แสดงผลในบรรทัดถัดไปถ้าสิ่งที่พิมพ์ยาวเกินความกว้างที่ตั้งค่าไว้.

mouse=a	ระบุให้ใช้เมาส์ได้ในทุกโหมด. vi ที่รันอยู่ในเทอร์มินอลโดยปรกติจะไม่สามารถใช้เมาส์ได้.

ผู้ใช้ไม่จำเป็นต้องจำชื่อคำสั่งหรือตัวเลือกเหล่านี้ทั้งหมด. โปรแกรม vi จะเติมเต็มชื่อคำสั่งหรือตัวเลือกให้โดยอัตโนมัติถ้ากดคีย์ Tab ตามหลังเครื่องหมาย :. ผู้ใช้สามารถกดแท็บต่อกันหลายครั้งเพื่อเปลี่ยนชื่อคำสั่งที่ vi ช่วยเติมเต็มให้.

vim จะอ่านไฟล์ตั้งค่าเริ่มต้นจากไฟล์ ~/.vimrc. เนื้อหาของไฟล์คือคำสั่งที่ใช้ใน vi แต่ตัดเครื่องหมาย : ทิ้ง. ผู้ใช้สามารถสร้างไฟล์ ~/.vimrc หลังจากที่ปรับแต่งแล้วด้วยคำสั่ง ":mkvimrc ~/.vimrc". ในกรณีที่มีไฟล์ ~/.vimrc อยู่แล้วและต้องการเพิ่มเติมสิ่งที่ปรับแต่งใหม่ให้ใช้คำสั่ง ":mkvimrc! ~/.vimrc" แทน. บรรทัดที่เริ่มต้นด้วยเครื่องหมายคำพูด " จะเป็นคอมเมนต์.

ตัวอย่างที่ 8.1: ไฟล์ ~/.vimrc.

```
" Use Vim settings, rather then Vi settings (much better!).
" This must be first, because it changes other options as a side effect.
set nocompatible
set history=50          " keep 50 lines of command line history
set ruler               " show the cursor position all the time
set showcmd             " display incomplete commands
set incsearch           " do incremental searching
set mouse=a             " use mouse in terminal

" Switch syntax highlighting on, when the terminal has colors
" Also switch on highlighting the last used search pattern.
if &t_Co > 2 || has("gui_running")
  syntax on
  set hlsearch
endif
```

จากตัวอย่างไฟล์ ~/.vimrc ข้างบนมีคำสั่งที่ไม่ได้อธิบายในที่นี้เช่น syntax, if ฯลฯ. ผู้อ่านสามารถเปิดเอกสารขอความช่วยเหลือในโปรแกรม vi เช่น ":help syntax" เป็นต้น.

สำหรับผู้ที่ไม่เคยใช้ vi มาก่อนอาจจะรู้สึกว่า vi เป็นบรรณาธิกรณ์ที่ใช้ยาก. แต่ถ้าเข้าใจหลักการการทำงานและลองใช้สักระยะจะคงจะเข้าใจได้ว่าทำไม vi จึงเป็นบรรณาธิกรณ์ยอดนิยม. วิธีใช้ vi ที่แนะนำไปนั้นเป็นการใช้เบื้องต้นเท่านั้น, ตัวโปรแกรมสามารถทำอะไรอีกมากมายและผู้ใช้สามารถปรับแต่งตัวการทำงานหรือการแสดงผลตามต้องการได้ด้วย.

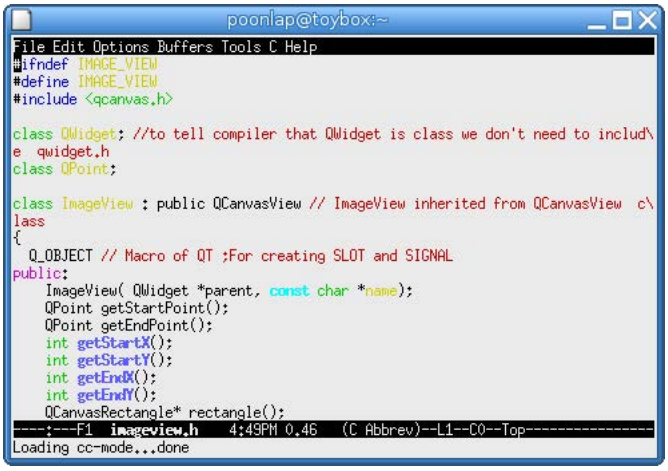
8.1.2 GNU Emacs

ในราวปี ค.ศ.1976 ก่อนหน้าที่มีบรรณาธิกรณ์ Emacs มีบรรณาธิกรณ์ที่ชื่อว่า TECMAC และ TMACS สร้างโดย Guy Steele, Dave Moon, Richard Greenblatt, Charles

Frankston และคณะ [52]. *Emacs (Editor Macros)* เป็นบรรณาธิกรณ์ที่สร้างจาก TECMAC และ TMACS โดย Richard Stallman, Guy Steele และ Dave Moon ในเวลาถัดมา. บรรณาธิกรณ์นี้ได้รับความนิยมเพราะสามารถแก้ไขข้อมูล, แสดงผลเป็นหน้าจอซึ่งแตกต่างจากบรรณาธิกรณ์เชิงบรรทัดที่มีใช้กันทั่วไปในขณะนั้น. Emacs กลายเป็นชื่อทั่วไปหมายถึงบรรณาธิกรณ์ที่ทำงานได้เหมือนกับ Emacs ที่สร้างในยุคแรกและต่อมามีคนหลายคนสร้างบรรณาธิกรณ์ในตระกูล Emacs เช่น Eine (Eien Is Not Emacs), Zmacs ฯลฯ. และบรรณาธิกรณ์ Emacs ที่มีชื่อเสียงได้แก่ *Gosling Emacs* ที่เขียนด้วยภาษา C เป็นครั้งแรกโดย James Gosling ในปี ค.ศ.1981. ต่อมาในปี ค.ศ.1985, Richard Stallman ประกาศเผยแพร่ Emacs ที่เรียกว่า GNU Emacs สู่สาธารณะเป็นซอฟต์แวร์เสรีหลักของโครงการ GNU.

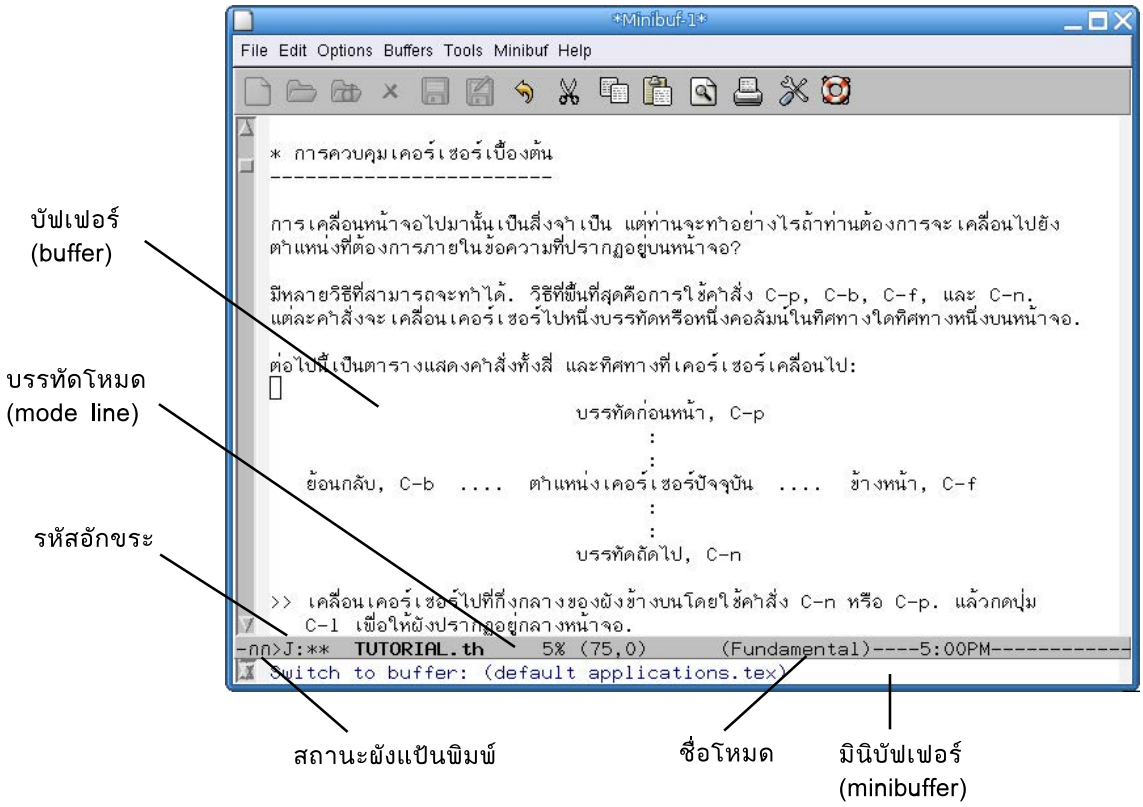
James Gosling ถือเป็นบิดาของภาษา Java.

หลังจากที่มี GNU Emacs แล้วยังมีการแตกราก Emacs ต่อเช่น Lucid Emacs, Eposch, NEmacs, Mule ฯลฯ. Lucid Emacs เป็น Emacs ที่สนับสนุนการใช้งานกับระบบ X วินโดว์อย่างเต็มตัว, มีวิดเจ็ตเฉพาะของตัวเองเพิ่มความสามารถในการแสดงผลแบบ GUI. Lucid Emacs พัฒนาโดย Jamie Zawinski และทีมงานและหลังจากนั้นกลายเป็น XEmacs. NEmacs (Nihongo Emacs) เป็นบรรณาธิกรณ์ที่สร้างจาก GNU Emacs สนับสนุนการใช้ภาษาญี่ปุ่นในปี ค.ศ.1987. ต่อมาเปลี่ยนชื่อเป็น *Mule (Multilingual Enhancement to GNU Emacs)* สนับสนุนภาษานานาชาติเช่น ภาษาจีน, เกาหลี ฯลฯ. Mule มีความสามารถสนับสนุนการใช้งานภาษาไทยด้วยซึ่งพัฒนาโดย Kenichi Handa. ในปี ค.ศ.1997 ตอนที่ GNU Emacs รุ่นที่ 20 เผยแพร่สู่สาธารณะได้รวมความสามารถรองรับการใช้งานภาษานานาชาติของ Mule ไว้ด้วย.



รูปที่ 8.7: Emacs ที่ทำงานใน `xterm`.

Emacs เป็นบรรณาธิกรณ์ที่รันได้ทั้งในเทอร์มินอลและแสดงผลเป็นหน้าต่างเฉพาะในระบบ X วินโดว์. โดยปรกติจะนิยมใช้ Emacs แบบ GUI มีหน้าต่างเฉพาะสำหรับโปรแกรมในรูปที่ 8.8. Emacs เป็นโปรแกรมขนาดใหญ่เมื่อเทียบกับ Vi. บางครั้งผู้ใช้ทั่วไปมักจะหลีกเลี่ยงการใช้ Emacs เพราะใช้ทรัพยากรในการทำงานสูงและความสามารถบางอย่างเกินความจำเป็น. สิ่งที่แตกต่างจาก Vi อย่างชัดเจนคือ Emacs ไม่มีแนวคิดเรื่องโหมดคำ



รูปที่ 8.8: ส่วนต่าง ๆของ Emacs.

สั่งและโหมดแก้ไขเหมือน Vi. หลังจากที่ Emacs เริ่มทำงานแล้วผู้ใช้สามารถป้อนข้อมูลด้วยแป้นพิมพ์ได้ทันที.

Emacs เป็นโปรแกรมที่เขียนด้วยภาษา C และมีตัวแปลภาษา Lisp ที่เรียกว่า *Emacs Lisp* ฝังอยู่ในตัวบรรณาธิกรณ์. การทำงานต่าง ๆใน Emacs จะดำเนินการโดยฟังก์ชันภาษา Lisp. ผู้ใช้สามารถนิยามฟังก์ชันใหม่ด้วยภาษา Lisp และรวบรวมเก็บในไฟล์ใช้เป็นไลบรารีเพื่อเพิ่มความสามารถให้กับ Emacs. ด้วยเหตุนี้เองทำให้ Emacs เป็นบรรณาธิกรณ์ที่ไม่ธรรมดา, สามารถทำอะไรได้หลายอย่างตั้งแต่แก้ไขข้อมูลในไฟล์เหมือนบรรณาธิกรณ์ทั่วไป, ใช้อ่านเมล, ส่งรับข้อมูลผ่าน ftp, อ่านนิวส์กรุป, ส่งข้อความ ICQ, อ่านเว็บ ฯลฯ.

ต่อไปนี้จะเรียก Emacs Lisp ย่อ ๆ เป็น Lisp.

ส่วนประกอบใน Emacs

ส่วนต่าง ๆของ Emacs อาจแบ่งได้เป็น 3 ส่วนได้แก่.

- *บัฟเฟอร์ (buffer)* —— เป็นพื้นที่แสดงข้อมูลและใช้แก้ข้อมูล. บัฟเฟอร์มีชื่อเฉพาะตัวซึ่งโดยปรกติจะเป็นชื่อไฟล์ที่ทำการแก้ไขอยู่. บัฟเฟอร์เป็นพื้นที่ในหน่วยความจำ, ถ้ามีการแก้ไขข้อมูลในบัฟเฟอร์, ข้อมูลในไฟล์ที่สัมพันธ์กับบัฟเฟอร์นั้นจะไม่เปลี่ยนแปลงทันที. ถ้าต้องการบันทึกสิ่งที่เปลี่ยนแปลงลงในไฟล์ต้องสั่งคำสั่งบันทึกข้อมูลในบัฟเฟอร์ลงในไฟล์.

  โดยปรกติ Emacs จะมีบัฟเฟอร์อย่างน้อยหนึ่งตัวได้แก่บัฟเฟอร์ชื่อ *scratch*.

บัฟเฟอร์นี้เป็นบัฟเฟอร์ที่ Emacs แสดงถ้ารันโดยไม่มีอาร์กิวเมนต์. สิ่งที่แก้ไขในบัฟเฟอร์นี้จะไม่มีการเก็บบันทึกถ้าไม่ระบุชื่อไฟล์ด้วยตัวเอง. ผู้ใช้สามารถเปิดบัฟเฟอร์ได้หลายตัวในหนึ่งหน้าต่าง. แต่ Emacs จะแสดงบัฟเฟอร์ที่เปิดตัวล่าสุดเท่านั้น. ผู้ใช้สามารถใช้คำสั่งเลือกบัฟเฟอร์ที่ต้องการมาแสดงผลและแก้ไขเมื่อต้องการ.

- *โหมดไลน์ (mode line)* — เป็นพื้นที่สำหรับแสดงสถานะต่าง ๆของ Emacs เช่น ชื่อบัฟเฟอร์, สถานะของบัฟเฟอร์และข้อมูลที่เกี่ยวข้องกับโหมดที่ใช้อยู่. โหมด (mode) คือสภาพแวดล้อมของบัฟเฟอร์และกำหนดพฤติกรรมต่าง ๆของบัฟเฟอร์ที่ทำงานอยู่. โหมดยังแบ่งออกได้เป็น*โหมดหลัก (major mode)* และ *โหมดรอง (minor mode)*. บัฟเฟอร์หนึ่งตัวจะมีโหมดหลักได้หนึ่งชนิดแต่มีโหมดรองได้หลายชนิด. เช่นเวลาเขียนรหัสต้นฉบับภาษา C ด้วย Emacs และแสดงสีแยกแยะคีย์เวิร์ดจะมีโหมดหลักเป็น c-mode และมีโหมดรองเป็น font-lock-mode เป็นต้น. ถ้าโหมดที่ใช้ต่างกัน, คีย์คำสั่งที่ใช้ได้จะแตกต่างกันด้วย.

รูปที่ 8.9: โหมดไลน์ (mode line).

ข้อมูลที่แสดงในโหมดไลน์สำคัญ ๆได้แก่.

- สภาพผังแป้นพิมพ์ — Emacs ที่สนับสนุนภาษานานาชาติมีวิธีการป้อนข้อมูลของตัวเองไม่ขึ้นกับระบบ X วินโดว์และจะแสดงผังแป้นพิมพ์ที่ใช้อยู่ในโหมดไลน์. ผังแป้นพิมพ์ภาษาไทยจะแสดงเป็น “กก>” บอกให้รู้ว่าสามารถพิมพ์ภาษาไทยได้. ส่วนผังแป้นพิมพ์ภาษาอังกฤษจะไม่มีการบอกสถานะ.
- รหัสอักขระที่ใช้ในบัฟเฟอร์ — Emacs สามารถระบุการเข้ารหัสอักขระของข้อมูลในบัฟเฟอร์ได้. การเข้ารหัสอักขระนี้จะพิจารณาตามโลแคลที่ใช้เช่น ภาษาไทยแสดงตัวอักษร “T:” หมายถึงบัฟเฟอร์นั้นมีการเข้ารหัสอักขระเป็น TIS-620.
- สภาพของบัฟเฟอร์ — ในโหมดไลน์จะแสดงสภาพของบัฟเฟอร์ด้วยเครื่องหมายต่อไปนี้
  - `--` — บัฟเฟอร์ไม่มีการเปลี่ยนแปลงใด ๆ.
  - `**` — มีการเปลี่ยนแปลงเกิดขึ้นในบัฟเฟอร์.
  - `%%` — บัฟเฟอร์นั้นเป็นบัฟเฟอร์ชนิดอ่านได้อย่างเดียว, ไม่สามารถแก้ไขได้.
- ชื่อบัฟเฟอร์ — ชื่อบัฟเฟอร์โดยปรกติจะเป็นชื่อไฟล์ที่เปิดแก้ไข.
- ชื่อโหมด — ชื่อโหมดจะเขียนอยู่ในวงเล็บ.

  - ข้อมูลอื่น ๆ —— เช่นในรูปที่ 8.9 จะมีข้อมูลแสดงตำแหน่งของเคอร์เซอร์, เวลา ฯลฯ ในโหมดไลน์. ข้อมูลเหล่านี้เกิดจากการใช้โหมดรองเช่น column-number-mode, display-time-mode เป็นต้น.

- *มินิบัฟเฟอร์ (minibuffer)* —— เป็นพื้นที่สำหรับแสดงพรอมต์หรือคำถามต่าง ๆให้ผู้ใช้โต้ตอบป้อนข้อมูลด้วยแป้นพิมพ์. คำสั่งบางคำสั่งเช่นการเปิดอ่านไฟล์แก้ไขในบัฟเฟอร์จะถามชื่อไฟล์, ผู้ใช้ต้องพิมพ์ชื่อไฟล์ในมินิบัฟเฟอร์. ถ้าต้องการยกเลิกการป้อนข้อมูลให้ใช้คำสั่ง keyboard-quit โดยการกดคีย์ Ctrl+g แล้วเคอร์จะกลับไปสู่บัฟเฟอร์ที่ทำงานอยู่.

การผูกเชื่อมคีย์

เวลาผู้ใช้กดคีย์ต่าง ๆบนแป้นพิมพ์, Emacs จะรับรู้สิ่งที่กดและเรียกใช้ฟังก์ชันภาษา Lisp ที่ผูกเชื่อม (key binding) กับคีย์นั้น ๆ. ตัวอย่างเช่น คีย์ลูกศรขวาจะผูกติดกับฟังก์ชัน forward-char. เมื่อผู้ใช้กดคีย์ลูกศรขวา, โปรแกรมจะเรียกใช้ฟังก์ชัน forward-char เลื่อนเคอร์เซอร์ถัดไปหนึ่งตัวอักขระ. การกดคีย์ “a” ก็เช่นกันจะเป็นการเรียกใช้ฟังก์ชัน self-insert-command ใส่อักขระที่พิมพ์ตรงตำแหน่งเคอร์ปัจจุบัน. ชื่อฟังก์ชันเหล่านี้มักจะเป็นชื่อที่มีความหมายและคั่นด้วยเครื่องหมาย “-”.

ฟังก์ชันพื้นฐานที่สำคัญ ๆจะมีคีย์ที่มอบหมายเตรียมไว้. ผู้ที่ใช้แป้นพิมพ์คล่องอาจจะไม่จำเป็นต้องแตะเมาส์เลยก็ได้. คีย์คำสั่งต่าง ๆที่ใช้ใน Emacs มักจะมีรูปแบบดังต่อไปนี้.

- C-*x* —— หมายถึงการกดคีย์ Ctrl พร้อมกับคีย์ *x*. ตัวอย่างเช่น “C-x C-c” หมายถึงให้กดคีย์ Ctrl ค้างไว้แล้วกด x ต่อด้วยการกดคีย์ Ctrl ค้างไว้แล้วกด c. คีย์คำสั่งนี้ผูกเชื่อมติดกับฟังก์ชัน save-buffer-kill-emacs บันทึกข้อมูลที่อยู่ในบัฟเฟอร์ลงไฟล์แล้วเลิกการทำงานของ Emacs.

- M-*x* —— หมายถึงการกดคีย์ Meta พร้อมกับคีย์ *x*. แป้นพิมพ์ของคอมพิวเตอร์ส่วนบุคคลโดยทั่วไปจะไม่มีคีย์ Meta. ในกรณีนี้จะใช้คีย์ Esc หรือคีย์ Alt แทนแล้ว Emacs จะรับรู้การกดคีย์เหล่านี้เหมือนกับการกดคีย์ Meta.

  ถ้าใช้คีย์ Esc ให้กดแล้วปล่อย, แล้วกดคีย์อื่น ๆที่ระบุตาม. แต่ถ้าใช้คีย์ Alt ให้กดพร้อมกับคีย์ที่ต้องกดตามได้เลย. วิธีใช้คีย์ต่างกันเพราะ Esc ส่งสัญญาณรหัส ASCII Esc ในขณะที่คีย์ Alt ไม่ได้ส่งสัญญาณของรหัสใด ๆแต่เป็นโมดิไฟเออร์คีย์จึงต้องกดค้างไว้. ตัวอย่างเช่น “M-v” หมายความว่าให้กดคีย์ Esc แล้วปล่อย, จากนั้นกดคีย์ v เป็นการเลื่อนหน้าจอขึ้นไปหนึ่งหน้า. ถ้าใช้คีย์ Alt ให้กดคีย์ Alt ค้างไว้แล้วตามด้วยการกดคีย์ v.

- C-M-*x* —— ใช้คีย์ Ctrl และ Meta ร่วมกัน. ถ้าใช้คีย์ Esc แทน Meta สามารถกดคีย์ Esc แล้วปล่อยตามด้วย C-*x*. ถ้าใช้คีย์ Alt, C-M-*x* หมายถึงการกดคีย์ Ctrl พร้อมกับ Alt ค้างไว้แล้วตามด้วย *x*

คำสั่งหนึ่งสามารถผูกเชื่อมกับคีย์ได้มากกว่าหนึ่งตัวและคำสั่งที่ผูกเชื่อมกับคีย์สามารถเปลี่ยนไปตามโหมดที่ใช้.

การเลื่อนเคอร์เซอร์

การเลื่อนเคอร์เซอร์สามารถใช้คีย์ลูกศรเลื่อนเคอร์เซอร์ไปมา, หรือใช้คีย์คำสั่งต่อไปนี้.

- `C-f` — forward-char —— เลื่อนเคอร์เซอร์ไปทางขวาหนึ่งอักขระ.
- `C-b` — backward-char —— เลื่อนเคอร์เซอร์ไปทางซ้ายหนึ่งอักขระ.
- `C-n` — next-line —— เลื่อนเคอร์เซอร์ไปขึ้นไปหนึ่งบรรทัด.
- `C-p` — previous-line —— เลื่อนเคอร์เซอร์ลงมาหนึ่งบรรทัด.
- `C-v` — scroll-up —— เลื่อนหน้าจอลงหนึ่งหน้า.
- `M-v` — scroll-down —— เลื่อนหน้าจอขึ้นหนึ่งหน้า.
- `C-a` — move-beginning-of-line —— เลื่อนเคอร์เซอร์ไปที่ต้นบรรทัด.
- `C-e` — move-end-of-line —— เลื่อนเคอร์เซอร์ไปที่ท้ายบรรทัด.
- `M-<` — beginning-of-buffer —— เลื่อนเคอร์เซอร์ไปที่ต้นบัฟเฟอร์.
- `M->` — end-of-buffer —— เลื่อนเคอร์เซอร์ไปที่ท้ายบัฟเฟอร์.
- `C-M-a` — beginning-of-defun —— เลื่อนเคอร์เซอร์ไปที่จุดเริ่มต้นของฟังก์ชัน. คำสั่งนี้มีประโยชน์เวลาใช้เขียนโปรแกรม.
- `C-M-e` — end-of-defun —— เลื่อนเคอร์เซอร์ไปที่ท้ายของฟังก์ชัน.
- `C-M-f` — forward-sexp —— เลื่อนเคอร์เซอร์ไปตำแหน่งถัดไปของวงเล็บหรือเครื่องหมายคำพูดที่เข้าคู่กัน. เช่นถ้ามีข้อมูล (abc) อยู่ในบัฟเฟอร์และเคอร์อยู่ที่ตำแหน่งอักษร a แล้วสั่งคำสั่ง forwawrd-sexp. เคอร์เซอร์จะเลื่อนไปอยู่ตำแหน่งของวงเล็บปิด.
- `C-M-b` — backward-sexp —— เลื่อนเคอร์เซอร์กลับไปตำแหน่งไปของวงเล็บหรือเครื่องหมายคำพูดที่เข้าคู่กัน.

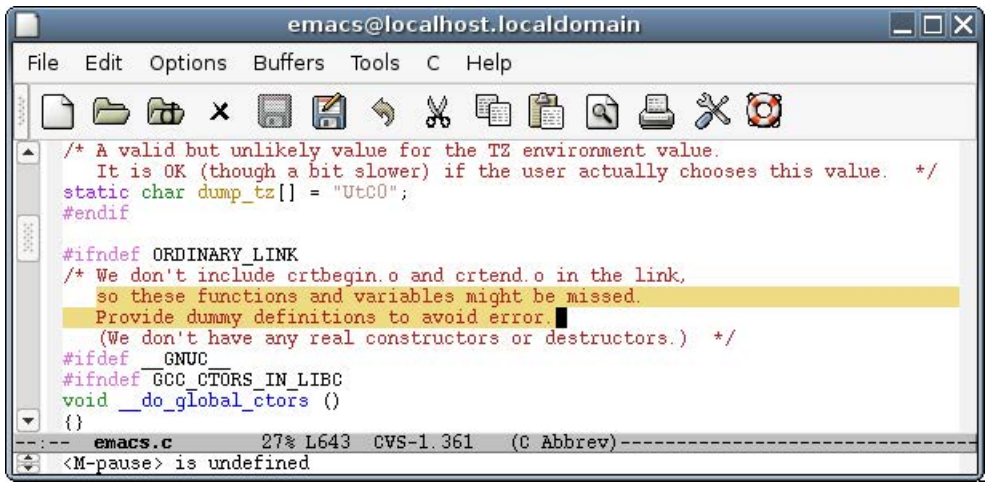
ลบอักขระหรือข้อความ

การลบอักขระสามารถใช้คีย์ Backspace หรือ Del ได้เหมือนกับบรรณาธิกรณ์ปรกติ. นอกจากนี้แล้วจะมีคำสั่งต่าง ๆลบอักขระแบบอื่น ๆเตรียมไว้ด้วยได้แก่.

- `C-d` — delete-char —— ให้ผลเหมือนกันกดคีย์ Del คือลบอักขระหนึ่งตัวที่อยู่ตรงตำแหน่งเคอร์เซอร์.
- `M-d` — kill-word —— ลบคำหนึ่งคำจากตำแหน่งเคอร์เซอร์ปัจจุบัน.
- `C-k` — kill-line —— ลบอักขระตั้งแต่ตำแหน่งเคอร์เซอร์ปัจจุบันจนถึงสุดบรรทัด.
- `C-w` — kill-region —— ลบข้อมูลที่อยู่ในรีเจียนที่กำหนดไว้.

มาร์กและรีเจียน

มาร์ก (mark) ใน Emacs หมายถึงการบันทึกตำแหน่งของเคอร์เซอร์เพื่อใช้กำหนด*รีเจียน (region)* สำหรับการประมวณผลข้อมูล. รีเจียนคือช่วงพื้นที่ระหว่างตำแหน่งที่มาร์กไว้และตำแหน่งเคอร์เซอร์ปัจจุบัน. การบันทึกตำแหน่งมาร์กให้ Emacs รับรู้จะใช้คำสั่ง set-mark-command ซึ่งโดยปรกติจะผูกเชื่อมต่อคำสั่งนี้ไว้กับคีย์ `C-@` และ `C-Space`. เนื่องจากการกดคีย์ `@` ต้องกด `Shift` ค้างไว้ด้วย, ดังนั้นการบันทึกตำแหน่งมาร์กมักจะใช้ `C-Space` ซึ่งทำได้ง่ายกว่าการกด `C-@`.



รูปที่ 8.10: การเซ็ตมาร์ก.

ในรูปที่ 8.10 เป็นตัวอย่างการเซ็ตมาร์กเริ่มที่ตำแหน่ง “so these ...” โดยการกดคีย์ `C-Space`. ตรงมินิบัฟเฟอร์จะมีข้อความ “Mark set” บอกให้ผู้ใช้รู้ว่าได้เซ็ตมาร์กแล้ว. หลังจากนั้นให้เลื่อนเคอร์เซอร์ไปตำแหน่งที่ต้องเพื่อกำหนดรีเจียนที่ต้องการ. ในตัวอย่างเคอร์เซอร์อยู่ที่ “... avoid error.” ดังนั้นรีเจียนจะเป็นช่วงข้อความ “so these ...” จนถึง “avoid error.”. Emacs ในรูปจะแสดงสีพื้นตัวอักษรของรีเจียนให้แตกต่างจากช่วงอื่น, แต่โดยปรกติจะไม่แสดงสีพื้นอักษรให้ถ้าไม่อยู่ในโหมด transient-mark-mode. หรือเวลาเซ็ตมาร์ก, กดคีย `C-Space` สองครั้งก็จะใช้โหมด transient-mark-mode ชั่วคราวให้.

คำสั่ง Emacs บางตัวจะประมวลข้อมูลที่อยู่ในรีเจียนเช่น kill-region, copy-region-as-kill, indent-region เป็นต้น. ถ้ากำหนดรีเจียนแล้วกดคีย์ `C-w` ซึ่งผูกเชื่อมไว้กับคำสั่ง kill-region, โปรแกรม Emacs จะลบข้อมูลที่เลือกไว้ในรีเจียน.

อาร์กิวเมนต์

คำสั่งใน Emacs สามารถระบุอาร์กิวเมนต์ได้เช่นในกรณีที่ต้องการลบบรรทัดที่อยู่ในบัฟเฟอร์ 10 บรรทัด, แทนที่จะกดคีย์ `C-k` สิบครั้ง, สามารถระบุจำนวนครั้งด้วยการกดคีย์ `C-u` (universal-argument) แล้วพิมพ์จำนวนบรรทัดที่ต้องการลบ.

คำสั่ง universal-argument เป็นวิธีการรับอาร์กิวเมนต์. เช่นคำสั่ง kill-line จะรับอาร์กิวเมนต์เป็นจำนวนบรรทัด. ถ้าไม่มีจำนวนบรรทัดให้เป็นอาร์กิวเมนต์ก็จะลบหนึ่ง

บรรทัด. ในทำนองเดียวกัน, คำสั่ง delete-char, delete-word สามารถรับอาร์กิวเมนต์ได้ด้วยเช่นกัน.

ค้นข้อมูล, เปลี่ยนสายอักขระ

การค้นข้อมูลใน Emacs จะกดคีย์ C-s ซึ่งผูกเชื่อมไว้กับคำสั่ง isearch-forward เป็นการหาสายอักขระขณะที่พิมพ์ไปเรื่อย ๆ. การหาข้อมูลนี้จะหาข้อมูลที่อยู่ถัดจากตำแหน่งเคอร์เซอร์ปัจจุบัน. เมื่อกดคีย์ C-s แล้วจะรอรับคำที่ต้องการหาโดยแสดงพรอมต์ "I-search:" ที่มินิบัฟเฟอร์. ขณะที่ผู้ใช้ป้อนข้อมูลก็จะเลื่อนเคอร์เซอร์ในบัฟเฟอร์ไปที่ตำแหน่งข้อมูลที่หาเจอทันที. ถ้าป้อนข้อมูลด้วยอักษรภาษาอังกฤษตัวเล็กทั้งหมดจะเป็นการหาโดยไม่แยกแยะอักษรตัวใหญ่ตัวเล็ก, แต่ถ้าในสายอักขระมีอักษรตัวใหญ่ปรากฏอยู่ด้วยจะแยกแยะการหาสายอักขระระหว่างอักษรตัวใหญ่กับตัวเล็ก. การเลื่อนตำแหน่งที่ต้องการหาในบัฟเฟอร์ถัดไปให้กดคีย์ C-s ไปเรื่อย ๆ.

การหยุดคำสั่งการหาข้อมูลมีอยู่ประเภทของกดคีย์ Enter เพื่อจบการหาข้อมูล. ในกรณีนี้เคอร์เซอร์จะหยุดอยู่ตรงตำแหน่งที่หาข้อมูลเจอ. การหยุดการทำงานอีกวิธีหนึ่งคือกดคีย์ C-g ซึ่งผูกเชื่อมไว้กับคำสั่ง keyboard-quit. ในขณะที่ Emacs กำลังหาข้อมูลที่ต้องการในบัฟเฟอร์, ตัวโปรแกรมจะรับข้อมูลที่ต้องการหาตลอดเวลาจนกว่าจะกดคีย์ Enter เพื่อจบการทำงานหรือกดคีย์ C-s เพื่อหาข้อมูลต่อไป. การกดคีย์ C-g จะเป็นการยกเลิกป้อนข้อมูลซึ่งหมายถึงเลิกหาข้อมูลไปในตัว.

ในทำนองเดียวกัน, การหาคำก่อนหน้าตำแหน่งเคอร์เซอร์จะใช้คีย์ C-r ซึ่งผูกเชื่อไว้กับคำสั่ง isearch-backward.

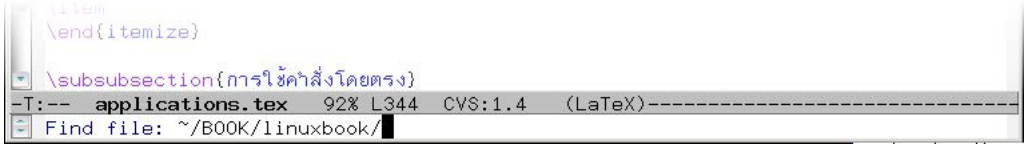
การจัดการบัฟเฟอร์และไฟล์

Emacs ที่รันอยู่ในระบบ X วินโดว์จะมีเมนูกระทำการเกี่ยวกับไฟล์เช่นอ่านไฟล์, บันทึกไฟล์ไว้ให้. การกระทำเหล่านี้เป็นคำสั่งซึ่งผูกเชื่อมกับคีย์ต่าง ๆและมักจะเริ่มต้นด้วยคีย์ C-x. คีย์บางตัวอาจจะเหมือนกับคีย์ลัดของแอพพลิเคชันในสภาพแวดล้อม GNOME เช่น Ctrl + S แต่มีความหมายแต่งกัน. ถ้าผู้ใช้คุ้นเคยกับคีย์คำสั่งต่าง ๆ ใน Emacs แล้ว, ก็จะไม่ต้องใช้เมาส์เลือกการกระทำต่าง ๆจากเมนูและใช้งานได้คล่องกว่า. นอกจากนั้นคีย์คำสั่งใน Emacs สามารถใช้ใน bash เชลล์ได้ด้วย.

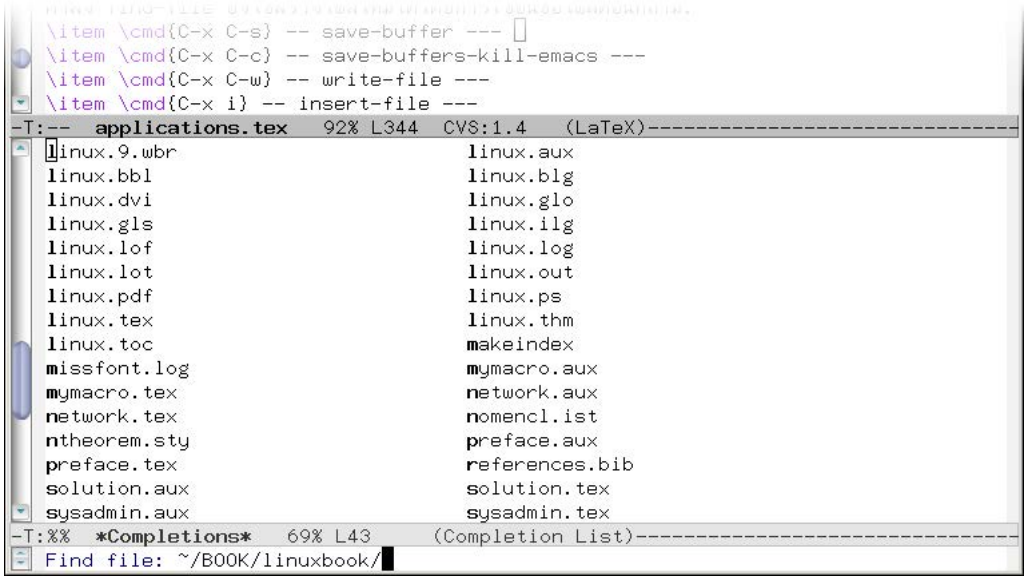
การกระทำกับไฟล์หรือบัฟเฟอร์บางครั้ง Emacs ต้องการให้ผู้ใช้ป้อนข้อมูลเช่นชื่อบัฟเฟอร์หรือชื่อไฟล์ด้วยแป้นพิมพ์. Emacs จะแสดงพรอมต์หรือคำถามต่าง ๆที่มินิบัฟเฟอร์. ถ้าผู้ใช้ต้องการเลิกการป้อนข้อมูลในมินิบัฟเฟอร์ให้กดคีย์ C-g (keyboard-quit).

- C-x C-f — find-file —— สำหรับเปิดอ่านไฟล์เข้าสู่บัฟเฟอร์เพื่อแก้ไขต่อไป. เมื่อสั่งคำสั่งนี้แล้ว Emacs จะแสดงพรอมต์ถามชื่อไฟล์ที่มินิบัฟเฟอร์. ผู้ใช้ต้องพิมพ์ชื่อไฟล์ที่ต้องการเปิดและการพิมพ์ชื่อไฟล์ Emacs จะชื่อเติมเต็มชื่อไฟล์ให้ด้วย. การเติมเต็มชื่อไฟล์ให้กดคีย์ Tab หรือกดคีย์ Tab Tab เพื่อแสดงรายการไฟล์หรือไดเรกทอรีในกรณีที่ไม่สามารถเติมเต็มชื่อไฟล์ให้ทันที.

  คำสั่ง find-file ยังใช้สร้างไฟล์ใหม่ได้โดยการเขียนชื่อไฟล์ตอนที่ถาม.



รูปที่ 8.11: เปิดไฟล์หรือสร้างไฟล์ใหม่.



รูปที่ 8.12: เติมเต็มชื่อไฟล์หรือแสดงรายการไฟล์.

- `C-x C-s` – save-buffer —– หลังจากที่แก้ไขข้อมูลที่อยู่ในบัฟเฟอร์และต้องการบันทึกข้อมูลลงในไฟล์ให้กดคีย์ `C-x C-s`. Emacs จะแสดงข้อความบอกให้ผู้ใช้รู้ว่าได้บันทึกข้อมูลลงในไฟล์แล้วที่มินิบัฟเฟอร์.

- `C-x C-w` – write-file —– บันทึกข้อมูลที่อยู่ในบัฟเฟอร์ลงในไฟล์โดยจะถามชื่อไฟล์ให้ผู้ใช้ระบุในบริเวณมินิบัฟเฟอร์.

- `C-x C-c` – save-buffers-kill-emacs —– จบการทำงาน Emacs และถ้ามีบัฟเฟอร์ที่ยังไม่ได้บันทึกลงในไฟล์จะถามให้ผู้ย้ำให้ผู้ใช้บันทึกบัฟเฟอร์ก่อนจบการทำงาน.

- `C-x i` – insert-file —– คำสั่งนี้ใช้สำหรับแทรกข้อมูลจากไฟล์อื่น ๆที่ระบุในตำแหน่งเคอร์เซอร์ปัจจุบัน.

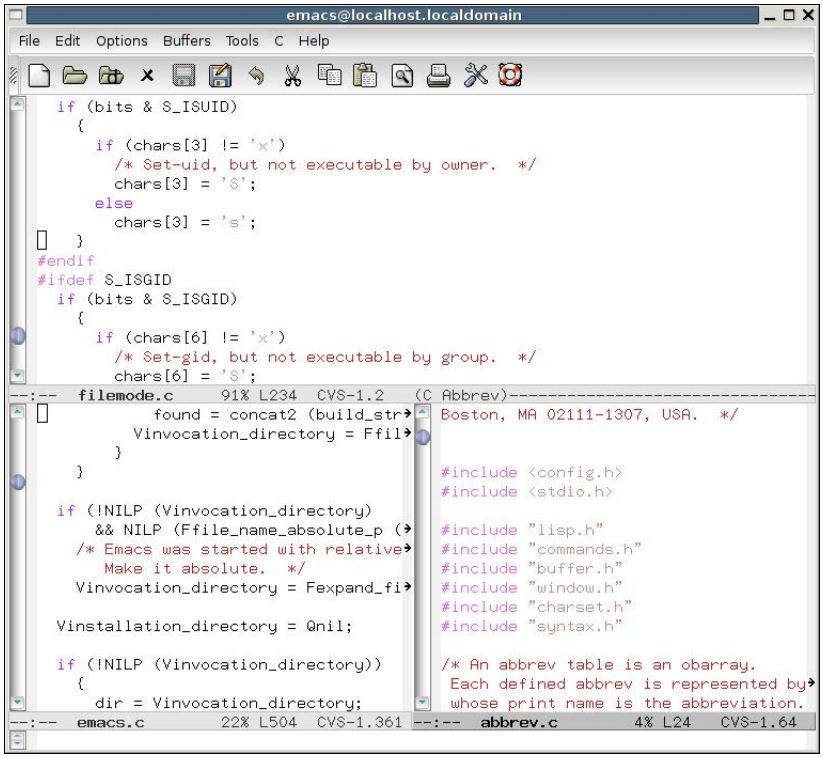
- `C-x b` – switch-to-buffer —– บรรณาธิกรณ์ Emacs สามารถเปิดแก้ไขไฟล์ได้หลายบัฟเฟอร์พร้อม ๆกัน. ถ้า Emacs ที่ใช้มีหน้าต่างเดียวและเปิดบัฟเฟอร์หลายตัว, จะแสดงบัฟเฟอร์ที่เปิดตัวล่าสุดเท่านั้น. บัฟเฟอร์ที่เปิดไว้ก่อนจะไม่หายไปไหนเพียงแต่จะไม่แสดงให้เห็นในหน้าต่าง. วิธีการสลับบัฟเฟอร์ไปมาให้กดคีย์

C-x b แล้วเขียนชื่อบัฟเฟอร์ที่ต้องการเปลี่ยนที่มินิบัฟเฟอร์. Emacs มีการช่วยเติมเต็มชื่อบัฟเฟอร์ให้ด้วยถ้ากด Tab หรือ Tab Tab.

- C-x C-b — list-buffers —— คำสั่งสำหรับแสดงชื่อบัฟเฟอร์ที่ Emacs รับรู้ทั้งหมด.
- C-x k — kill-buffer —— ทำลายบัฟเฟอร์ที่ใช้อยู่. Emacs จะถามย้ำถ้าบัฟเฟอร์ยังไม่ได้บันทึกลงไฟล์.

หน้าต่าง

หน้าต่างใน Emacs คือพื้นที่สำหรับแสดงเนื้อหาของบัฟเฟอร์และในที่นี้จะเรียกหน้าต่างโดยรวมที่แสดงในระบบ X วินโดว์ว่า*เฟรม (frame)*. หน้าต่างหนึ่งสามารถแบ่งออกเป็นหน้าต่างย่อย ๆได้ตามที่แสดงในรูปที่ 8.13. การแบ่งหน้าต่างย่อยนี้เหมาะสำหรับการแสดงบัฟเฟอร์หลายบัฟเฟอร์พร้อม ๆกันแทนที่จะมีหน้าต่างเดียวและแสดงบัฟเฟอร์ได้ตัวเดียว.



รูปที่ 8.13: หน้าต่างย่อยแบบต่าง ๆใน Emacs.

คีย์คำสั่งที่ใช้ควบคุมหน้าต่างมีดังต่อไปนี้.

- C-x 2 — split-window-vertically —— แบ่งหน้าต่างที่ใช้อยู่ออกเป็นสองส่วนในแนวตั้ง.

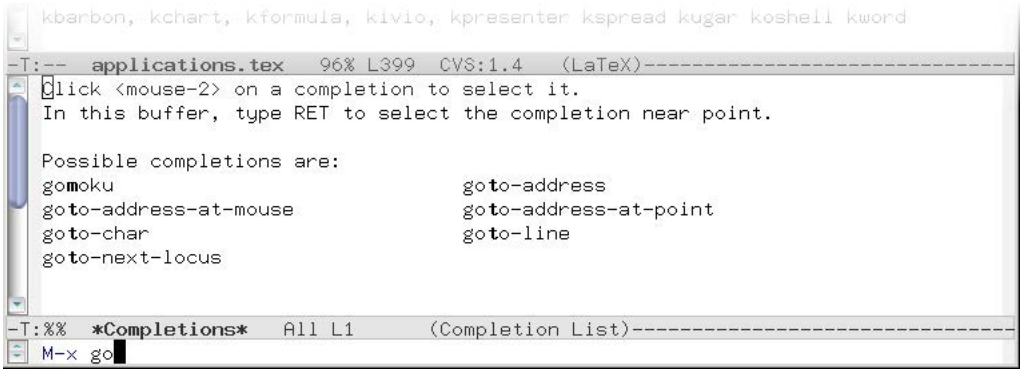
- C-x 3 — split-window-horizontally —— แบ่งหน้าต่างที่ใช้อยู่ออกเป็นสองส่วนในแนวนอน.
- C-x o — other-window —— เลื่อนเคอร์เซอร์ไปในแต่ละหน้าต่างย่อยที่มีอยู่ในเฟรม.
- C-x 0 — delete -window —— ปิดหน้าต่างที่ใช้อยู่แต่ไม่ทำลายบัฟเฟอร์ที่หน้าต่างนั้นแสดงผล.
- C-x 1 — delete-other-windows —— ปิดหน้าต่างอื่น ๆที่ไม่ใช่หน้าต่างที่ใช้อยู่.
- C-x 5 2 — make-frame-command —— สร้างเฟรมใหม่ต่างหาก.

ระวังสับสนระหว่างแนวคิดเรื่องหน้าต่างและบัฟเฟอร์.

การใช้คำสั่งโดยตรง

สำหรับผู้ที่เริ่มใช้ Emacs ควรจะจำคีย์คำสั่งสำคัญ ๆเช่นการเปิดอ่านไฟล์, บันทึกบัฟเฟอร์, เลิกการทำงาน เป็นต้น. คำสั่งใน Emacs มีมากมายและไม่มีทางที่จะจำได้หมดแต่ Emacs มีวิธีช่วยให้ผู้ใช้สั่งคำสั่งได้โดยตรงโดยการการใช้คีย์คำสั่ง M-x (execute-extended-command) เป็นการใช้คำสั่งโดยตรงในกรณีที่คำสั่งนั้นไม่ได้ผูกเชื่อมกับคีย์ใด ๆหรือต้องการสั่งคำสั่งเป็นชื่อคำสั่งแทนการกดคีย์คำสั่ง.

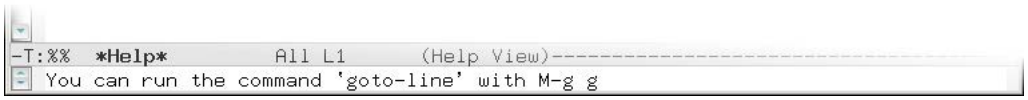
Emacs จะรอรับชื่อคำสั่งจากผู้ในบริเวณมินิบัฟเฟอร์หลังจากที่กดคีย์ M-x. ถ้าต้องการยกเลิกการสั่งคำสั่งให้กดคีย์ C-g. ถ้าผู้ใช้พอจะจำชื่อคำสั่งได้สามารถเขียนชื่อคำสั่ง (ชื่อฟังก์ชันใน Emacs Lisp) ทั้งหมดหรือบางส่วนและกด Tab ให้ Emacs ช่วยเติมเต็มชื่อคำสั่งให้. หรือถ้าต้องการดูชื่อคำสั่งที่เป็นไปได้ทั้งหมดก็ให้กด Tab Tab.



รูปที่ 8.14: การสั่งคำสั่งโดยตรงใน Emacs.

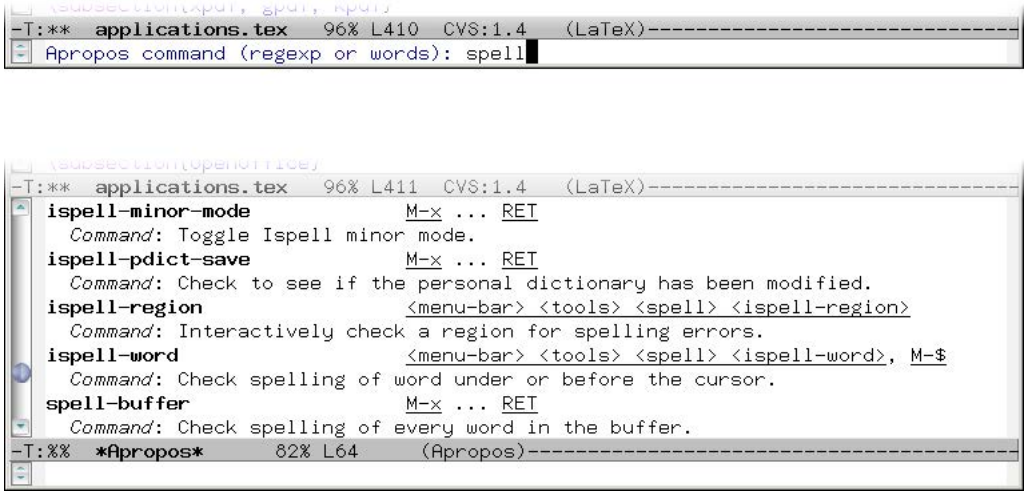
จากรูปที่ 8.14 เป็นตัวอย่างการสั่งคำสั่ง goto-line โดยตรง. หลังจากที่กดคีย์ M-x แล้วพิมพ์คำว่า “go” แล้วกด Tab Tab ดูว่ามีคำสั่งอะไรที่ขึ้นต้นคำว่า “go” บ้าง. ในกรณีนี้เราต้องการสั่งคำสั่ง goto-line ก็พิมพ์ชื่อคำสั่งต่อ “to-l” แล้วกด Tab ให้ Emacs ช่วยเติมเต็มคำสั่งให้, ไม่ต้องเขียนคำสั่งทั้งหมดด้วยตัวเอง. เมื่อพิมพ์ชื่อคำสั่งเรียบร้อยแล้วกด Enter เพื่อสั่งงานต่อไป.

ถ้าคำสั่งนั้นมีการผูกเชื่อมไว้กับคีย์, Emacs จะแสดงข้อมูลเพิ่มเติมบอกไว้ว่ามีคีย์คำสั่งอะไรที่ใช้สั่งคำสั่งนั้น (รูปที่ 8.15).



รูปที่ 8.15: ข้อมูลเพิ่มเติมหลังจากสั่งคำสั่งโดยตรงใน Emacs.

ถ้าเรากดคีย์ M-x แล้วกด Tab Tab จะเห็นชื่อคำสั่งต่าง ๆที่สามารถใช้ได้. ในกรณีที่ต้องทำอะไรสักอย่างแต่ไม่รู้ชื่อคำสั่งอาจจะดูรายการคำสั่งที่แสดงไว้ทั้งหมด. แต่วิธีไม่ใช่วิธีที่เหมาะสมและ Emacs มีคำสั่ง apropos-command ซึ่งผูกเชื่อมไว้กับคีย์ C-h a. การหาชื่อคำสั่งจะใช้ regular expression หรือคำธรรมดาในการค้นหา.



รูปที่ 8.16: คำสั่ง apropos และผลลัพธ์ใน Emacs.

ผลลัพธ์ที่แสดงจะมีรายละเอียดคร่าวอธิบายการทำงานและคีย์คำสั่งที่ผูกเชื่อมไว้. จากรูป 8.16 เป็นตัวอย่างหาคำสั่งที่เกี่ยวกับการตรวจสะกดคำ. ถ้าต้องการตรวจการสะกดคำภาษาไทยอังกฤษสามารถใช้คำสั่ง ispell-word หรือคีย์คำสั่ง M-$.

ขอความช่วยเหลือ

ใน Emacs จะมีคำสั่งช่วยเหลือต่าง ๆและจะมีคีย์คำสั่งขึ้นต้นด้วย C-h. คีย์คำสั่งที่เกี่ยวกับการช่วยเหลือต่าง ๆที่สำคัญ ๆได้แก่.

- C-h a — command-apropos —— ใช้ค้นหาคำสั่งใช้งาน.

- `C-h b` – describe-bindings —— อธิบายคีย์คำสั่งในโหมดต่าง ๆที่ใช้อยู่ว่ามีคีย์คำสั่งอะไรบ้างและผูกเชื่อมไว้กับคำสั่งชื่ออะไร. ผู้ใช้สามารถสำรวจคีย์คำสั่งทั้งหมดที่ใช้ได้ด้วยคำสั่งนี้.
- `C-h f` – describe-function —— หลังจากที่ระบุชื่อฟังก์ชันจะอธิบายการทำงานอย่างคร่าว ๆและคีย์คำสั่งที่ผูกเชื่อมไว้ (ถ้ามี).
- `C-h i` – info —— ดูคู่มือการใช้งานโปรแกรมต่างแบบ info ด้วย Emacs.
- `C-h k` – describe-key —— อธิบายคีย์คำสั่งที่กดว่าผูกเชื่อมอยู่กับคำสั่งอะไร.
- `C-h m` – describe-mode —— อธิบายโหมดที่บัฟเฟอร์ใช้อยู่.
- `C-h p` – finder-by-keyword —— หาชื่อแพกเกจเพิ่มเติมที่ใช้กับ Emacs.
- `C-h t` – help-with-tutorial —— แสดงติวทอเรียลการใช้การ Emacs ในภาษาท้องถิ่นเช่นภาษาไทยเป็นต้น (รูปที่ 8.8).

ปรับแต่ง Emacs

โปรแกรม Emacs มีไฟล์ตั้งค่าเริ่มต้นได้แก่ `~/.emacs` เป็นไฟล์เท็กซ์ที่เขียนด้วยภาษา Emacs Lisp. ในไฟล์นี้สามารถใช้ฟังก์ชันที่เตรียมไว้ปรับแต่งพฤติกรรมต่าง ๆของ Emacs เช่นโหมดที่ต้องการใช้, ผูกเชื่อมคีย์คำสั่งเป็นต้น. การเขียนไฟล์ตั้งค่าเริ่มต้นนี้ต้องมีความรู้เบื้องต้นเกี่ยวกับภาษา Emacs Lisp ซึ่งจะไม่อธิบายในหนังสือเล่มนี้แต่สามารถหาอ่านได้จากคู่มือ Emacs จากอินเทอร์เน็ตหรือในเมนูของ Emacs.

ตัวอย่างที่ 8.2: ไฟล์ตั้งค่าเริ่มต้นของ Emacs

```
;; ตัวอย่างไฟล์ตั้งค่าเริ่มต้น ~/.emacs
(set-language-environment 'thai) ;; สภาพแวดล้อมภาษาไทย
(global-font-lock-mode)          ;;
(transient-mark-mode t)          ;;
;; ตั้งค่าตัวแปล user-mail-address
(setq user-mail-address "poonlap@linux.thai.net")
;; ให้ text-mode เป็นโหมดปริยายของบัฟเฟอร์ใหม่
(setq default-major-mode 'text-mode)
```

การใช้ภาษาไทยกับ Emacs

Emacs ที่สามารถใช้ภาษาไทยจะต้องเป็นรุ่น 20 เป็นต้นไป, แสดงผลภาษาไทยและพิมพ์ภาษาไทยได้โดยมีผังแป้นพิมพ์ในตัวโปรแกรมไม่เกี่ยวข้องกับระบบ X วินโดว์. Emacs จะเลือกฟอนต์ภาษาไทยที่เหมาะสมในการแสดงผลซึ่งถ้าไม่ชอบฟอนต์ใช้สามารถระบุด้วยตัวเลือก `-fn` *`fontname`* ตอนที่รันโปรแกรม Emacs. ตัวอย่างต่อไปนี้เป็นการรันโปรแกรมโดยใช้ฟอนต์ -nectec-emacs-bold-r-normal—18-180-72-72-c-90-tis620-0 ซึ่งเป็นฟอนต์ที่แก้ไขมาจากฟอนต์บิตแมปของเนคเทคใช้กับ Emacs ได้.

ตัวอย่างที่ 8.3: การระบุฟอนต์ที่ต้องการใช้กับ Emacs.

```
$ emacs -fn -nectec-emacs-bold-r-normal--18-180-72-72-c-90-tis620-0↵
 --- หรือ ---
$ emacs -fn themacs18↵
```

ฟอนต์ภาษาไทยที่ใช้กับ Emacs ต้องเป็นฟอนต์ที่ขนาดของอักขระทุกตัวมี่ค่าคงที่รวมถึงสระและวรรณยุกต์ด้วย.

Emacs ที่ต้องการใช้ภาษาไทยต้องสั่งคำสั่ง set-language-environment ให้เป็นภาษาไทยด้วย. คำสั่งนี้จะเตรียมแป้นพิมพ์และสภาพแวดล้อมการทำงานภาษาไทยให้. ถ้าไม่สั่งคำสั่งนี้โดยตรงก็เขียนไว้ในไฟล์ตั้งค่าเริ่มต้นที่แสดงในตัวอย่างที่ 8.2. เวลาพิมพ์ต้องการพิมพ์ภาษาไทยให้กดคีย์ `C-\` จะสลับเปลี่ยนผังแป้นพิมพ์ไปมาระหว่างภาษาอังกฤษและภาษาไทย.

8.2 แอพพลิเคชันสำนักงาน

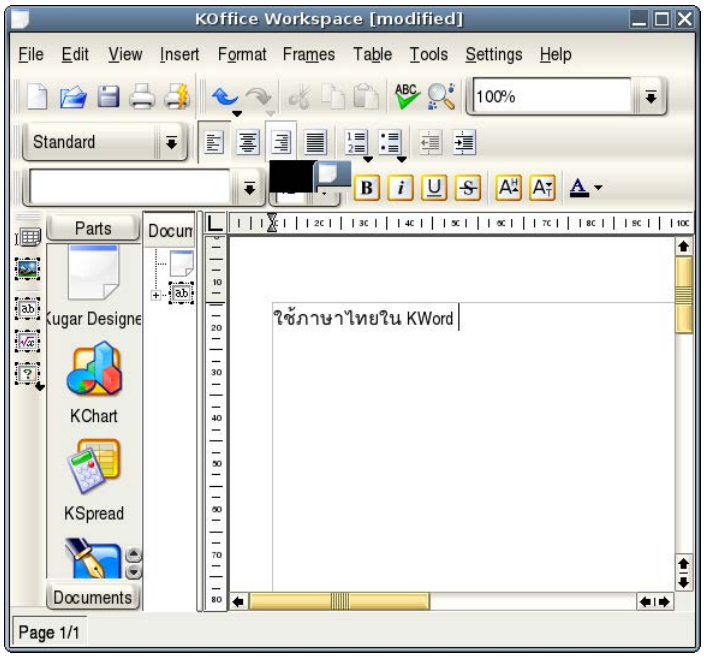
ในปัจจุบันแอพพลิเคชันสำนักงานเช่นเตรียมเอกสาร, ตารางคำนวณ, นำเสนอผลงาน ฯลฯ มีความสำคัญสำหรับการใช้งานประเภทเดสก์ท็อป. โปรแกรมประเภทนี้เริ่มพัฒนาและใช้งานได้ดีหลังจากที่มีทูลคิตสมัยใหม่เช่น GTK+ และ Qt ออกมา.

ถ้าพูดถึงแอพพลิเคชันสำนักงานคงจะหนีไม่พ้นชุดแอพพลิเคชันสำนักงาน OpenOffice ซึ่งเดิมที่เป็นชุดแอพพลิเคชันที่เรียกว่า ApplixWare สำหรับใช้ระบบปฏิบัติการยูนิกซ์. ต่อมาบริษัท Sun Microsystems ซื้อ ApplixWare และพัฒนาเพิ่มเติมให้เป็น StarOffice และท้ายสุดเปิดเผยรหัสต้นฉบับและมอบให้ชุมชนพัฒนาต่อในชื่อของ OpenOffice. ในประเทศไทยจะมีแอพพลิเคชันสำนักงานที่ใช้กับลินุกซ์ 2 ตัวที่ใช้กันอย่างกว้างขวางได้แก่ OfficePladao และ OfficeTLE ซึ่งเป็นชุดซอฟต์แวร์ที่พัฒนาต่อยอดมาจาก OpenOffice ให้สนับสนุนการใช้งานกับภาษาไทยได้อย่างดี. แต่ในปัจจุบันเริ่มมีการรวมความสามารถการประมวลผลภาษาไทยเหล่านั้นกลับเข้าสู่โครงการ OpenOffice เรื่อย ๆ และในอนาคตคาดว่าสามารถใช้ภาษาไทยได้กับ OpenOffice โดยไม่ต้องพึ่ง OfficePladao หรือ OfficeTLE.

นอกจากชุดแอพพลิเคชันสำนักงาน OpenOffice แล้วยังมีแอพพลิเคชันอื่น ๆที่ใช้ทูลคิตสมัยใหม่เช่น GTK+ และ Qt เช่น

- Abiword — โปรกแกรมเวิร์ดโปรเซส WYSIWYG (What you see is what you get) ที่ใช้ทูลคิต GTK+.
- Gnumeric — โปรแกรมตารางคำนวณสำหรับสภาพแวดล้อม GNOME.
- Koffice — ชุดแอพพลิเคชันสำนักงานในสภาพแวดล้อมเดสก์ท็อป KDE. ผู้ใช้สามารถเริ่มการทำงานด้วยคำสั่ง `koshell` ซึ่งเป็นตัวโปรแกรมสำหรับเรียกใช้โปรแกรมส่วนประกอบอื่น ๆอีกที. แอพพลิเคชันเหล่านี้ใช้ภาษาไทยได้ในระดับดี.
  - `kaddressbook` — โปรแกรมเก็บที่อยู่สำหรับติดต่อกับบุคคล. สามารถเก็บข้อมูลในระบบไฟล์, LDAP หรือ SQL เซิร์ฟเวอร์.

- `karbon` — โปรแกรมสำหรับวาดรูปแบบเวคเตอร์.
- `kchart` — โปรแกรมสำหรับสร้างรูปชาร์ต, แผนภาพสถิติต่าง ๆ.
- `kformula` — โปรแกรมสำหรับเขียนสูตรเลขใช้กับแอพพลิเคชันสำนักงานอื่น ๆ.
- `kivio` — โปรแกรมสำหรับเขียนแผนผังหรือโฟลชาร์ต.
- `kpresenter` — โปรแกรมนำเสนอผลงาน.
- `kspread` — โปรแกรมตารางคำนวณ.
- `kugar` — โปรแกรมสำหรับสร้างรายงาน.
- `kword` — โปรแกรมเวิร์ดโปรเซสเซอร์.



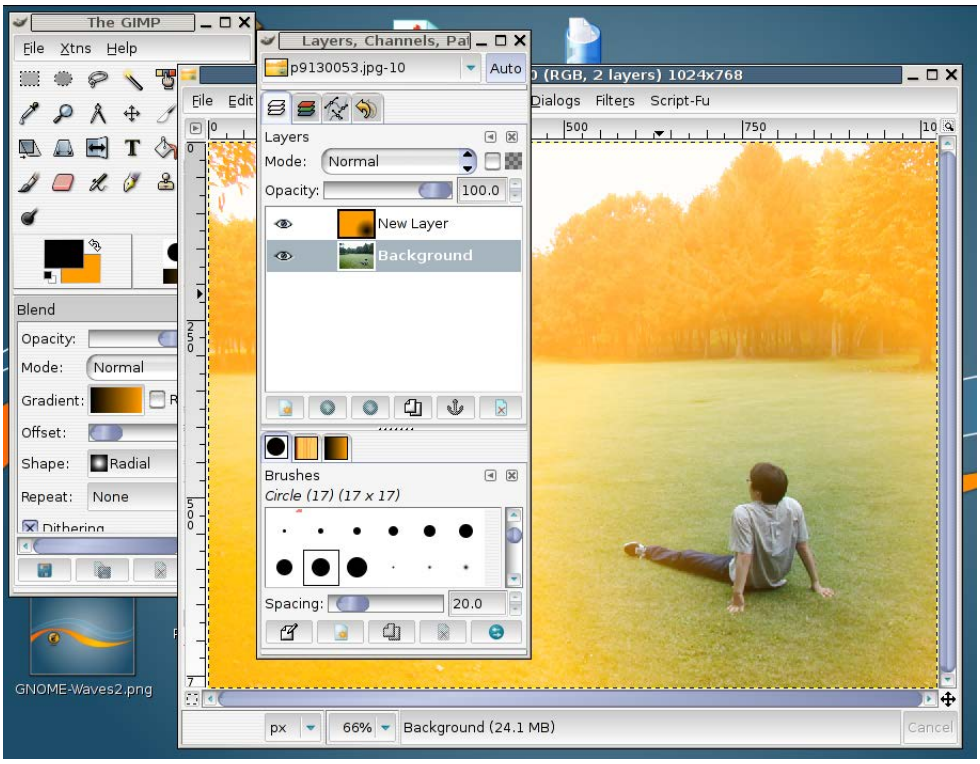
รูปที่ 8.17: ชุดแอพพลิเคชันสำนักงาน KOffice.

8.3 กราฟิก

โปรแรกมที่เกี่ยวกับงานกราฟิกในลินุกซ์ก้าวหน้าไปมากเนื่องจากมีการพัฒนาทูลคิตและไลบรารีอำนวยความสะดวกมากมาย. ในช่วงนี้จะแนะนำโปรแกรมพอสังเขปเป็นแนวทางในการเลือกใช้แอพพลิเคชันให้เหมาะสมกับงาน.

8.3.1 Gimp

Gimp (GNU Image manipulation program) เป็นแอพพลิเคชันเดสก์ท็อปหลักสำหรับลินุกซ์ใช้ปรับแต่งภาพหรือสร้างภาพดิจิตอล. Gimp เป็นที่ยอมรับว่าเป็นโปรแกรมก



รูปที่ 8.18: โปรแกรม Gimp.

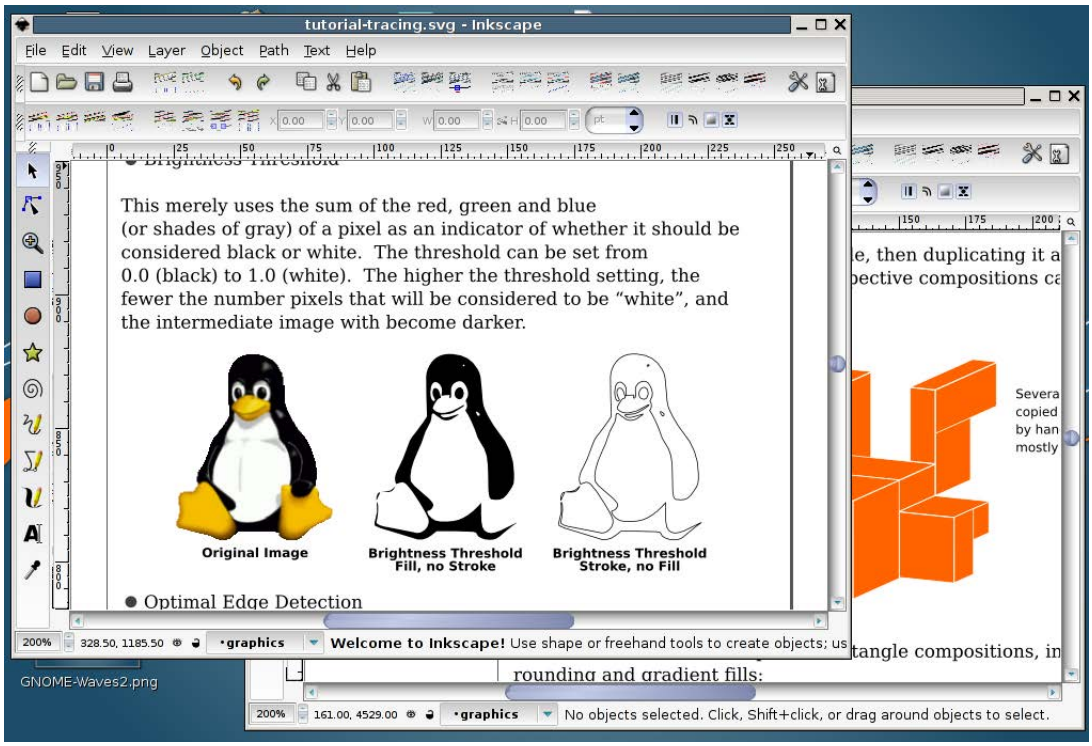
ราฟิกที่เยี่ยมยอดและเป็นตัวเลือกใช้แทนโปรแกรมในระบบปฏิบัติการวินโดวส์เช่น PhotoShop ได้ในเกือบทุกกรณี. Gimp มีการจัดการรูปด้วย*เลเยอร์ (layer)*, รองรับไฟล์รูปภาพหลายฟอร์แมต, มี*ปลั๊กอิน (plugin)* ช่วยอำนวยความสะดวกในการสร้างเอฟเฟคต่าง ๆ. นอกจากนั้นมีภาษา scheme ในตัวใช้เขียนสคริปต์เพื่อสร้างปลั๊กอินหรือเขียนสคริปต์ประมวลรูปภาพโดยไม่ใช้ GUI ก็ได้.

8.3.2 Inkscape

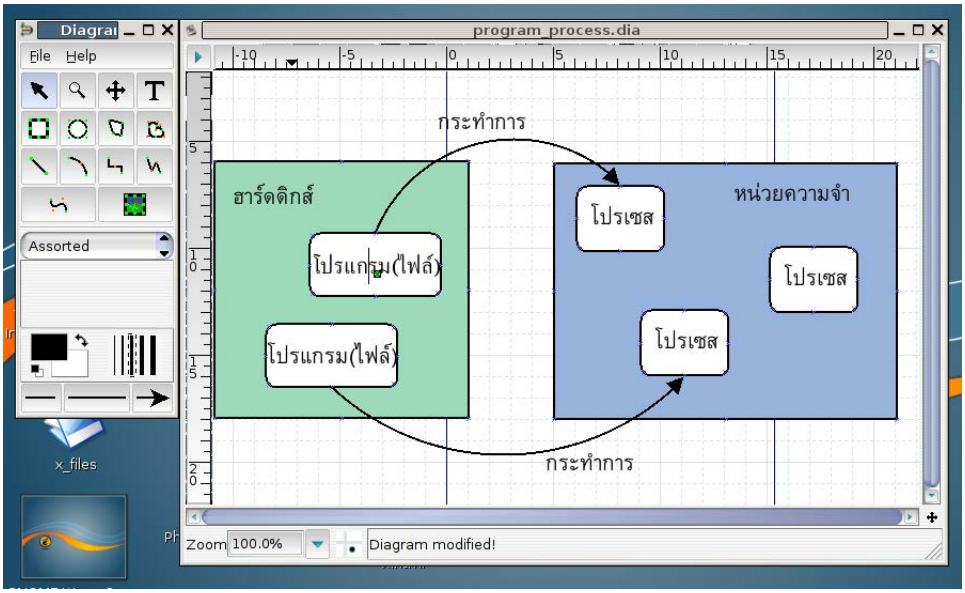
Inkscape เป็นโปรแกรมสำหรับวาดรูปแบบเวคเตอร์. ไฟล์ที่สร้างด้วย Inkscape เป็นฟอร์แมต SVG (Scalable Vector Graphic) ซึ่งเป็นไฟล์เท็กซ์บันทึกของในรูปของ XML.

ภาพที่สร้างด้วย Inkscape มีข้อดีที่เป็นเวคเตอร์สามารถย่อขยายได้โดยที่ภาพยังคงรายละเอียดเดิมไว้. นอกจากนั้นสามารถเอ็กซ์พอร์ตเป็นฟอร์แมตอื่น ๆเช่น eps หรือ png ได้ด้วย. Inkscape รุ่นใหม่สามารถกับโปรแกรมเทรส (trace) รูปภาพเช่น `postrace` ช่วยวาดรูปได้ง่ายขึ้นด้วย.

โปรแกรมประเภทเดียวกันกับ Inkscape เช่น `killustrator`, `karbon` ฯลฯ.



รูปที่ 8.19: โปรแกรม Inkscape.



รูปที่ 8.20: โปรแกรม Dia.

8.3.3 Dia

Dia เป็นโปรแกรมสำหรับวาดแผนภาพประกอบเอกสาร. ตัวโปรแกรมมีประเภทของแผนภาพเตรียมไว้ให้เช่นในประเภท UML จะมีเครื่องหมายต่าง ๆที่เกี่ยวกับการเขียน

แผนภาพ UML, ประเภทของ Circuit จะมีรูปตัวต้านทาน, ไดโอด ฯลฯเตรียมไว้ให้ใช้เป็นต้น. โปรแกรม รุ่นใหม่ ๆเช่น 0.94 สามารถเอ็กซ์พอร์ตรูปที่วาดเป็นฟอร์แมตเวคเตอร์อื่น ๆได้เช่น eps และแสดงฟอนต์ภาษาไทยได้ด้วย.

โปรแกรมประเภทเดียวกันกับ Dia เช่น `kivio`, `xfig`, `tgif`, `kdraw` ฯลฯ.

8.3.4 ImageMagick

ImageMagick เป็นไลบรารีภาษาคอมพิวเตอร์ต่าง ๆสำหรับประมวลผลรูปภาพและมีจุดเด่นที่รับรู้ประเภทรูปได้หลายฟอร์แมต. ในแพ็กเกจ ImageMagick จะมีโปรแกรมบรรทัดคำสั่งรวมที่สำคัญ ๆอยู่ด้วยได้แก่

- `display` — สำหรับแสดงรูปฟอร์แมตต่าง ๆในระบบ X วินโดว์. ตัวโปรแกรมจะมีเมนูสำหรับประมวลผลรูปภาพเช่น หมุนรูป, ปรับแสงสี, แปลงฟอร์แมตเป็นต้น. วิธีใช้ที่ง่ายที่สุดคือให้ชื่อไฟล์รูปภาพที่ต้องการแสดงผลเป็นอาร์กิวเมนต์ของคำสั่ง.

- `convert` — แปลงฟอร์แมตรูปภาพด้วยบรรทัดคำสั่ง (ตัวอย่างที่ 4.92). คำสั่งนี้ถ้าใช้งานอย่างจริงจังสามารถทำอะไรได้มากกว่าการแปลงฟอร์แมต, ให้อ่านรายละเอียดเพิ่มเติมจากแมนเพจ.

- `composite` — บรรทัดคำสั่งสำหรับสร้างประกอบรูป. เช่นการสร้างรูปใหม่โดยหาผลต่างของรูปสองรูปที่กำหนดเป็นต้น.

- `conjure` — ตัวแปรภาษา MSL (Magick Scripting Language) เป็นภาษาแบบ XML เขียนสคริปต์ประมวลผลรูปภาพ.

- `identify` — แสดงข้อมูลต่างของรูปที่ระบุเป็นอาร์กิวเมนต์เช่นฟอร์แมตรูปภาพและขนาดเป็นต้น. ถ้าใช้ตัวเลือก `-verbose` ประกอบจะแสดงรายละเอียดเพิ่มขึ้นกว่าปรกติ.

- `import` — จับหน้าจอแล้วบันทึกเป็นรูปภาพตามฟอร์แมตชื่อไฟล์ที่ระบุเป็นอาร์กิวเมนต์. ในกรณีที่ต้องการจับหน้าจอพื้นโต๊ะให้ใช้ตัวเลือก `-window root`. โปรแกรมประเภทเดียวกันเช่น `gnome-panel-screenshot` เป็นต้น.

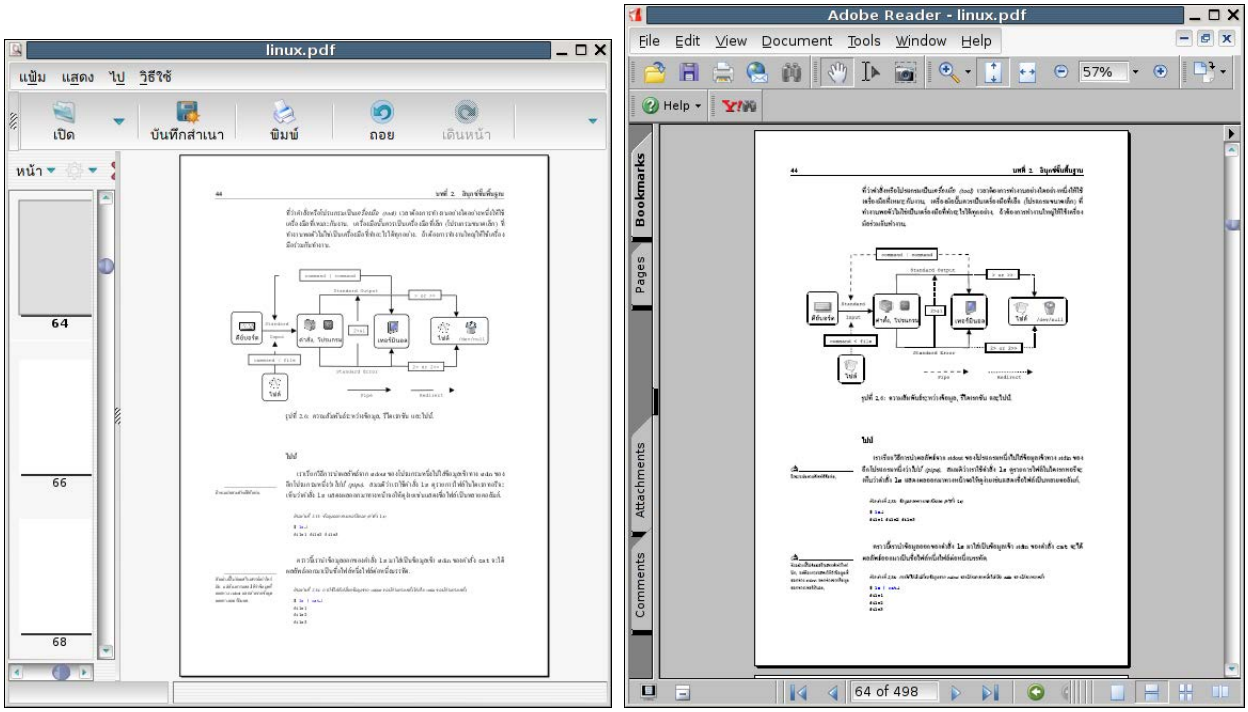
8.3.5 โปรแกรมดูรูปภาพ

`gthumb` เป็นโปรแกรมสำหรับดูรูปภาพ, ใช้ง่าย. วิธีการดูรูปมีหลายแบบเช่นดูรูปแบบเล็ก ๆที่อยู่ในไดเรกทอรี, ดูรูปเดี่ยว ๆ, ดูรูปเต็มหน้าจอ, และดูแบบสไลด์โชว์. นอกจากการดูรูปแล้วยังสามารถปรับแสงสี, ขนาดได้และเลือกรูปสร้างอัลบัมแสดงบนเว็บ, สร้างภาพดรรชนีจากรูปที่เลือกไว้.

โปรแกรมประเภทเดียวกันกับ gThumb เช่น `eog` (Eye Of GNOME), `xv`, `nautilus` ฯลฯ.

8.3.6 โปรแกรมดูเอกสาร

เอกสารที่เผยแพร่ทางอินเทอร์เน็ตในปัจจุบันมักจะเป็นฟอร์แมตในตระกูล PostScript ซึ่งรวมถึงเอกสารแบบ PDF ด้วย. เอกสารเหล่านี้สามารถเปิดดูด้วยตัวแปลภาษา PostScript ได้แก่โปรแกรม `gs` (GhostScript). โปรแกรมที่ใช้เปิดอ่านเอกสารเหล่านี้เป็นซอฟต์แวร์เสรีและมักจะเป็นฟรอนต์เอ็น (frontend) ของตัวแปลภาษา GhostScript เช่น `gv`, `ggv`, `gpdf`, `xpdf`, `kpdf` ฯลฯ. ส่วนโปรแกรมที่ไม่เปิดเผยรหัสแต่ใช้เปิดอ่านเอกสาร PDF ที่นิยมได้แก่ Adobe Acrobat Reader (`acroread`).



รูปที่ 8.21: โปรแกรม `gpdf` และ `acroread`.

8.4 มัลติมีเดีย

โปรแกรมที่เกี่ยวกับมัลติมีเดียในลินุกซ์มีให้ใช้มากขึ้นเรื่อย ๆและพัฒนาได้ดีขึ้น. ปัญหาหลักของโปรแกรมเหล่านี้ไม่ใช่ปัญหาทางเทคนิคแต่เป็นปัญหาเรื่องสิทธิบัตรเช่น *codec (compressor-decompressor)* ที่ใช้เปิดไฟล์ดูหนังจดสิทธิบัตรไว้และไม่สามารถใช้ได้อย่างเสรี. ดังนั้นดิสโทรบางค่ายจะไม่เตรียมแพ็กเกจสำหรับดูหนังให้, แต่ผู้ใช้สามารถดาว์นโหลดไบนารีหรือรหัสต้นฉบับได้จากเว็บไซต์ของผู้พัฒนาซอฟต์แวร์นั้น ๆโดยตรง.

codec ▶ การเข้าถอดรหัสข้อมูลเสียงหรือภาพในระบบดิจิตอล.

8.4.1 โปรแกรมฟังเพลง

โปรแกรมเล่นเพลงในยุคแรก ๆเป็นโปรแกรมบรรทัดคำสั่งคือโปรแกรม `mpg123` แต่ใช่ซอฟต์แวร์เสรีจึงมีการพัฒนาซอฟต์แวร์ใหม่ในชื่อ `mpg321` และเปิดเป็นซอฟต์แวร์

เสรี. ปัจจุบันโปรแกรมเล่นไฟล์เพลงดิจิตอลเปลี่ยนเป็นโปรแกรมแบบ GUI เสียส่วนใหญ่. โปรแกรมที่ได้ความนิยมเช่น `xmms`, `beep-media-player`, `rhythmbox` เป็นต้น. โปรแกรมเหล่านี้ใช้ฟังเพลงได้เหมือนกันแต่อาจจะแตกต่างกันตรงที่รูปร่างหน้า, ความสามารถต่าง ๆแล้วแต่โปรแกรม.

รูปที่ 8.22: โปรแกรม `xmms`.

รูปที่ 8.23: โปรแกรม `rhythmbox`.

8.4.2 โปรแกรมสร้างไฟล์เพลงดิจิตอล

ในลินุกซ์มีโรปแกรมสำหรับแปลงเพลงจาก CD ไปเป็นไฟล์ในฟอร์แมต MP3 (MPEG layer 3) หรือฟอร์แมตอื่น ๆด้วยโปรแกรม `cdparanoia` หรือ `cdda2wav`. โปรแกรมนี้เป็นโปรแกรมบรรทัดคำสั่งทำหน้าที่สกัดข้อมูลเพลงจากแผ่นเพลง CD ไปเป็นไฟล์ฟอร์แมต WAV หรือฟอร์แมตพื้นฐานอื่น ๆ. ไฟล์เพลงดิจิตอลที่สกัดจากโปรแกรมจะเป็นฟอร์แมต WAV ซึ่งไม่มีการบีบอัดข้อมูลทำให้ไฟล์ที่ได้มีขนาดใหญ่. ดังนั้นจึงมีความจำเป็นต้องบีบอัดข้อมูลเพลงเหล่านี้ต่อไปเป็นไฟล์ฟอร์แมตอื่น ๆเช่น MP3, OGG (Ogg Vobris), FLAC (Free Loose-less Audio Codec) ฯลฯ. โปรแกรมสำหรับแปลงไฟล์ WAV เป็นฟอร์แมตอื่น ๆเช่น flacen, lame, mp3enc??? ฯลฯ.

ในการใช้งานจริงมักจะใช้โปรแกรมแบบ GUI เช่น `grip` จัดการตั้งแต่การสกัดข้อมูลออกจากแผ่น CD และบีบอัดให้ด้วยซึ่งจะสะดวกกว่าการใช้โปรแกรมบรรทัดคำสั่ง. `grip`

เป็นโปรแกรมฟรอนต์เอ็นของโปรแกรมที่เกี่ยวข้องต่าง ๆเช่น `cdparanoia`, `lame` ฯลฯ และมีความสามารถดึงฐานข้อมูลของเพลงในแผ่น CD มาเขียนเป็นชื่อไฟล์เพลงดิจิตอลด้วย.

8.4.3 โปรแกรมดูหนัง

โปรแกรมดูหนัง, DVD หรือไฟล์วิดิโอเช่นไฟล์ AVI ในลินุกซ์มีอยู่หลายตัวเช่น `xine`, `vlc`, `totem`, `mplayer` ฯลฯ แล้วแต่เลือกใช้ตามชอบ. เนื่องจากโปรแกรมดูหนังเหล่านี้มักจะมีปัญหาเรื่องสิทธิบัตรทำให้ดิสโทรบางค่ายไม่เผยแพร่แพ็กเกจนั้น ๆแต่ผู้ใช้สามารถดาว์นโหลดซอฟต์แวร์และติดตั้งด้วยตัวเอง.

รูปที่ 8.24: ดูหนังด้วย `totem`.

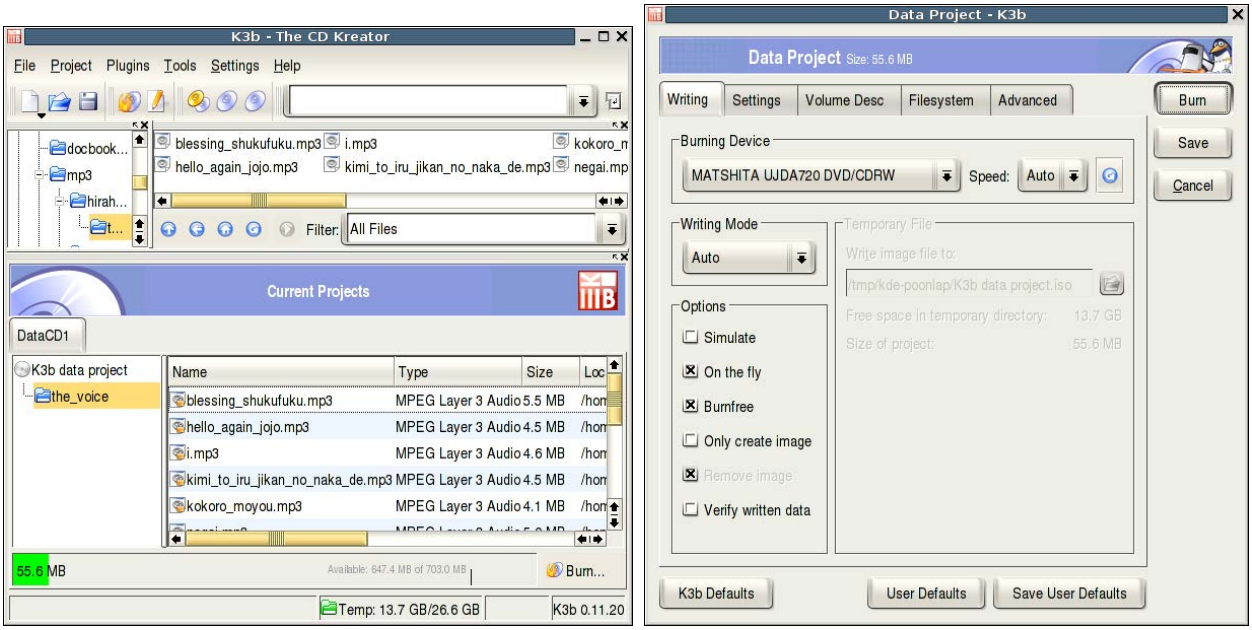
โปรแกรมอื่น ๆที่เกี่ยวกับวิดิโอได้แก่โปรแกรมแปลง*โคเดก (codec)*ของไฟล์วิดิโอ. โปรแกรมสำหรับแปลงโคเดกเช่น `ffmpeg`, `transcode` ฯลฯ. สำหรับการนำข้อมูลเข้าจากกล้องดิจิตอลวิดิโอใช้โปรแกรม `dvgrab`, `kino` แล้วแปลงเป็นไฟล์โคเดกอื่น ๆภายหลัง.

8.4.4 โปรแกรมเขียน CD, DVD

ขั้นตอนเขียน CD หรือ DVD ในระบบปฏิบัติการลินุกซ์โดยปรกติจะสร้างไฟล์อิมเมจ ISO ก่อน. ไฟล์อิมเมจ ISO สร้างด้วยคำสั่ง `mkisofs` รวมไฟล์หรือไดเรกทอรีต่าง ๆเข้าด้วยกันในไฟล์เดียวและมักจะตั้งชื่อส่วนขยายชื่อไฟล์เป็น `.iso` หรือ `.raw`. หลังจากที่ได้ไฟล์อิมเมจ ISO แล้วก็จะใช้โปรแกรม `cdrecord` ซึ่งเป็นโปรแกรมบรรทัดคำสั่งเขียนข้อมูลในไฟล์อิมเมจ ISO ลงในแผ่น CD. ถ้าเป็นการเขียนแผ่น DVD ก็จะใช้โปรแกรม `growisofs` หรือ `dvdrecord`.

การใช้งานจริงจะใช้โปรแกรมคำสั่งที่แนะนำก็ได้หรือใช้โปรแกรมแบบ GUI เพื่ออำนวยความสะดวกเช่นโปรแกรม `xcdroast`, `nautilus`, `k3b` เป็นต้น. ปัจจุบันโปรแกรมที่

นิยมและเป็นที่ยอมรับว่าใช้งานได้ดีคือ k3b. k3b ไม่ใช่โปรแกรมเขียน CD หรือ DVD ธรรมดา, โปรแกรมนี้สามารถสกัดข้อมูลหนังจากแผ่น DVD แล้วแปลงเป็นไฟล์ MPEG-4 ได้ด้วย.



รูปที่ 8.25: โปรแกรม k3b สำหรับเขียน CD หรือ DVD.

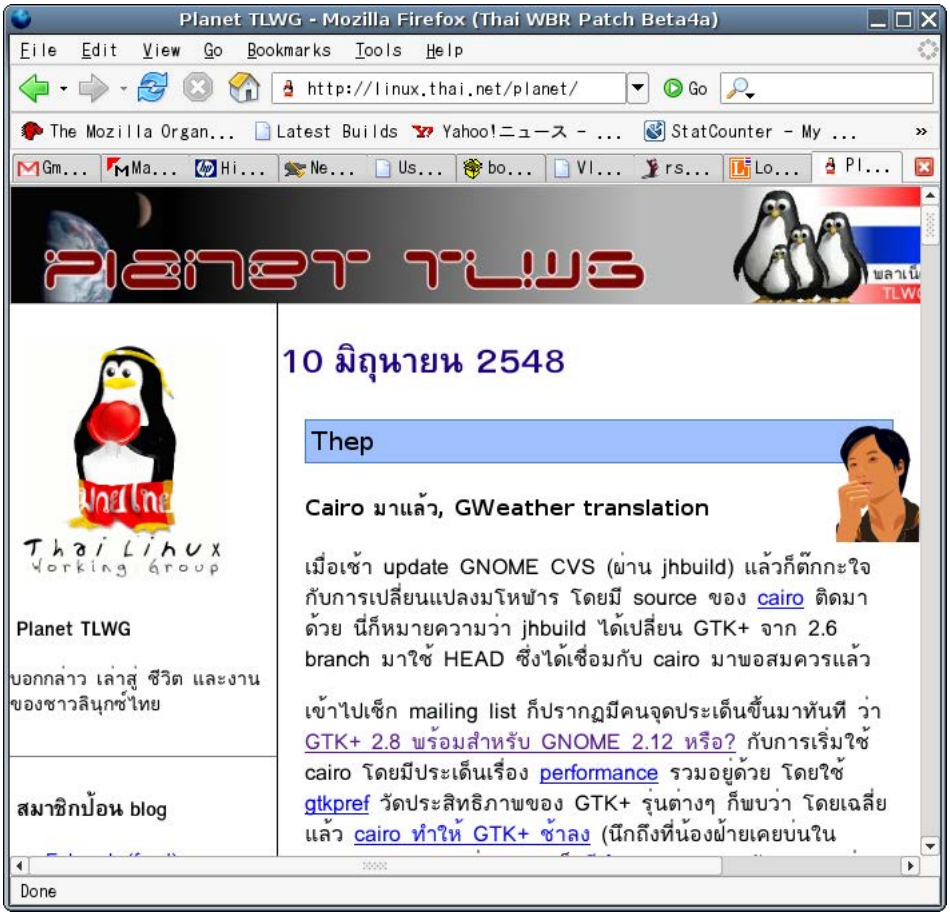
8.5 อินเทอร์เน็ต

8.5.1 เบราเซอร์

xmosaic เป็นเบราเซอร์ตัวแรก ๆในยุคเริ่มต้นของอินเทอร์เน็ต. โปรแกรมนี้เป็นเบราวเซอร์ที่ทำงานบนระบบ X วินโดว์ในระบบปฏิบัติการยูนิกซ์และต่อมาผู้สร้างพัฒนาเบราเซอร์ตัวต่อมาคือ Netscape และเผยแพร่รหัสต้นฉบับเป็น Mozilla ในที่สุดราวปี 1996?.

Mozilla ยังคงรูปร่างและคุณสมบัติต่าง ๆเหมือน Netscape เช่นมีโปรแกรมประกอบอื่น ๆนอกจากตัวเบราเซอร์เช่น โปรแกรมอ่านเมล, โปรแกรมช่วยเขียนโฮมเพจ ฯลฯ. ต่อมามีการแยกโปรแกรมเบราเซอร์ออกมาอย่างเดียวและเปลี่ยนชื่อมาเป็น Firefox ที่รู้จักกันอย่างกว้างขวางทุกวันนี้. Firefox เป็นเบราเซอร์ที่ออกแบบแยกส่วนประกอบการทำงานต่าง ๆเช่น XP (Cross Platform), XUI (Extensible User Interface), Gekko ฯลฯ. Gekko เอ็นจินเป็นส่วนสำหรับแสดงผลข้อมูล HTML ทางหน้าจอและโปรแกรมอื่น ๆ สามารถใช้ Gekko ในโปรแกรมของตัวเองด้วย.

สภาพแวดล้อมเดสก์ท็อปต่าง ๆมีเบราเซอร์ของตัวเองเช่นใน KDE มี Konqueror, ใน GNOME มี Epiphany หรือ Galeon. เบราเซอร์เหล่านี้ใช้ทูลคิตของสภาพแวดล้อมเดสก์ท็อปที่สังกัดจึงทำงานร่วมกับโปรแกรมอื่น ๆได้ดีในสภาพแวดล้อมเดสก์ท็อปนั้น ๆ.



รูปที่ 8.26: Firefox ที่รองรับการตัดคำภาษาไทย.

Konqueror ใช้เอ็นจินการแสดงผล KHTML, Epiphany และ Galeon ใช้เอ็นจิน Gekko ของ Mozilla.

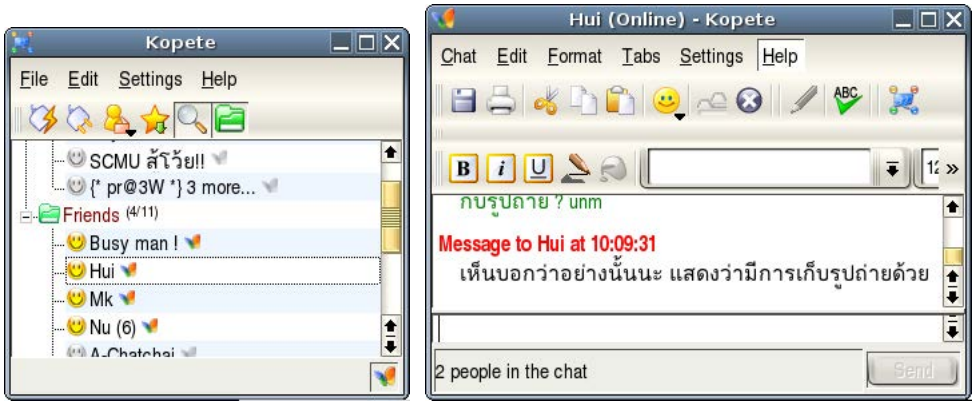
Konqueror เป็นเบราเซอร์อินเทอร์เน็ตและไฟล์เบราเซอร์ภายในตัวในสภาพแวดล้อมเดสก์ท็อป KDE. Konqueror ยังสามารถติดต่อโปรโตคอลประเภทต่างๆเช่น smb โดยเขียนที่อยู่เป็น `smb:///hostname` ได้ด้วย.

สำหรับคนที่ไม่ต้องการใช้ GUI ในการแสดงผลอาจจะใช้เบราเซอร์ `lynx` หรือ `w3m` แทน. โปรแกรมเหล่านี้จะรันในเทอร์มินอล.

8.5.2 Instant Messenger Service

ลินุกซ์สามารถใช้บริการ Instant Messenger Service ของ MSN Messenger, AOL หรือ Yahoo Messenger ได้โดยใช้โปรแกรมเช่น `kopete`, `gaim` ฯลฯ. โปรแกรม Messaging Service ในลินุกซ์เช่น `kopete` มีคุณสมบัติรองรับบริการหลายตัวคือติดต่อกับ MSN, AOL, Yahoo ได้ด้วยโปรแกรมเดียว. โปรแกรมเหล่านี้ยังมีข้อจำกัดบางอย่างเช่นยังไม่สามารถสื่อสารด้วยเสียงหรือภาพจากกล้องวิดีโอแต่ใช้สื่อสารด้วยการพิมพ์แป้นพิมพ์หรือส่งไฟล์ให้ซึ่งกันและกันได้. บางครั้งโปรโตคอลที่ใช้ในการสื่อสารเช่น MSN Mes-

senger มีการเปลี่ยนแปลง, โปรแกรมเหล่านี้ไม่สามารถใช้ได้จนกว่าจะอัปเดทตัวโปรแกรมเป็นรุ่นล่าสุด.



รูปที่ 8.27: kopete, โปรแกรม Instant Messenger Service ที่รองรับหลายเครือข่ายในตัวโปรแกรมเดียว.

Jabber เป็นระบบ Instant Messenger Service แบบโอเพนซอร์ส, ผู้ใช้สามารถสร้างเซิร์ฟเวอร์ด้วยตัวเองโดยไม่ต้องใช้เน็ตเวิร์กปิดของ Instant Messenger Service อื่น ๆ.

8.5.3 โปรแกรมช่วยดาว์นโหลด

การดาว์นโหลดไฟล์ทางอินเทอร์เน็ตคราวละไม่มากอาจจะใช้เบราเซอร์ดาว์นโหลด. แต่สำหรับการดาว์นโหลดโดยอัตโนมัติ, เขียนเป็นสคริปต์ หรือดาว์นโหลดหลายคอนเนกชันต้องใช้โปรแกรมพิเศษต่างหาก.

wget เป็นโปรแกรมบรรทัดคำสั่งสำหรับดาว์นโหลดไฟล์ด้วยโปรโตคอล HTTP หรือ FTP. การดาว์นโหลดด้วยคำสั่ง wget มีข้อดีที่ใช้ดาว์นโหลดไฟล์แบบรีเคอร์ซีฟ (recursive) ทั้งไดเรกทอรีได้, ดาว์นโหลดแบบมิรเรอร์ (mirror), เลือกดาว์ฯโหลดเฉพาะไฟล์ที่มีตัวขยายชื่อไฟล์ตามต้องการเช่น ดาว์นโหลดไฟล์ .pdf อย่างเดียว ฯลฯ. การใช้งานโดยละเอียดสามารถอ่านได้จาก man wget.

ตัวอย่างที่ 8.4: วิธีใช้ wget

```
$ SITE=http://linux.thai.net/~poonlap/images↵
$ wget -nd $SITE/gdm.png↵          ← ดาว์นโหลดไฟล์ gdm.png ไว้ที่ไดเรกทอรีปัจจุบัน
$ wget -r $SITE↵                        ← ดาว์นโหลดไฟล์ทุกไฟล์ทั้งไดเรกทอรี
$ wget -nd -r -A .gif $SITE↵                ← ดาว์นโหลดเฉพาะไฟล์ .gif
```

Prozilla เป็นโปรแกรมดาว์นโหลดที่สามารถสร้างคอนเนกชันเวลาดาว์นโหลดได้หลายตัวพร้อม ๆกัน. โปรแกรมประเภทนี้เรียกว่า download accelerator ช่วยให้ดาว์นโหลดไฟล์ที่ต้องการได้เร็วขึ้นโดยการเพิ่มคอนเนกชัน. proz เป็นโปรแกรม Prozilla ที่ทำงานในเทอร์มินอลและมีฟรอนต์เอ็นแบบ GUI ด้วยแต่อาจจะต้องติดตั้งต่างหาก.

8.5.4 Video Conference

สำหรับระบบที่มีกล้องเช่น USB และมีระบบเสียงและไมโครโฟนพร้อมสามารถใช้โปรแกรมประชุมข้ามเครือยข่ายได้ด้วย gnome-meetinggnome-meeting. Gnome meeting เป็นโปรแกรมสร้างตามมาตรฐาน H.326 ดังนั้นจึงสามารถใช้ได้กับอุปกรณ์ video conference ที่ใช้มาตรฐานเดียวกันหรือโปรแกรม NetMeeting ในระบบปฏิบัติการวินโดวส์ได้.

8.6 โปรแกรมรับส่งอีเมล

โปรแกรมรับส่งอีเมลในยุคแรก ๆเรียบง่ายและเป็นบรรทัดคำสั่งเช่นโปรแกรม mail หรือ mailx. ปัจจุบันคำสั่ง mail ไม่นิยมใช้แล้วยกเว้นจะสั่งส่งเมลด้วยบรรทัดคำสั่ง.

โปรแกรมรับส่งเมลที่ได้รับความนิยมในช่วงต่อมาคือ pine, Rmail, MH ฯลฯ. โปรแกรมเหล่านี้เช่น pine ทำงานในเทอร์มินอลเป็นเมนูอ่านหรือเขียนเมล, Rmail, MH เป็นแพ็กเกจรับส่งเมลที่ใช้ใน Emacs เป็นต้น. สำหรับคนที่นิยมใช้เทอร์มินอลมักจะใช้ mutt ซึ่งเป็นโปรแกรมรับส่งเมลใช้ในเทอร์มินอลคล้าย pine แต่มีความสามารถต่าง ๆ เหนือกว่า.

ปัจจุบันโปรแกรมรับส่งเมลแบบ GUI มีให้เลือกมากมายเช่น Thunderbire, Evolution, Kmail ฯลฯ ตามความชอบพอของผู้ใช้แต่ละคน.

8.7 โปรแกรมมิ่ง

การพัฒนาซอฟต์แวร์แบบดั้งเดิมในระบบระบบปฏิบัติการลินุกซ์นิยมใช้บรรณาธิกรณ์เช่น Vi, Emacs เขียนรหัสต้นฉบับ, สร้างไฟล์ Makefile เป็นสคริปต์สร้างโปรแกรมด้วยคอมไพเลอร์โดยอัตโนมัติ. IDE (Integrated Deveopment Environment) เป็นแอพพลิเคชันที่รวมการขั้นตอนการพัฒนาซอฟต์แวร์ต่าง ๆตั้งแต่การควบคุมโปรเจค, บรรณาธิกรณ์สำหรับเขียนรหัสต้นฉบับ, การออกแบบอินเทอร์เฟสของโปรแกรม, การคอมไพล์, ดีบักเข้าด้วยกัน. โปรแกรม IDE ในลินุกซ์เริ่มมีบทบาทมากขึ้นเรื่อยและโปรแกรมที่นิยมใช้กันได้แก่ Kdevelop, Anjuta, Monodevelop ฯลฯ.

8.8 วิทยาศาสตร์

ในลินุกซ์มีโปรแกรมสำหรับใช้งานคำนวณ, สร้างกราฟ, สถิติ เหมาะสำหรับนักศึกษาและนักเรียนทั่วไป. โปรแกรมง่าย ๆทีตั้งแต่เครื่องคิดเลขวิทยาศาสตร์จนถึงโปรแกรมคำนวณข้อมูลสถิติเช่น R.

สำหรับการเขียนกราฟคณิตศาสตร์ทั่วไปอาจจะใช้ gnuplot เขียนกราฟจากสมการ, หรือข้อมูลดิบ. ส่วนการคำนวณด้วยวิทยาศาสตร์, เมตริก, สถิติ จะใช้โปรแกรม R หรือ octave เป็นต้น.

8.9 รีโมตเดสก์ท็อป

การติดต่อใช้ระบบปฏิบัติการอื่น ๆเช่นระบบปฏิบัติการวินโดวส์สามารถใช้ `rdesktop` ติดต่อขอหน้าจอล็อกอินของเครื่องวินโดวส์ที่อยู่ในเครือข่าย. `rdesktop` เป็นทางเลือกแทน *VNC (Virtual Network Computing)* กรณีที่ต้องการล็อกอินเครื่องวินโดวส์ที่มี Terminal Service ทำงานอยู่. การใช้ `rdesktop` สะดวกในกรณีที่ใช้ลินุกซ์ประจำแล้วต้องการใช้วินโดวส์ของเครื่องคอมพิวเตอร์อีกเครื่องหนึ่งผ่านทางเครือข่าย. ผู้ใช้สามารถเลือกขนาดหน้าจอเป็นหน้าต่างแอพพลิเคชันต่างหากหรือทำงานเต็มหน้าต่างแล้วเปลี่ยนหน้าจอไปมา.

สำหรับการรีโมตเดสก์ท็อปหน้าจอลินุกซ์หรือระบบปฏิบัติการอื่น ๆจะใช้ VNC. ในระบบ VNC จะแบ่งออกเป็นตัวแชร์หน้าจอและตัวดูหน้าจอ. โปรแกรมสำหรับแชร์หน้าจอ `vino-server`, x0vncserver ฯลฯ. โปรแกรมดูหน้าจอด้วย VNC เช่น `vncviewer`, `krdc` ฯลฯ.

8.10 พจนานุกรม

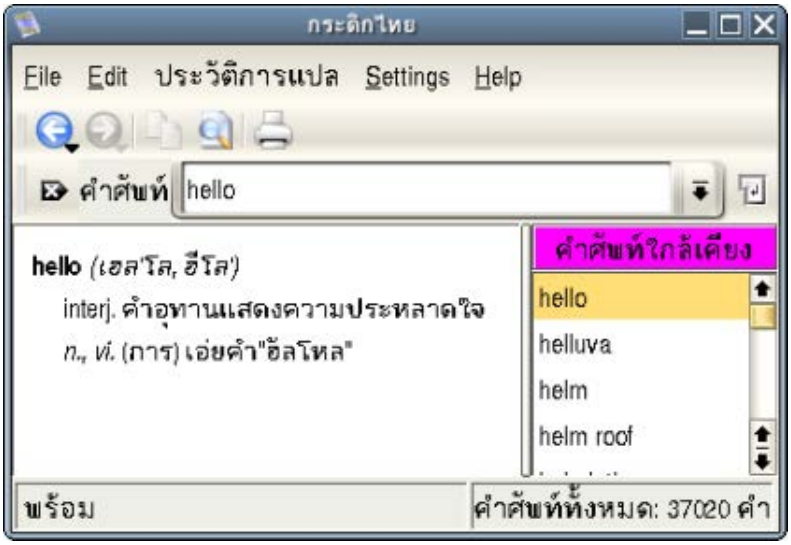
โปรแกรมพจนานุกรมในลินุกซ์มีหลายชนิดแบบที่เป็นแอพเพล็ตฝังอยู่ในพาเนลและเป็นโปรแกรมหน้าต่างเช่น `gdict`, `kdict`. โปรแกรมเหล่านี้มักเป็นโปรแกรมพจนานุกรมอธิบายภาษาอังกฤษด้วยภาษาอังกฤษ.

โปรแกรมพจนานุกรมแปลศัพท์อังกฤษไทยมีหลายตัวและพัฒนาโดยอาสาสมัคร.

- Lexitron — เป็นโปรแกรมพจนานุกรมที่สร้างด้วย Java โดย NECTEC.
- `kdicthai` — โปรแกรมนี้สร้างดัดแปลงต่อจากโปรแกรม `kdict` พัฒนาโดยคุณ Donga. ข้อมูลของตัวโปรแกรมมาจากพจนานุกรม DictHope โดย ??? ซึ่งเป็นโปรแกรมบนวินโดวส์.
- `cetdict` — โปรแกรมแปลศัพท์อังกฤษไทยด้วยบรรทัดคำสั่งหรือโต้ตอบในเทอร์มินอล. มีความสามารถเติมเต็มคำหรือแสดงรายการคำที่เป็นไปได้. ข้อมูลคำแปลมาจาก DictHope เช่นเดียวกับ `kdictthai`
- ข้อมูลศัพท์อังกฤษไทยสำหรับ dictd เซิร์ฟเวอร์ — ไม่ใช้โปรแกรมแต่เป็นข้อมูลคำศัพท์ใช้กับเซิร์ฟเวอร์ `dictd` ซึ่งเป็นเซิร์ฟเวอร์พจนานุกรมให้ไคลเอ็นต์เช่น `gdict`, `kdict` ติดต่อแปลคำผ่านทางเครือข่าย. ผู้ใช้สามารถเลือกใช้โปรแกรมอะไรก็ได้ที่รองรับโปรโตคอล dictd และปรับแต่งโปรแกรมให้ใช้ศัพท์จากเซิร์ฟเวอร์ที่เตรียมไว้.

8.11 สรุปท้ายบท

- ในบทนี้เป็นการแนะนำโปรแกรมใช้งานลินุกซ์อย่างคร่าว ๆโดยเน้นอธิบายการใช้งานบรรณาธิกรณ์ Vi และ Emacs.



รูปที่ 8.28: โปรแกรมพจนานุกรมแปลอังกฤษไทย `kdictthai`.

- ผู้ใช้ควรจะเรียนรู้การใช้งานบรรณาธิกรณ์อะไรก็ได้ตัวหนึ่งที่มีใช้ในลินุกซ์ทุกดิสโทร โดยปริยายเช่น Vi.
- โปรแกรมต่าง ๆอาจจะเป็นโปรแกรมบรรทัดคำสั่งและมีโปรแกรมแบบ GUI เป็นฟรอนต์เอ็น. สำหรับการใช้งานทั่วไปควรใช้โปรแกรมฟรอนต์เอ็นต์จะสะดวกกว่า.
- โปรแกรมใช้งานในลินุกซ์ยังมีอีกมากมาย, ผู้ใช้อาจจะหาโปรแกรมหรือแพ็กเกจที่ต้องการใช้เพิ่มเติมได้จากอินเทอร์เน็ต.
- โปรแกรมติดตั้งแพ็กเกจบางระบบมีคำสั่งสำหรับหาโปรแกรมหรือแยกประเภทแพ็กเกจตามการใช้งานให้แล้ว. ผู้ใช้อาจจะใช้ระบบติดตั้งแพ็กเกจช่วยหาโปรแกรมที่ต้องการ.

ภาคผนวก ก
รหัสอักขระ ASCII

ตารางที่ ก.1: รหัสอักขระ ASCII

ค่าฐานสิบ	ค่าฐานสิบหก	ค่าฐานแปด	อักขระ	
0	0x00	000	NUL*	null
1	0x01	001	SOH*	start of heading
2	0x02	002	STX*	start of text
3	0x03	003	ETX*	end of text
4	0x04	004	EOT*	end of transmission
5	0x05	005	ENQ*	enquiry
6	0x06	006	ACK*	acknowledge
7	0x07	007	BEL*	bell
8	0x08	010	BS*	backspace
9	0x09	011	TAB*	horizontal tab
10	0x0A	012	LF*	new line, line feed
11	0x0B	013	VT*	vertical tab
12	0x0C	014	FF*	new page, form feed
13	0x0D	015	CR*	carriage return
14	0x0E	016	SO*	shift out
15	0x0F	017	SI*	shift in
16	0x10	020	DLE*	data line escape
17	0x11	021	DC1*	device control 1
18	0x12	022	DC2*	device control 2
19	0x13	023	DC3*	device control 3
20	0x14	024	DC4*	device control 4
21	0x15	025	NAK*	negative acknowledge

ต่อหน้าถัดไป

ต่อจากหน้าที่แล้ว

ค่าฐานสิบ	ค่าฐานสิบหก	ค่าฐานแปด	อักขระ	
22	0x16	026	SYN*	synchronous idle
23	0x17	027	ETB*	end of transmission block
24	0x18	030	CAN*	cancel
25	0x19	031	EM*	end of medium
26	0x1A	032	SUB*	substitute
27	0x1B	033	ESC*	escape
28	0x1C	034	FS*	file separator
29	0x1D	035	GS*	group separator
30	0x1E	036	RS*	record separator
31	0x1F	037	US*	unit separator
32	0x20	040	SP	space
33	0x21	041	!	exclamation mark
34	0x22	042	"	(double) quotation mark
35	0x23	043	#	number sign
36	0x24	044	$	dollar sign
37	0x25	045	%	percent sign
38	0x26	046	&	ampersand
39	0x27	047	'	apostrophe, single quote mark
40	0x28	050	(	left parenthesis
41	0x29	051	)	right parenthesis
42	0x2A	052	*	asterisk
43	0x2B	053	+	plus sign
44	0x2C	054	,	comma
45	0x2D	055	−	minus sign, hyphen
46	0x2E	056	.	period, decimal point, full stop
47	0x2F	057	/	slash, virgule, solidus
48	0x30	060	0	digit 0
49	0x31	061	1	digit 1
50	0x32	062	2	digit 2
51	0x33	063	3	digit 3
52	0x34	064	4	digit 4
53	0x35	065	5	digit 5
54	0x36	066	6	digit 6

ต่อหน้าถัดไป

ต่อจากหน้าที่แล้ว

ค่าฐานสิบ	ค่าฐานสิบหก	ค่าฐานแปด	อักขระ	
55	0x37	067	7	digit 7
56	0x38	070	8	digit 8
57	0x39	071	9	digit 9
58	0x3A	072	:	colon
59	0x3B	073	;	semicolon
60	0x3C	074	<	less-than sign
61	0x3D	075	=	equal sign
62	0x3E	076	>	greater-than sign
63	0x3F	077	?	question mark
64	0x40	100	@	commercial at sign
65	0x41	101	A	capital A
66	0x42	102	B	capital B
67	0x43	103	C	capital C
68	0x44	104	D	capital D
69	0x45	105	E	capital E
70	0x46	106	F	capital F
71	0x47	107	G	capital G
72	0x48	110	H	capital H
73	0x49	111	I	capital I
74	0x4A	112	J	capital J
75	0x4B	113	K	capital K
76	0x4C	114	L	capital L
77	0x4D	115	M	capital M
78	0x4E	116	N	capital N
79	0x4F	117	O	capital O
80	0x50	120	P	capital P
81	0x51	121	Q	capital Q
82	0x52	122	R	capital R
83	0x53	123	S	capital S
84	0x54	124	T	capital T
85	0x55	125	U	capital U
86	0x56	126	V	capital V
87	0x57	127	W	capital W
88	0x58	130	X	capital X

ต่อหน้าถัดไป

ต่อจากหน้าที่แล้ว

ค่าฐานสิบ	ค่าฐานสิบหก	ค่าฐานแปด	อักขระ	
89	0x59	131	Y	capital Y
90	0x5A	132	Z	capital Z
91	0x5B	133	[	left square bracket
92	0x5C	134	\	backslash, reverse solidus
93	0x5D	135	]	right square bracket
94	0x5E	136	^	spacing circumflex accent
95	0x5F	137	_	spacing underscore, low line, horizontal bar
96	0x60	140	`	spacing grave accent, back apostrophe, back quote
97	0x61	141	a	small a
98	0x62	142	b	small b
99	0x63	143	c	small c
100	0x64	144	d	small d
101	0x65	145	e	small e
102	0x66	146	f	small f
103	0x67	147	g	small g
104	0x68	150	h	small h
105	0x69	151	i	small i
106	0x6A	152	j	small j
107	0x6B	153	k	small k
108	0x6C	154	l	small l
109	0x6D	155	m	small m
110	0x6E	156	n	small n
111	0x6F	157	o	small o
112	0x70	160	p	small p
113	0x71	161	q	small q
114	0x72	162	r	small r
115	0x73	163	s	small s
116	0x74	164	t	small t
117	0x75	165	u	small u
118	0x76	166	v	small v
119	0x77	167	w	small w
120	0x78	170	x	small x

ต่อหน้าถัดไป

ต่อจากหน้าที่แล้ว

ค่าฐานสิบ	ค่าฐานสิบหก	ค่าฐานแปด	อักขระ	
121	0x79	171	y	small y
122	0x7A	172	z	small z
123	0x7B	173	{	left brace (curly bracket)
124	0x7C	174	|	vertical bar
125	0x7D	175	}	right brace (curly bracket)
126	0x7E	176	~	tilde accent
127	0x7F	177	DEL*	delete

* เครื่องหมายควบคุม (control character)

ภาคผนวก ข
สรุปคำสั่ง, โปรแกรม

ในบทนี้จะเป็นการสรุปการใช้โปรแกรมคำสั่งที่สำคัญและคำสั่งที่ใช้กันบ่อยในลินุกซ์. ตัวเลือกต่าง ๆที่แสดงร่วมกับคำสั่งนั้นเป็นตัวเลือกที่ใช้บ่อย, เป็นตัวเลือกแค่ส่วนหนึ่งของตัวเลือกทั้งหมดของคำสั่งนั้น ๆ. สำหรับผู้ที่ต้องการศึกษาตัวเลือกทั้งหมดที่ใช้ได้ให้ใช้คำสั่ง `man` หรือ `info` หรือดูเอกสารกำกับของโปรแกรมนั้น ๆ.

วิธีดูการใช้งาน

- ตัวอักษรธรรมดาหมายถึงชื่อคำสั่ง, ชื่อตัวเลือก.
- อักษรตัวเอียงแสดงสิ่งที่ผู้ใช้ต้องเขียนเองเช่น *filename* หมายถึงชื่อไฟล์ซึ่งจะเป็นชื่ออะไรก็ได้แล้วแต่ผู้ใช้กำหนด.
- สิ่งที่อยู่ในเครื่องหมายวงเล็บเหลี่ยม `[ ]` หมายถึงเป็นตัวเลือก, จะใช้หรือไม่ใช้ก็ได้.

ข.1 ประมวลผลข้อมูลในไฟล์

bzip2

```
bzip2 [-cdtvk] [-] [file ...]
```

บีบอัดหรือขยายไฟล์.

- `-c` ส่งข้อมูลที่บีบอัดแล้วออกทาง stdout แทนที่จะบันทึกลงไฟล์.
- `-d` ขยายไฟล์. ถ้าตัวเลือกนี้ใช้ร่วมกับตัวเลือก `-c` ก็จะแสดงข้อมูลที่ขยายแล้วออกทาง stdout.
- `-k` เก็บ (keep) ไฟล์เก่าไว้ในกรณีที่คลายไฟล์บีบอัดออกแล้วจะไม่ลงไฟล์เดิมทิ้ง.
- `-t` ทดสอบไฟล์ที่บีบอัดแล้วว่าถูกต้องหรือไม่.

`-v`	ย่อมาจาก verbose. ใช้แสดงสถานะการทำงาน. โดยปรกติจะบอกสัดส่วนของข้อมูลที่บีบอัดแล้วเทียบกับข้อมูลเดิม.
`-`	ตัวเลือกสำหรับอ่านข้อมูลเข้าจาก stdin. ในกรณีนี้จะเขียนข้อมูลที่บีบอัดแล้วทาง stdout.
`file`	ไฟล์ที่ต้องการบีบอัด.

cat

```
cat [file ...]
```

ชื่อคำสั่งมาจากคำว่า concatenate, คำสั่งสำหรับรวมไฟล์หรือข้อมูลให้เป็นไฟล์หรือข้อมูลเดียว. หลักการทำงานของ `cat` คืออ่านข้อมูลเข้ามาอย่างไรก็ส่งข้อมูลออกไปอย่างนั้น. ข้อมูลออกที่ stdout.

`file ...`	ชื่อไฟล์ที่เป็นข้อมูลเข้า. ถ้าไม่ระบุจะรับข้อมูลจาก stdin.
`-n`	เพิ่มหมายเลขบรรทัดให้แต่ละบรรทัด.

cksum

```
cp [file] ...
```

คำนวณค่า cyclic redundancy check (CRC). ใช้สำหรับตรวจสอบข้อมูลว่าถูกต้องหรือไม่. คำสั่งที่คล้ายกับคำสั่งนี้คือ `md5sum`.

comm

```
comm [-123] file1 file2
```

แสดงส่วนที่เหมือนกัน, หรือส่วนเฉพาะของไฟล์.

`-1`	ไม่แสดงส่วนที่เป็นข้อมูลเฉพาะของไฟล์ที่หนึ่ง.
`-2`	ไม่แสดงส่วนที่เป็นข้อมูลเฉพาะของไฟล์ที่สอง.
`-3`	ไม่แสดงส่วนที่เหมือนกันของไฟล์ทั้งสอง.

compress

```
compress [-cvdt] [filename ...]
```

บีบอัดหรือขยายข้อมูล.

`-c` ส่งข้อมูลที่บีบอัดแล้วออกทาง stdout แทนที่จะบันทึกลงไฟล์.

`-d` ขยายไฟล์. ถ้าตัวเลือกนี้ใช้ร่วมกับตัวเลือก `-c` ก็จะแสดงข้อมูลที่ขยายแล้วออกทาง stdout.

`-t` ทดสอบไฟล์ที่บีบอัดแล้วว่าถูกต้องหรือไม่.

`-v` ย่อมาจาก verbose. ใช้แสดงสถานะการทำงาน. โดยปรกติจะบอกสัดส่วนของข้อมูลที่บีบอัดแล้วเทียบกับข้อมูลเดิม.

cmp

```
cmp [-s] file1 file2
```

ตรวจสอบเนื้อหาของไฟล์ว่ามีความแตกต่างหรือไม่.

`-s`	ไม่แสดงข้อความใด ๆบนหน้าจอไม่ว่าจะมีความแตกต่างหรือไม่ก็ตาม. ผู้ใช้จะตรวจสอบว่ามีความแตกต่างหรือไม่จากตัวแปรสถานะจบการทำงาน.
file1, *file2*	ชื่อไฟล์ที่ต้องการตรวจสอบ.

cut

```
cut [-f F] [-d s] [--output-delimiter=string] [file] ...
```

ตัดคอลัมภ์ที่ต้องการจากข้อมูลเป็นบรรทัด ๆ.

`-f` *F*	ระบุคอลัมน์ *F* (ตัวเลข) ที่ต้องการ. ถ้าเป็นคอลัมน์ที่ต่อเนื่องกันสามารถใช้เครื่องหมาย - เช่น 1-3 หมายถึงคอลัมน์ที่ 1 ถึง 3. ถ้าเป็นคอลัมน์ที่ไม่ต่อเนื่องสามารถใช้เครื่องหมาย , เช่น 1,3 หมายถึงคอลัมน์ที่ 1 และ 3.
`-d` *s*	คำหรืออักขระที่เป็นตัวแบ่งคอลัมภ์. ถ้าไม่ระบุจะถือว่า Tab เป็นตัวแบ่งคอลัมภ์โดยปริยาย.
`--output-delimiter=`*string*	ใช้สายอักขระ *string* เป็นตัวแบ่งคอลัมน์เวลาแสดงผล. ถ้าไม่ระบุตัวเลือกนี้จะใช้ตัวแบ่งที่กำหนดโดยตัวเลือก `-d` หรือ tab.

dd

```
dd [if=infile] [of=outfile] [bs=s] [count=c]
```

ชื่อคำสั่งมาจากคำว่า convert and copy file แต่ตอนนั้นมีโปรแกรมชื่อ `cc` อยู่แล้วจึงเลื่อนอักษรหนึ่งตัวเป็น `dd`. ใช้ก็อปปี้ไฟล์และแปลงข้อมูล. ถ้าไม่มีอาร์กิวเมนต์จะก็อปปี้ข้อมูลจาก stdin ไป stdout.

`if=`*`infile`*	ชื่อไฟล์ที่สำหรับข้อมูลเข้า.
`of=`*`outfile`*	ชื่อไฟล์ที่สำหรับข้อมูลออก.
`bs=`*`s`*	*`s`* คือตัวเลขขนาดของ block size ที่ต้องการใช้อ่านและเขียน. หน่วยที่ใช้เป็น b (block) = 512 ไบต์, k (kilo byts), c (character, byte) ฯลฯ.
`count=`*`c`*	*`c`* คือตัวเลขจำนวนของ block size (จำนวนครั้ง) ที่ต้องการก็อปปี้.

df

```
df [-skmh] [file ...]
```

แสดงจำนวนพื้นที่ที่ถูกใช้งานตามระบบไฟล์ที่ mount. ในกรณีที่มีชื่อไฟล์เป็นอาร์กิวเมนต์, จะแสดงพื้นที่ระบบไฟล์ที่ไฟล์นั้นอยู่.

`-s`	สรุปผลลัพธ์รวม. ไม่แสดงรายละเอียด.
`-k`	แสดงหน่วยของพื้นที่เป็นกิโบไบต์.
`-m`	แสดงหน่วยของพื้นที่เป็นเมกกาไบต์.
`-h`	แสดงหน่วยของพื้นที่ที่มนุษย์อ่านแล้วเข้าใจง่าย.
`file ...`	ชื่อไฟล์หรือไดเรกทอรี.

diff

```
diff [-Naru] file1 file2
```

คำสั่งนี้ต้องการอาร์กิวเป็นไฟล์สองไฟล์โดยแสดงความแตกต่างของจากไฟล์แรก (*`file1`*) กับไฟล์หลัง (*`file2`*). เวลาสร้าง patch มักจะใช้ตัวเลือก `-Naur`.

`-N`	ในกรณีที่เปรียบเทียบไฟล์ต่าง ๆที่อยู่ในไดเรกทอรี, ถ้าในไดเรกทอรีหนึ่งมีไฟล์แต่อีกไดเรกทอรีหนึ่งไม่มีไฟล์จะถือว่าเป็นการเปรียบเทียบไฟล์ใหม่. ไม่ทำให้เกิด error.
`-a`	เปรียบเทียบไฟล์แต่ละไฟล์เสมือนเป็นไฟล์เท็กซ์. ใช้เปรียบเทียบไฟล์ไบนารีและแสดงผลต่างให้.
`-r`	ใช้เปรียบเทียบไดเรกทอรีและไฟล์ที่อยู่ในไดเรกทอรีไปเรื่อย ๆ (recursive).
`-u`	
`file1, file2`	ชื่อไฟล์ทั้งสองที่ต้องการแสดงผลต่าง.

expand

```
expand [-i] [--tabs=N] [file] ...
```

เปลี่ยน tab ทุกตัวที่อยู่ในไฟล์ให้เป็น space.

`-i`	เปลี่ยน tab ที่อยู่ต้นบรรทัดเท่านั้น. `-i` ย่อมาจาก `--initial`.
`--tabs=`*N*	กำหนดให้ tab หนึ่งตัวแทนด้วยจำนวน (*N*) space ตามต้องการ. ถ้าไม่กำหนดจะถือว่า tab หนึ่งตัวแทนด้วย space แปดตัว.
`-p`	สงวนสิทธิ์การใช้ไฟล์ที่ตั้งไว้ให้เหมือนเดิม.
`-v`	แสดงชื่อไฟล์ที่ก็อปปี้.
file	ชื่อไฟล์.

fmt

```
fmt [-us] [-w WIDTH] [file] ...
```

รวมบรรทัดหรือแบ่งบรรทัดแล้วแสดงผลให้แต่ละบรรทัดมีความยาวไม่เกินตัวอักษรที่กำหนด (75 ตัวอักษรโดยปริยาย).

`-u`	ลบช่องว่างที่เกินและไม่จำเป็น.
`-s`	แบ่งบรรทัดอย่างเดียว, ไม่รวมบรรทัด.
`-w` *WIDTH*	กำหนดความกว้าง (จำนวนตัวอักษร) ของบรรทัดที่ใช้แสดงผล.
file	ไฟล์ข้อมูล.

grep

```
grep [-irvln] keyword [filename] ...
```

สกัดบรรทัดที่มีคำที่ต้องการแสดงทางหน้าจอ. ชื่อคำสั่งมีที่มาจากบรรณาธิกรณ์แบบบรรทัด (line editor) ที่เรียกว่า ed. บรรณาธิกรณ์นี้เป็นบรรณาธิกรณ์รุ่นแรก ๆของยูนิกซ์และจะแก้ไขหรือเขียนข้อความได้เป็นบรรทัดต่อบรรทัด. แม้กระทั่งการแสดงผลก็สามารถแสดงได้เป็นบรรทัด ๆ, ไม่ใช่เป็นหน้า ๆเหมือนกับบรรณาธิกรณ์ที่ใช้กันทั่วไปในปัจจุบัน. บรรณาธิกรณ์ ed มีความสามารถที่จะหาคำในบรรทัดต่าง ๆได้โดยใช้ regular expression แล้วแสดงผลออกมา. คำสั่งที่สำหรับบรรณาธิกรณ์ ed ที่จะทำอย่างนี้มีรูปแบบเป็น `g/`*regular expression*`/p` ซึ่งเป็นที่มาของชื่อคำสั่ง grep.

- `-i` ไม่แยกแยะตัวอักษรตัวใหญ่ตัวเล็ก (insensitive case)
- `-n` ให้แสดงเลขบรรทัดต้นบรรทัดที่แสดง.
- `-v` แสดงบรรทัดที่ไม่มีคีย์เวิร์ด.
- `-l` แสดงชื่อไฟล์อย่างเดียว, ไม่แสดงบรรทัดที่มีคีย์เวิร์ดนั้น.
- `-r` หาคำที่อยู่ในไฟล์แบบ recursive ถ้าไฟล์ที่ระบุเป็นไดเรกทอรี.

gzip

```
gzip [-cdtv] [-] [file ...]
```

บีบอัดหรือขยายไฟล์.

`-c`	ส่งข้อมูลที่บีบอัดแล้วออกทาง stdout แทนที่จะบันทึกลงไฟล์.
`-d`	ขยายไฟล์. ถ้าตัวเลือกนี้ใช้ร่วมกับตัวเลือก `-c` ก็จะแสดงข้อมูลที่ขยายแล้วออกทาง stdout.
`-t`	ทดสอบไฟล์ที่บีบอัดแล้วว่าถูกต้องหรือไม่.
`-v`	ย่อมาจาก verbose. ใช้แสดงสถานะการทำงาน. โดยปรกติจะบอกสัดส่วนของข้อมูลที่บีบอัดแล้วเทียบกับข้อมูลเดิม.
`-`	ตัวเลือกสำหรับอ่านข้อมูลเข้าจาก stdin. ในกรณีนี้จะเขียนข้อมูลที่บีบอัดแล้วทาง stdout.
file	ไฟล์ที่ต้องการบีบอัด.

head

```
head [-c B] [-n N] [file]
```

แสดงบรรทัดตอนต้นของไฟล์.

`-c` *B*	แสดงข้อมูลส่วนต้นของไฟล์ *B* ไบต์.
`-n` *N*	ระบุจำนวนบรรทัด *N* ที่ต้องการดู.
file	ชื่อไฟล์ที่จะเป็นข้อมูลเข้า. ถ้าไม่ระบุจะเป็น stdin.

iconv

```
iconv [-c] [-f enc] [-t enc] [-o outfile] [filename]
iconv -l
```

แปลงการเข้ารหัสอักขระข้อมูล.

`-c`	เพิกเฉยข้อผิดพลาดของอักขระที่ไม่สามารถแปลงการเข้ารหัส.
`-f` *enc*	ระบุการเข้ารหัสอักขระของข้อมูลนำเข้า.
`-t` *enb*	ระบุการเข้ารหัสอักขระของข้อมูลออก.
`-o` *outfile*	ระบุชื่อไฟล์ผลลัพธ์. ถ้าไม่ใช้ตัวเลือกนี้จะแสดงผลลัพธ์ทาง stdout.
filename	ชื่อไฟล์ข้อมูลนำเข้า. ถ้าไม่ระบุชื่อไฟล์จะรับข้อมูลจาก stdin.
`-l`	แสดงรายชื่อการเข้ารหัสอักขระที่รองรับ.

install

```
install [-s] [-g group] [-m mode] [-o  owner] file path
```

ก็อปปี้ไฟล์หรือไดเรกทอรีและตั้งสิทธิ์การใช้ไฟล์.

`-s`	เอาส่วนที่เป็น symbol table ที่อยู่ในไฟล์ไบนารีออก. จะทำให้ไฟล์โปรแกรมไบนารีมีขนาดเล็กลง. `-s` ย่อยมาจากคำว่า strip.
`-g` *group*	กำหนดกลุ่มของไฟล์. *groups* จะเป็นชื่อกลุ่มหรือ gid ก็ได้.
`-m` *mode*	ตั้งสิทธิ์การใช้ไฟล์. *mode* จะเป็นแบบเลขฐานแปดหรือตัวอักษรก็ได้.
`-o` *owner*	กำหนดเจ้าของไฟล์. *owner* จะเป็นชื่อผู้ใช้หรือ uid ก็ได้.
file	ไฟล์ที่ต้องการก็อปปี้. ถ้ามีหลายไฟล์, อาร์กิวเมนต์ตัวสุดท้ายต้องเป็นชื่อไดเรกทอรี.
path	ชื่อไฟล์หลังการก็อปปี้หรือไดเรกทอรี.

join

```
join [-t C] file1 file2
```

ต่อคอลัมน์ของไฟล์สองไฟล์จากคอลัมน์ร่วมที่มีอยู่ในทั้งสองไฟล์. ถ้าไม่ระบุตัวเลือกพิเศษจะถือว่าคอลัมน์แรกของทั้งสองไฟล์เป็นคอลัมน์ร่วมกัน (มีค่าที่เหมือนกันเก็บอยู่). อ่าน `man join` สำหรับตัวเลือกโดยละเอียด.

`-t`	ใช้ตัวอักษร *C* เป็นตัวแบ่งคอลัมน์เวลาอ่านข้อมูลเข้าและเขียนข้อมูลออก. จะใช้ช่องว่างเป็นตัวแบ่งคอลัมน์ถ้าไม่กำหนดด้วยตัวเลือกนี้.
file1	ไฟล์ที่หนึ่ง.
file2	ไฟล์ที่สอง.

md5sum

```
md5sum [-c] [ref]
md5sum [file] ...
```

คำนวณและแสดงค่าผลรวม MD5 (128-bit).

`-c`	ตรวจสอบค่าผลรวม MD5 กับไฟล์รายการ *ref* ที่เตรียมไว้ .
file	ชื่อไฟล์ที่ต้องการตรวจสอบ.

nl

```
nl [-b style] [-n format] [-w width] [file] ...
```

แสดงบรรทัดของข้อมูล.

`-b`	แสดงเลขบรรทัดตามแบบ *style* ที่ระบุ.
แบบเลขบรรทัด	
`a`	แสดงเลขทุกบรรทัด.
`t`	แสดงเลขบรรทัดเฉพาะบรรทัดที่มีข้อมูล.
`n`	ไม่แสดงเลขบรรทัด.
`-n`	ใช้รูปแบบเลขบรรทัด *format* ที่ระบุ.
รูปแบบ	
`ln`	ชิดซ้าย.
`rn`	ชิดขวา.
`rz`	ชิดขวาและมีเลขศูนย์เติมให้มีจำนวนอักษรเท่ากัน.
`-w`	จำนวนหลักตัวเลข *width* ที่ใช้แสดงเลขบรรทัด.
file	ชื่อไฟล์ที่ต้องการแสดงหมายเลขบรรทัด. ถ้าไม่ระบุจะใช้ข้อมูลจาก stdin.

paste

```
paste [-d C] [-s] [file] ...
```

นำไฟล์เป็นบล็อกมาต่อกันตามแนวนอน. ใช้ Tab เป็นตัวเชื่อมส่วนที่ต่อ.

`-d` *C*	ใช้ตัวอักษร *C* เป็นตัวเชื่อมส่วนที่ต่อแทน Tab.
`-s`	ย่อมาจากคำว่า `--serial`. นำบรรทัดทุกบรรทัดในไฟล์แรกมาต่อกันในบรรทัดที่หนึ่ง, นำบรรทัดทุกบรรทัดในไฟล์ที่สองมาต่อกันในบรรทัดที่สอง, ...
file	ไฟล์ที่ต้องการนำมาต่อ. ถ้าชื่อไฟล์เป็น `-` จะหมายถึงข้อมูลจาก stdin.

sort

```
sort [-nr] [-o file] [input ...]
```

เรียงลำดับข้อมูลเป็นบรรทัด ๆ.

`-n`	จัดลำดับตามตัวเลข (number) ไม่ใช่ตัวอักษร.
`-r`	เรียงลำดับจากมากไปหาน้อย (reverse) ซึ่งการเรียงลำดับโดยปริยายจะเป็นน้อยไปหามาก.
`-o` *file*	บันทึกผลลัพธ์ (output) ลงไฟล์ *file*.
input ...	ชื่อไฟล์ที่เป็นข้อมูลเข้า. ถ้าไม่ระบุจะรับข้อมูลจาก stdin.

split

```
split [-a n] [-b bytes] [-l N] [-d] [--verbose] [file [prefix]]
```

แบ่งไฟล์เป็นไฟล์ย่อย ๆ.

`-a` *n*	จำนวนอักษร *n* ที่ใส่หลังชื่อไฟล์ย่อย. ถ้าไม่ระบุจะใช้ตัวอักษรสองตัวเป็น suffix หรือเลขสองหลัก (ถ้าใช้ตัวเลือก `-d`).
`-b` *bytes*	แบ่งไฟล์ให้มีขนาดจำนวนไบต์ *bytes* ที่กำหนด.
`-l` *N*	แบ่งไฟล์ให้แต่ละไฟล์มีจำนวนบรรทัด *N* ที่กำหนด.
`-d`	ใช้ชื่อไฟล์ย่อยที่ได้เป็นตัวเลขแทนตัวอักษร. ถ้าไม่ใช้ตัวเลือกนี้ไฟล์ย่อยที่ได้จะมีชื่อเป็น xaa, xab, ...
`--verbose`	แสดงสถานะการทำงาน.
file	ชื่อไฟล์ที่ต้องการแบ่ง. ถ้าไม่กำหนดจะรับข้อมูลจาก stdin.
prefix	ชื่อหน้าของไฟล์ย่อย. ถ้าไม่กำหนดจะใช้ x เป็นชื่อนำหน้าไฟล์.

tail

```
tail [-n N] [-f] [--follow=name|descriptor] [file] ...
```

แสดงข้อมูลส่วนท้ายของไฟล์.

`-n`	ระบุจำนวนบรรทัด *N* ที่ต้องการแสดง. ถ้าไม่ระบุจะแสดงข้อมูล 10 บรรทัดโดยปริยาย.
`-f`	รอแสดงผลต่อไปเรื่อย ๆถ้ามีการเขียนข้อมูลเพิ่มเข้ามาในไฟล์.
`--follow`	รอแสดงผลต่อไปเรื่อย ๆถ้ามีการเขียนข้อมูลเพิ่มเข้ามาในไฟล์. ให้ผลเหมือนตัวเลือก `-f` แต่สามารถระบุรายละเอียดได้ว่าถ้าไฟล์ที่ดูอยู่เปลี่ยนไปจะตามเปิดนั้นด้วยชื่อหรือ file descriptor. ตัวเลือก `-f` จะมีผลเหมือน `--follow=descriptor`.
file	ไฟล์ที่ต้องการดูข้อมูล.

tar

```
tar [cxtzjpvf] archive filename ..
```

สร้างไฟล์สำรองจากรายการไฟล์หรือไดเรกทอรี.

`-c`	สร้างไฟล์ archive.
`-x`	กระจายไฟล์ archive.
`-f` *`filename.tar`*	ระบุชื่อไฟล์ archive.
`-r`	เพิ่มไฟล์เข้าไปในไฟล์ archive ที่มีอยู่แล้ว.
`-t`	ทดสอบและแสดงรายการไฟล์ที่อยู่ในไฟล์ archive.
`-v`	แสดงสถานะการทำงาน.
`-z`	บีบอัดหรือคลายไฟล์ archive ด้วย `gzip`.
`-j`	บีบอัดหรือคลายไฟล์ archive ด้วย `bzip2`.
`-p`	รักษาสิทธิ์การใช้ไฟล์ที่ทำการ archive.
`-O`	กระจายไฟล์ที่อยู่ใน archive ออกทาง stdout.

tr

```
tr [-d] SET1 [SET2]
```

ชื่อของคำสั่งมาจากคำว่า translate or delete characters ใช้สำหรับเปลี่ยนตัวอักษรจาก stdin, หรือลบอักษรที่ไม่ต้องการ.

`-d`	ลบคำที่ระบุในอาร์กิวเมนต์ *SET1*.
SET1	อักษรที่เป็นเป้าหมายสำหรับเปลี่ยนหรือลบ.
SET2	อักษรที่ต้องการเปลี่ยนแทน.

unexpand

```
unexpand [-a] [--tabs=N] [file] ...
```

เปลี่ยน space ที่อยู่ต้นบรรทัดให้เป็น tab โดยถือว่า space แปดตัวคือ tab หนึ่งตัว.

`-a`	เปลี่ยน space ที่ปรากฏในไฟล์ทุกที่ให้เป็น tab. `-a` ย่อมาจาก `--all`.
`--tabs=`*N*	กำหนดให้ space จำนวน *N* ตัวแทน tab หนึ่งตัว.
file	ชื่อไฟล์.

uniq

```
uniq [-du] [filename]
```

ลบบรรทัดที่ซ้ำออกจากข้อมูลที่เรียงลำดับแล้ว. ชื่อคำสั่ง uniq มาจากคำว่า unique ซึ่งแปลว่าไม่ซ้ำ, ไม่เหมือน.

-d	แสดงบรรทัดที่ซ้ำเท่านั้น.
-u	แสดงบรรทัดที่ไม่ซ้ำเท่านั้น.
filename	ชื่อไฟล์ที่ต้องการประมวลผล. ถ้าไม่ระบุจะรับข้อมูลจาก stdin.

uudecode

```
uudecode [-o outfile] [file] ...
```

ถอดรหัสข้อมูลเท็กซ์ที่เข้ารหัสด้วยคำสั่ง uuencode.

-o *outfile*	ระบุชื่อไฟล์ผลลัพธ์ *outfile* บันทึกข้อมูลที่ถอดรหัส.
file	ชื่อไฟล์ข้อมูลที่เข้ารหัสด้วย uuencode. ถ้าไม่กำหนดชื่อไฟล์จะรับข้อมูลจาก stdin.

uuencode

```
uuencode [-m] [file] name
```

เข้ารหัสข้อมูลไบนารีให้เป็นข้อมูลเท็กซ์ ASCII.

-m	ใช้วิธีการเข้ารหัสแบบ base64. ถ้าไม่ระบุตัวเลือกนี้จะใช้การเข้ารหัสแบบ UU. การเข้ารหัสแบบ base64 จะเป็นที่นิยมกว่าและรองรับในภาษาคอมพิวเตอร์ต่าง ๆด้วยเช่น Perl, Ruby เป็นต้น.
file	ชื่อไฟล์ข้อมูลไบนารีที่ต้องการเข้ารหัส. ถ้าไม่กำหนดชื่อไฟล์จะรับข้อมูลจาก stdin.
name	ชื่อไฟล์สำหรับบันทึกในผลลัพธ์การเข้ารหัส. คำสั่ง uudecode จะสร้างไฟล์ชื่อที่ระบุ name แล้วบันทึกข้อมูลเข้าในไฟล์นี้.

wc

```
wc [-clw] [filename ...]
```

ชื่อคำสั่งมาจากคำว่า words count ใช้นับจำนวนบรรทัด, คำ, ตัวอักษรที่อยู่ในไฟล์. ถ้าไม่

ระบุชื่อไฟล์จะรับข้อมูลจาก stdin.

`-c`	นับเฉพาะจำนวนไบต์.
`-l`	นับเฉพาะจำนวนบรรทัด. บรรทัดในที่นี้หมายถึงจำนวนอักขระ newline.
`-w`	นับเฉพาะจำนวนคำ. คำในที่นี้คือคำที่แยกด้วยช่องไฟ.
`filename`	รับข้อมูลจากไฟล์. ถ้าไม่ระบุจะรับข้อมูลจาก stdin.

zip

```
zip [-j] [-P password] [zipfile] [filename ...]
```

บีบอัดไฟล์ลงในไฟล์ zip.

`-j`	เก็บเฉพาะชื่อไฟล์อย่างเดียวโดยไม่เก็บส่วนที่เป็นชื่อ path เช่นชื่อไดเรกทอรี.
`-P`	เข้ารหัสสำหรับไฟล์ที่ต้องการบีบอัด. เวลาจะขยายไฟล์ต้องใส่รหัสผ่าน *`password`* ที่ตั้งไว้.
`zipfile`	ชื่อไฟล์ zip ซึ่งจะเขียนส่วนขยายชื่อไฟล์ `.zip` หรือไม่ก็ได้.
`filename`	ชื่อไฟล์ที่ต้องการบีบอัด.

ข.2 จัดการระบบไฟล์

chgrp

```
chgrp [-R] group filename
```

แก้ไขเปลี่ยนกลุ่มของไฟล์.

`-R`	การแก้กลุ่มของไฟล์แบบ recursive. ใช้สำหรับแก้ไขสิทธิ์ทั้งไดเรกทอรีและไดเรกทอรีย่อย.
`group`	ชื่อกลุ่มที่ต้องการเปลี่ยน.
`filename`	ชื่อไฟล์ที่ต้องการแก้ไข.

chmod

```
chmod [-R] mode file
```

มาจากคำว่า change mode ใช้แก้ไขสิทธิ์การใช้ไฟล์.

`-R`	การแก้ไขสิทธิ์แบบ recursive. ใช้สำหรับแก้ไขสิทธิ์ทั้งไดเรกทอรีและไดเรกทอรีย่อย.
mode	รายละเอียดของสิทธิ์ที่ต้องการตั้ง.
file	ชื่อไฟล์หรือไดเรกทอรีที่ต้องการแก้ไขสิทธิ์.

cp

```
cp [-irpv] file path
```

ก็อปปี้ไฟล์หรือไดเรกทอรี.

`-i`	ถามย้ำก่อนก็อปปี้.
`-r`	ก็อปปี้ไฟล์ทั้งไดเรกทอรีและไฟล์ที่อยู่ใต้ไดเรกทอรีนั้น.
`-p`	สงวนสิทธิ์การใช้ไฟล์ที่ตั้งไว้ให้เหมือนเดิม.
`-v`	แสดงชื่อไฟล์ที่ก็อปปี้.
file	ไฟล์ที่ต้องการก็อปปี้. ถ้ามีหลายไฟล์, อาร์กิวเมนต์ตัวสุดท้ายต้องเป็นชื่อไดเรกทอรี.
path	ชื่อไฟล์หลังการก็อปปี้หรือไดเรกทอรี.

du

```
du [-skmh] [file ...]
```

ชื่อคำสั่งมาจากคำว่า disk usage. แสดงจำนวนพื้นที่ที่ถูกใช้งานในไดเรกทอรีที่ต้องการ.

`-s`	สรุปผลลัพธ์รวม. ไม่แสดงรายละเอียด.
`-k`	แสดงหน่วยของพื้นที่เป็นกิโบไบต์.
`-m`	แสดงหน่วยของพื้นที่เป็นเมกกาไบต์.
`-h`	แสดงหน่วยของพื้นที่ที่มนุษย์อ่านแล้วเข้าใจง่าย.
file ...	ชื่อไฟล์หรือไดเรกทอรีที่ต้องตรวจสอบ. ถ้าไม่ใส่จะแสดงการใช้พื้นที่ฮาร์ดดิสก์ที่ทำงานอยู่.
`-D`	ใช้ดูจำนวนใช้พื้นที่ในแต่ละไดเรกทอรี. ในลินุกซ์หน่วยที่แสดงโดยปริยายเป็นหน่วย Kilo-bytes (1024 ไบต์)

eject

```
eject [-v] [device]
```

ดีด CD หรือมีเดียอื่น ๆออกจากเครื่อง.

`-v`	แสดงรายละเอียดการทำงานทางหน้าจอ.
`device`	ชื่อดีไวซ์ที่ต้องการดีดออก. ถ้าไม่ระบุจะถือว่าเป็น CD.

find

```
find [path ...] [expression ...]
```

ค้นหาไฟล์ใต้ไดเรกทอรีที่ระบุ.

`path ...`	ไดเรกทอรีที่ต้องการหา.
`expression ...`	ข้อแม้ที่ต้องการหาหรือการกระทำเมื่อหาไฟล์เจอ.
	`-name` *`NAME`* ระบุชื่อไฟล์ที่ต้องการ.
	`-print` แสดงชื่อไฟล์ (relative path) ทางหน้าจอ.
	`-exec`

ln

```
ln [-s] filename link
```

สร้างลิงค์.

`-s`	ตัวเลือกสำหรับสร้างซอฟต์ลิงค์.
`filename`	ชื่อไฟล์ที่ต้องการสร้างลิงค์.
`link`	ชื่อไฟล์ใหม่ (ฮาร์ดลิงค์).

locate

```
locate [-i] search_string
locate -r regexpr
```

หาตำแหน่งของไฟล์ในระบบที่จากฐานข้อมูลที่เตรียมไว้.

`-i`	ไม่แยกแยะอักษรตัวเล็กหรืออักษรตัวใหญ่ (insensitive case).
`search_string`	ชื่อไฟล์, ไดเรกทอรี หรือส่วนของชื่อไฟล์ที่ต้องการหา.
`-r`	หาไฟล์โดยใช้ regular expression.
`regexpr`	ไฟล์หรือส่วนของชื่อไฟล์ที่ต้องการหาโดยเขียนในรูปแบบของ regular expression.

ls

```
ls [-AaFlti] [filename ...]
```

ชื่อคำสั่งมาจากคำว่า list directory contents, แสดงรายการไฟล์ในไดเรกทอรีที่ทำงานอยู่ในกรณีที่ไม่ชื่อไฟล์เป็นอาร์กิวเมนต์.

`-A`	แสดงรายการไฟล์เกือบทั้งหมดรวมถึงไฟล์ที่ขึ้นต้นด้วยเครื่องหมายจุดด้วยแต่ไม่แสดงไดเรกทอรี . และ ..
`-a`	แสดงรายการไฟล์ทั้งหมดรวมถึงไฟล์ที่ขึ้นต้นด้วยเครื่องหมายจุดด้วย.
`-F`	เพิ่มเครื่องหมายพิเศษหนังชื่อไฟล์เพื่อแยกแยะประเภทของไฟล์ให้ชัดเจน. เช่นเพิ่ม @ หลังชื่อไฟล์ที่เป็นลิงค์. เพิ่ม / หลังชื่อไฟล์ที่เป็นไดเรกทอรี. เพิ่ม | หลังชื่อไฟล์ที่เป็นไฟล์ไปป์. เพิ่ม = หลังชื่อไฟล์ที่เป็นไฟล์ socket. เพิ่ม * หลังไฟล์ที่กระทำการได้.
`-l`	แสดงรายละเอียดของไฟล์ต่าง ๆเช่น ขนาดไฟล์, ประเภทไฟล์ เป็นต้น.
`-t`	เรียงลำดับตามเวลา (timestamp) ของไฟล์.
`-i`	แสดง i-node ของไฟล์.
filename	ชื่อไฟล์หรือไดเรกทอรี.

mkdir

```
mkdir [-p] [file] ...
```

make directories, สร้างไดเรกทอรี.

-p	สร้างไดเรกทอรีแม่ด้วยโดยอัตโนมัติถ้าชื่อไดเรกทอรีที่ต้องการสร้างเป็นไดเรกทอรีย่อยซ้อนกัน.
file	ชื่อไดเรกทอรีที่ต้องการสร้าง.

mkfifo

```
mkfifo file ...
```

สร้างไฟล์ไปป์.

file	ชื่อไปป์ที่ต้องการสร้าง.

mknod

```
mknod name p
mknod name {b|c} major minor
```

สร้างไฟล์ fifo หรือสร้างไฟล์ดีไวส์.

p	สร้างไฟล์ fifo (named pipe).
b	สร้างไฟล์ดีไวส์แบบ block.
c	สร้างไฟล์ดีไวส์แบบ character.
major	major device number ของดีไวส์ที่ต้องการสร้าง.
minor	minor device number ของดีไวส์ที่ต้องการสร้าง.

mv

```
mv [-iv] old .. new
```

เปลี่ยนชื่อไฟล์หรือย้ายไฟล์.

-i	ถามย้ำก่อนกระทำ.
-v	แสดงชื่อไฟล์และที่ที่ย้ายไป.
old	ชื่อไฟล์ที่ต้องการเปลี่ยนชื่อหรือย้าย.
new	ชื่อไฟล์หรือไดเรกทอรีที่ต้องการเปลี่ยนชื่อหรือย้าย.

pwd

```
pwd
```

ชื่อคำสั่งมาจากคำว่า print working directory, แสดงชื่อไดเรกทอรีที่กำลังทำงานอยู่

rename

```
rename from to file ...
```

เปลี่ยนชื่อไฟล์หลาย ๆไฟล์พร้อม ๆกัน.

from	รูปแบบหรือส่วนของชื่อไฟล์ที่ต้องการเปลี่ยน.
to	รูปแบบหรือส่วนของชื่อไฟล์ที่หลังจากการเปลี่ยนชื่อ.
file	ชื่อไฟล์ที่ต้องการเปลี่ยนชื่อ.

rm

```
rm [-ivfr] file ...
```

ชื่อคำสั่งมาจากคำว่า remove file, ลบไฟล์หรือไดเรกทอรี.

`-i`	ถามย้ำก่อน (interactive) ที่จะลบไฟล์.
`-v`	แสดงชื่อไฟล์ที่ลบ (verbose).
`-f`	บังคับ (force) ให้ลบไฟล์.
`-r`	ลบไฟล์ทั้งหมดที่อยู่ในไดเรกทอรีและไดเรกทอรีนั้นด้วย. (recursive)
file	ชื่อไฟล์ที่ต้องการลบ.

rmdir

```
rmdir directory ...
```

ลบไดเรกทอรีที่ว่างเปล่า.

directory	ชื่อไดเรกทอรีที่ต้องการลบ.

shred

```
shred [-u] [-v] file ...
```

ทำลายข้อมูลที่อยู่ในไฟล์.

`-u`	ลบไฟล์หลังจากทำลายข้อมูล.
`-v`	แสดงรายละเอียดการทำงาน.
file	ชื่อไฟล์.

stat

```
stat [-f] filename ...
```

แสดงสถานะของไฟล์หรือระบบไฟล์.

`-f`	แสดงสถานะของระบบไฟล์.
filename	ชื่อไฟล์ที่ต้องการแสดงรายละเอียด.

touch

```
touch file ...
```

เปลี่ยน timestamps ของไฟล์.

file เปลี่ยน timestamps ของ *file* ที่ระบุ. ถ้าไฟล์ที่ระบุไม่มีตัวตน, จะเป็นการสร้างไฟล์ว่างเปล่าให้.

ข.3 เคอร์เนล

free

```
free [-b | -k | -m] [-t] [-s N]
```

แสดงการใช้งานของหน่วยความจำ.

- `-b` แสดงหน่วยเป็นไบต์.
- `-k` แสดงหน่วยเป็นกิโลไบต์.
- `-m` แสดงหน่วยเป็นเมกกาไบต์.
- `-t` แสดงผลรวม.
- `-s` แสดงผลของคำสั่งทุก ๆ *N* วินาที.

lsmod

```
lsmod
```

แสดงรายการโมดูลเคอร์เนลที่โหลดอยู่ในระบบ.

lspci

```
lspci [-v] [-vv]
```

ก็อปปี้ไฟล์หรือไดเรกทอรี.

- `-v` แสดงรายละเอียด.
- `-vv` แสดงรายละเอียดยิ่งขึ้น.

sync

```
sync
```

เขียนข้อมูลที่อยู่ใน buffer ลงในดิสก์.

vmstat

```
vmstat
```

ก็อปปี้ไฟล์หรือไดเรกทอรี.

`-i`	ถามย้ำก่อนก็อปปี้.
`-r`	ก็อปปี้ไฟล์ทั้งไดเรกทอรีและไฟล์ที่อยู่ใต้ไดเรกทอรีนั้น.
`-p`	สงวนสิทธิ์การใช้ไฟล์ที่ตั้งไว้ให้เหมือนเดิม.
`-v`	แสดงชื่อไฟล์ที่ก็อปปี้.
file	ไฟล์ที่ต้องการก็อปปี้. ถ้ามีหลายไฟล์, อาร์กิวเมนต์ตัวสุดท้ายต้องเป็นชื่อไดเรกทอรี.
path	ชื่อไฟล์หลังการก็อปปี้หรือไดเรกทอรี.

ข.4 ควบคุมโปรเซส

fuser

```
fuser [-m] file
```

ก็อปปี้ไฟล์หรือไดเรกทอรี.

`-i`	ถามย้ำก่อนก็อปปี้.
`-r`	ก็อปปี้ไฟล์ทั้งไดเรกทอรีและไฟล์ที่อยู่ใต้ไดเรกทอรีนั้น.
`-p`	สงวนสิทธิ์การใช้ไฟล์ที่ตั้งไว้ให้เหมือนเดิม.
`-v`	แสดงชื่อไฟล์ที่ก็อปปี้.
file	ไฟล์ที่ต้องการก็อปปี้. ถ้ามีหลายไฟล์, อาร์กิวเมนต์ตัวสุดท้ายต้องเป็นชื่อไดเรกทอรี.
path	ชื่อไฟล์หลังการก็อปปี้หรือไดเรกทอรี.

kill

```
kill -l
kill [-signal] pid ...
```

ทำให้โปรเซสสิ้นสุดการทำงาน, หรือส่งสัญญาณให้โปรเซส.

`-l`	แสดงรายการสัญญาณและหมายเลขสัญญาณที่สามารถใช้ได้.
`-signal`	ระบุสัญญาณ *`signal`* ที่ต้องการส่งด้วยหมายเลขหรือชื่อสัญญาณ.
`pid`	โปรเซส ID. หรือจ็อบ ID ก็ได้.

killall

```
killall -l
killall [-i] [-signal] name ...
```

ทำให้โปรเซสสิ้นสุดการทำงาน, หรือส่งสัญญาณให้โปรเซสโดยระบุชื่อโปรเซส.

`-i`	ตัวเลือกแบบย่อของ `--interactive` จะถามย้ำก่อนส่งสัญญาณ.
`-l`	แสดงรายการสัญญาณและหมายเลขสัญญาณที่สามารถใช้ได้.
`-signal`	ระบุสัญญาณ *`signal`* ที่ต้องการส่ngด้วยหมายเลขหรือชื่อสัญญาณ.
`name`	ชื่อโปรเซสโปรเซส.

pgrep

```
pgrep [-lx] RE
```

แสดงโปรเซส ID โดยจาก regular expression ที่ระบุ.

`-l`	ใช้แสดงชื่อโปรแกรมของโปรเซสประกอบกับ ID ด้วย.
`-x`	หาโปรเซส ID จากคำที่กำหนด. เช่นถ้าชื่อโปรเซสที่ต้องการหาคือ `emacs` ก็จะแสดงโปรเซส ID ของ `emacs` อย่างเดียวไม่แสดงโปรเซส ID ของ `xemacs` หรืออย่างอื่น.
`RE`	Regular expression สำหรับหาปรเซส ID เช่นชื่อโปรเซส.

pkill

```
pkill -signal
```

แสดงโปรเซส ID โดยจาก regular expression ที่ระบุ.

`-`*`signal`*	ระบุสัญญาณ *`signal`* ที่ต้องการส่งด้วยหมายเลขหรือชื่อสัญญาณ.
`-x`	หาโปรเซส ID จากคำที่กำหนด. เช่นถ้าชื่อโปรเซสที่ต้องการหาคือ `emacs` ก็จะแสดงโปรเซส ID ของ `emacs` อย่างเดียวไม่แสดงโปรเซส ID ของ `xemacs` หรืออย่างอื่น.
`RE`	Regular expression สำหรับหาปรเซส ID เช่นชื่อโปรเซส.

ps

```
ps [option ...]
```

รายงานสถานะของโปรเซสต่าง ๆ (process status).

`-e`	แสดงโปรเซสทุกตัวในระบบ.
`-f`	แสดงรายละเอียดของโปรเซส.
`-F`	แสดงรายละเอียดเกี่ยวกับโปรเซสคล้าย `-f` แต่จะให้ข้อมูลน้อยกว่า.

pstree

```
pstree [-cnpu] [pid | user]
```

แสดงความสัมพันธ์ระหว่างโปรเซสต่าง ๆด้วยแผนภาพต้นไม้.

`-c`	แสดงโปรเซสที่ซ้ำ. โดยปรกติถ้ามีโปรเซสซ้ำติดต่อกัน, คำสั่ง `pstree` จะแสดงจำนวนโปรเซสที่ซ้ำกันในวงเล็บเหลี่ยม.
`-n`	จัดลำดับตามเลขโปรเซส ID.
`-p`	แสดงโปรเซส ID.
`-u`	แสดงชื่อเจ้าของโปรเซส.
`pid`	โปรเซส ID ที่ต้องการแสดง.
`user`	ชื่อล็อกอินของผู้ใช้. ถ้าไม่มีการระบุ *`pid`* หรือ *`user`* จะแสดงโปรเซสทั้งหมดในระบบ.

strace

```
strace command [arg ...]
```

ดูซิสเต็มคอลล์และสัญญาณของโปรเซส.

`command`	คำสั่งที่ต้องการดูการทำงาน.

time

```
time command [args] ...
```

จับเวลาการทำงานของคำสั่งที่ระบุ.

command	คำสั่ง, โปรแกรม.
args	อาร์กิวเมนต์ของคำสั่ง *command*.

top

```
top -d s -n N -p pid[, pid ...]
```

แสดงรายละเอียดการใช้ทรัพยากรของโปรเซสต่าง ๆในขณะนั้น.

- `-d` ระบุจำนวนวินาที (*s*) สำหรับอัปเดทหน้าจอ.
- `-n` ระบุจำนวนครั้ง *N* ที่ต้องการอัปเดทในแต่ระรอบการแสดงผล.
- `-p` ระบุโปรเซส ID ของโปรเซสที่ต้องการ. ใช้เครื่องหมาย , คั่นถ้ามีต้องการดูโปรเซสหลายตัวพร้อม ๆกัน.

uname

```
uname [-a]
```

แสดงข้อมูลรายละเอียดเกี่ยวกับเคอร์เนลที่ใช้อยู่. ถ้าไม่มีอาร์กิวเมนต์จะแสดงแค่ชื่อเคอร์เนล.

- `-a` แสดงรายละเอียดทั้งหมด.
- `-n` แสดงชื่อเน็ตเวิร์ก (node name) ซึ่งได้แก่ชื่อโฮส.
- `-r` แสดง kernel release ซึ่งหมายถึงรุ่นของเคอร์เนล เช่น 2.6.5-gentoo-r1 เป็นต้น.
- `-v` แสดงรุ่นของเคอร์เนล. หมายถึงเวลาที่สร้างเคอร์เนล.
- `-m` แสดงสถาปัตยกรรมของเครื่อง.
- `-p` แสดงข้อมูลเบื้องต้นของหน่วยประมวลผล.

uptime

```
uptime
```

แสดงเวลาตั้งแต่เครื่องเริ่มทำงาน, จำนวนผู้ใช้ที่ใช้งาน, และ load ของเครื่อง.

ข.5 เน็ตเวิร์ก, สื่อสาร

hostname

```
hostname [-i] [-s] [-f] [-d] [-a]
```

แสดงหรือตั้งชื่อโฮสของระบบ. ไฟล์ที่เกี่ยวข้องกับคำสั่งนี้ได้แก่ไฟล์ /etc/hosts.

- `-i` แสดง IP address ของระบบ.
- `-s` แสดงชื่อสั้น.
- `-f` แสดงชื่อเต็มเช่นชื่อแบบ FQDN.
- `-d` แสดงชื่อโดเมน.
- `-a` แสดงชื่อเล่น (alias).

ข.6 ดูและระบบ

chown

```
chown [-R] loginname[:group] filename
```

เปลี่ยนเจ้าของไฟล์และกลุ่ม.

`-R`	เปลี่ยนเจ้าของไฟล์ไปทั้งไดเรกทอรี, ไดเรกทอรีย่อยและไฟล์ในนั้น (recursive).
loginname	ชื่อล็อกอินของเจ้าของไฟล์.
group	ชื่อกลุ่มที่ต้องการตั้งค่า.
filename	ชื่อไฟล์หรือไดเรกทอรี.

fdisk

```
fdisk device
```

จัดการตารางพาร์ทิชันของฮาร์ดดิสก์.

device ชื่อไฟล์ดีไวส์เช่น /dev/hda หมายถึงฮาร์ดดิสก์ตัวแรกแบบ (E)IDE.

init

```
init [0123456SsQqAaBbCcUu]
```

เปลี่ยนรันเลเวล. มีซอฟต์ลิงก์เป็น telinit.

Q หรือ q ให้ init อ่านไฟล์ตั้งค่าเริ่มต้น /etc/inittab อีกครั้ง.

localedef

```
localedef -f charmap -i locale locale_name
```

สร้างฐานข้อมูลโลแคล *locale_name* ในระบบ.

-f ระบุไฟล์การเข้ารหัสอักขระ *charmap*. ไฟล์นี้อยู่ในไดเรกทอรี /usr/share/i18n/charmaps.

-i ระบุไฟล์นิยามโลแคล locale. ไฟล์นี้จะอยู่ในไดเรกทอรี /usr/share/i18n/locales.

makewhatis

```
makewhatis [-v]
```

สร้างฐานข้อมูลสำหรับค้นหาคีย์เวิร์ดด้วยคำสั่ง man, apropos และ whatis.

-v แสดงสถานะการทำงานทางหน้าจอ.

mkfs

```
mkfs -t type device
```

สร้างระบบไฟล์.

type ชื่อระบบไฟล์ที่ต้องการสร้างเช่น ext2, xfs ฯลฯ.

device ชื่อไฟล์ดีไวส์ที่ต้องการสร้างระบบไฟล์.

mount

```
mount [-t fs] devicefile directory [-o options]
```

mount ระบบไฟล์หรือแสดงรายเอียดเกี่ยวกับระบบไฟล์ที่ mount อยู่ถ้าไม่มีตัวเลือก.

devicefile ชื่อดีไฟล์ไวส์ที่ต้องการ mount

directory ชื่อไดเรกทอรีที่จะเป็นตำแหน่ง mount. บางทีเรียกว่า mount point.

-t ระบุประเภทของระบบไฟล์ที่ต้องการ mount.

-o	ตัวเลือกต่าง ๆแล้วแต่ประเภทของระบบไฟล์.

newgrp

```
newgrp [-] [group]
```

เปลี่ยนกลุ่มปัจจุบันที่ตัวเองอยู่.

-	ใช้ล็อกอินเชลล์หลังจากที่เปลี่ยนเป็นกลุ่มที่ต้องการ.
group	ชื่อกลุ่มที่ต้องการเปลี่ยน. ถ้าไม่มีการระบุ, จะถือว่ากลุ่มที่ต้องการเปลี่ยนคือกลุ่มหลักที่กำหนดไว้ในไฟล์ /etc/passwd ของผู้ใช้นั้น ๆ.

passwd

```
passwd login
```

เปลี่ยนรหัสผ่านของผู้ใช้.

login	ชื่อล็อกอินของผู้ใช้ที่ต้องการเปลี่ยนรหัสผ่าน.

sg

```
sg [-] group [-c command]
```

เปลี่ยนกลุ่มปัจจุบันที่ตัวเองอยู่. คำสั่งนี้ต้องการอาร์กิวเมนต์ที่เป็นชื่อของกลุ่มในระบบเสมอ.

-	ใช้ล็อกอินเชลล์หลังจากที่เปลี่ยนเป็นกลุ่มที่ต้องการ.
group	ชื่อกลุ่มที่ต้องการเปลี่ยน.
-c *command*	สั่งคำสั่ง *command* เสมือนกับผู้ที่สั่งคำสั่งนั้นอยู่ในกลุ่ม *group*.

su

```
su [-] [username] [-c command]
```

ชื่อคำสั่งมาจากคำว่า substitue user ID. ใช้สำหรับเปลี่ยนผู้ใช้ที่ล็อกอินอยู่ให้เป็นผู้ใช้อื่นที่ต้องการ.

-	ใช้ล็อกอินเชลล์หลังจากที่เปลี่ยนเป็นผู้ใช้คนที่ต้องการ.
username	ชื่อผู้ใช้ที่ต้องการเปลี่ยน ID ไปเป็นคนนั้น. ถ้าไม่ระบุจะถือว่าเป็น root โดยปริยาย.

`-c` *command* สั่งคำสั่ง *command* เสมือนกับ *username* เป็นผู้สั่งคำสั่ง.

umount

```
umount directory
```

unmount ระบบไฟล์.

directory ชื่อไดเรกทอรีที่ต้องการ unmount.

update-rc.d

```
update-rc.d [-n] [-f] basename remove
update-rc.d [-n] name defaults [NN | NN-start NN-stop]
update-rc.d [-n] name start|stop NN runlevel runlevel  ...   .
             start|stop NN runlevel runlevel ...  . ...
```

คำสั่งเพิ่มหรือลบอินนิตสคริปต์แบบ System V ในรันเลเวลที่ต้องการ. คำสั่งนี้ใช้ในดิสโทรตระกูล Debian.

`-n` ไม่ทำจริง, แสดงให้ดูว่าจะทำอะไร.

`-f`

wall

```
wall [message]
```

ส่งข้อความไปสู่เทอร์มินอลทุกตัวในระบบถ้าอนุญาตให้ส่งข้อความ.

message ข้อความที่ต้องการส่ง. ถ้าไม่ระบุจะรับข้อมูลจาก stdin.

ข.7 พัฒนาโปรแกรม

gcc

```
gcc [file] ...
```

โปรแกรมสำหรับคอมไพล์รหัสต้นฉบับภาษา C หรือ C++.

file ไฟล์รหัสต้นฉบับ. ถ้าไม่มีจะรับข้อมูลเข้าจาก stdin.

ldd

```
ldd filename
```

ตรวจสอบดูว่าโปรแกรมที่ต้องใช้ขึ้นอยู่กับไลบรารี (shared library) อะไรบ้าง.

filename ชื่อไฟล์ (โปรแกรม) แบบ full path.

ข.8 อื่น ๆ

alias

```
alias [name=value] ...
```

ดู alias ที่ตั้งในเชลล์หรือใช้ตั้ง alias ของคำสั่ง.

name ชื่อ alias ที่ต้องการตั้ง. อาจจะเป็นชื่อโปรแกรมที่มีแล้วก็ได้.
value นิยามของ alias.

apropos

```
apropos [keyword] ...
```

ค้นหาคีย์เวิร์ดที่ปรากฏอยู่ในฐานข้อมูล whatis.

keyword คีย์เวิร์ดที่ต้องการค้นหา.

bc

```
bc [-l]
```

โปรแกรมภาษาเครื่องคิดเลขชั้นสูง.

`-l` คำนวณเป็นแบบละเอียดมีจุดทศนิยม.

cal

```
cal [[month] year]
```

displays a calendar, แสดงปฏิทินของเดือนปัจจุบันถ้าไม่มีอาร์กิวเมนต์.

month เดือนเป็นตัวเลขตั้งแต่ 1 ถึง 12.
year ปีเช่น 1998.

dircolors

```
dircolors [file]
```

ติดตั้งสีที่ใช้กับคำสั่ง ls.

file ไฟล์ที่ระบุว่าเนื้อหาของสีที่จะใช้กับไฟล์ประเภทต่าง ๆ.

env

```
env [NAME=ENV] ... [command arg ...]
```

แสดงตัวแปรสภาพแวดล้อมและค่าของตัวแปรถ้าไม่มีอาร์กิวเมนต์. ถ้ามีชื่อคำสั่ง *command* เป็นอาร์กิวเมนต์ก็จะรันคำสั่งในสภาพแวดล้อมที่กำหนด.

NAME ชื่อตัวแปรสภาพแวดล้อม.
ENV ค่าตัวแปรสภาพแวดล้อม.
command คำสั่งที่ต้องการกระทำ.
arg อาร์กิวเมนต์ของคำสั่ง.

info

```
info [manual]
```

อ่านคู่มือการใช้งานทางเทอร์มินอลที่บันทึกเป็นฟอร์แมต info.

manual ชื่อคู่มือที่ต้องการอ่าน. ถ้าไม่ระบุจะแสดงหน้าหลักของ info ซึ่งเป็นสารบัญของเอกสารทั้งหมด.

less

```
less [-NL] filename
```

โปรแกรมเพจเจอร์ (pager) สำหรับดูข้อมูลในไฟล์เป็นหน้า ๆในเทอร์มินอล.

-N แสดงหมายเลขบรรทัดที่ต้นบรรทัด.

-L ไม่ใช้โปรแกรมช่วยที่ระบุไว้ในตัวแปรสภาพแวดล้อม LESSOPEN. โปรแกรมช่วยนี้จะพยายามใช้โปรแกรมตัวอื่นช่วยประมวลผลก่อนที่จะส่งข้อมูลให้ less. ตัวอย่างเช่นถ้าไฟล์ที่ต้องการเปิดเป็นไฟล์ที่บีบอัดมา (.gz) ก็จะใช้โปรแกรม gzip ขยายไฟล์นั้นแล้วส่งผลลัพธ์ให้ less.

logout

```
logout
```

ล็อกเอาท์ออกจากล็อกอินเชลล์. ถ้าไม่ใช่ล็อกอินเชลล์จะไม่สามารถใช้คำสั่งนี้ได้.

date

```
date [MMDDhhmm[[CC]YY][.ss]]
date +FORMAT
```

ตั้งค่าหรือแสดงวันเวลาของระบบ.

MMDDhhmm เวลาที่ต้องการตั้งค่า. ตัวอย่างเช่น 08112157 หมายถึงวันที่ 11 เดือนสิงหาคม, เวลา 21 นาฬิกา 57 นาที. *CCYY* คือปีเช่น 2004. และ *ss* คือวินาที.

FORMAT แสดงวันเวลาตามแบบที่ระบุด้วย FORMAT ที่พิมพ์ไป.
%F ปี-เดือน-วัน
%D วัน/เดือน/ปี
%a ชื่อย่อของวันในสัปดาห์ตาม locale ที่กำหนด.
%A ชื่อเต็มของวันในสัปดาห์ตาม locale ที่กำหนด.
%b ชื่อย่อของเดือนตาม locale ที่กำหนด.
%B ชื่อเต็มของเดือนตาม locale ที่กำหนด.
%c วันและเวลาตาม locale ที่กำหนด.
%C หน่วยร้อยปีที่ตัดส่วนร้อยออก. รูปแบบนี้ไม่เกี่ยวกับ locale เช่น 2004 จะแสดงเป็น 20.
%d วันที่ในเดือน (01-31).
%D วันที่ (mm/dd/yy).
%e วันที่ในเดือนโดยไม่เติมศูนย์นำหน้า (1-31).
%F ให้ผลเหมือนกับ %Y-%m-%d.
%g เลขท้ายสองหลักของปี.
%G เลขสี่หลักแสดงปี.
%h ให้ผลเหมือนกับ %b.
%H ชั่วโมง (00-23).
%I ชั่วโมง (01-12).

%j วันในหนึ่งปี (001-366).
%k ชั่วโมงไม่มีศูนย์นำหน้า (0-23).
%l ชั่วโมงไม่มีศูนย์นำหน้า (1-12).
%m เดือนแสดงเป็นตัวเลข (01-12).
%M นาที (00-59).
%N nanoseconds (000000000-999999999).
%p แสดง AM หรือ PM เป็นตัวอักษรตัวใหญ่.
%P แสดง am หรือ pm เป็นอักษรตัวเล็ก.
%r เวลาแบบ 12 ชั่วโมง (hh:mm:ss [AP]M).
%R เวลาแบบ 24 ชั่วโมง (hh:mm).
%s จำนวนวินาทีตั้งแต่ 00:00:00 1970-01-01 UTC.
%S วินาที (00-60).
%T เวลาแบบ 24 ชั่วโมง (hh:mm:ss).
%u ตัวเลขแสดงวันในสัปดาห์ (1-7). เลข 1 หมายถึงวันจันทร์.
%U จำนวนสัปดาห์ในหนึ่งปีโดยที่วันอาทิตย์เป็นวันแรกของสัปดาห์ (00-53).
%V จำนวนสัปดาห์ในหนึ่งปีโดยที่วันจันทร์เป็นวันแรกของสัปดาห์ (01-53).
%w ตัวเลขแสดงวันในสัปดาห์ (0-6). เลข 0 หมายถึงวันอาทิตย์.
%W จำนวนสัปดาห์ในหนึ่งปีโดยที่วันจันทร์เป็นวันแรกของสัปดาห์ (00-53).
%x แสดงวันตาม locale (mm/dd/yy).
%X แสดงเวลาตาม locale (%H:%M:%S).
%y เลขท้ายสองหลักของปีคริสตศักราช.
%Y ปีคริสตศักราช.
%z แสดงเขตเวลา (timezone) เป็นตัวเลข. เช่นประเทศไทยจะเป็น +0700.
%Z แสดงเขตเวลาเป็นตัวอักษร.

file

```
file filename ...
```

ตรวจสอบและรายงานประเภทของไฟล์.

filename ไฟล์ที่ต้องการตรวจสอบ.

groups

```
users [user]
```

แสดงกลุ่มที่ *user* นั้นอยู่. ถ้าไม่มีอาร์กิวเมนต์จะแสดงกลุ่มของผู้ใช้ที่สั่งคำสั่ง.

id

```
id
```

สำรวจ UID และ GID ของผู้ใช้.

last

```
last [-n N] [-N] [-f file] [-x] tty ...
```

แสดงชื่อและข้อมูลการล็อกอินล็อกเอาท์.

- `-n` แสดงจำนวนบรรทัดตามที่ต้องการ. ถ้าไม่ระบุจะแสดงข้อมูลทั้งหมดที่อยู่ในไฟล์ `/var/log/wtmp` โดยปริยาย.
- *N* จำนวนบรรทัด.
- `-f` ไฟล์ที่ดึงข้อมูลมา. จะใช้ไฟล์ `/var/log/wtmp` โดยปริยายถ้าไม่ระบุเป็นไฟล์อื่น.
- `-x` แสดงข้อมูลเกี่ยวกับการรีบูต, เปิดเครื่อง, ปิดเครื่องด้วย.
- *tty* ชื่อเทอร์มินอล. จะแสดงข้อมูลเกี่ยวกับเทอร์มินอลที่ระบุเท่านั้น.

locale

```
locale [-a] [-m]
```

แสดงข้อมูลเกี่ยวกับโลเคลในระบบ. ถ้าสั่งคำสั่งโดยไม่มีอาร์กิวเมนต์จะแสดงตัวแปรสภาพแวดล้อมและค่าที่เกี่ยวกับโลแคลของผู้ใช้.

- `-a` แสดงชื่อโลแคลทั้งหมดในระบบ.
- `-m` แสดงชื่อการเข้ารหัสทั้งหมดที่เกี่ยวกับโลแคลในระบบ.

look

```
look [-a] string
```

แสดงคำภาษาอังกฤษที่ขึ้นต้นด้วย *string*.

-a	ใช้ ไฟล์ /usr/share/dict/web2 แทน ไฟล์ /usr/share/dict/words.
string	ส่วนของคำที่ต้องการหา.

man

```
man [section] manual
```

on-line manual page, คำสั่งสำหรับดูคู่มือการใช้งานต่าง ๆทางเทอร์มินอล.

section	ตัวเลขหรืออักษรเพื่อระบุหมวดหมู่.
manual	ชื่อคู่มือ, ชื่อคำสั่ง หรือโปรแกรมที่ต้องการดูคู่มือ.

od

```
od [-s] [-x] [-c] [file]
```

ดัมป์ (dump) เนื้อหาของไฟล์เป็นค่าเลขฐานแปดหรือฟอร์แมตอื่น ๆ.

-s	แสดงส่วนที่เป็นคำที่อ่านได้
-x	แสดงค่าเป็นเลขฐาน 16
-c	แสดงค่าที่แสดงได้เป็นอักขระ ASCII
file	ไฟล์ไบนารีที่ต้องการสำรวจ, ถ้าไม่มีข้อมูลจะมาจาก stdin.

printenv

```
printenv [VAR]
```

แสดงตัวแปรสภาพแวดล้อมและของตัวแปรทั้งหมดหรือที่ระบุ.

VAR	ชื่อตัวแปรสภาพแวดล้อมที่ต้องการแสดงค่า.

reset

```
reset
```

เริ่มต้นเทอร์มินอลใหม่.

script

```
script [-a] [file]
```

บันทึกสิ่งที่แสดงบนเทอร์มินอลลงในไฟล์.

- `-a` บันทึกข้อมูลต่อลงในไฟล์ที่มีอยู่แล้ว.
- *file* ไฟล์ผลลัพธ์ที่บันทึกข้อมูลบนเทอร์มินอล. ถ้าไม่ระบุชื่อไฟล์จะสร้างไฟล์ชื่อ typescript ให้ในไดเรกทอรีที่ทำงานอยู่โดยอัตโนมัติ.

set

```
set [-o option]
```

เป็นคำสั่งภายในเชลล์ใช้ตั้งค่าปัจจัยต่าง ๆของเชลล์.

- `-o noclobber` ปรับพฤติกรรมเชลล์ไม่ให้รีไดเรก. ยกเลิกการการตั้งค่า noclobber ได้ด้วยตัวเลือก +o noclobber.

strings

```
strings [-a] [-N] file ...
```

สกัดส่วนที่เป็นข้อมูลเท็กซ์จากไฟล์ไบนารีแล้วแสดงทางหน้าจอ.

- `-a` ตรวจสอบทั้งไฟล์. ถ้าไม่ระบุตัวเลือกนี้จะเป็นการสกัดข้อมูลเท็กซ์จากไฟล์ช่วงที่เป็นข้อมูล (data section) เท่านั้น.
- *-N* *N* คือจำนวนอักขระอย่างน้อยที่เรียงติดกันและถือว่าเป็นข้อมูลเท็กซ์. ถ้าไม่ระบุจะถือว่าข้อมูลเท็กซ์คืออักขระ ASCII ที่เรียงติดต่อกันอย่างนั้น 4 ตัว.

stty

```
stty [-a] [-]setting
```

แสดงหรือกำหนดพฤติกรรมต่างของเทอร์มินอล. ให้อ่าน stty(1) เพิ่มเติมสำหรับรายละเอียดสิ่งที่กำหนดได้ทั้งหมด.

- `-a` แสดงค่าต่าง ๆที่ใช้ในเทอร์มินอล.

setting ค่าที่ต้องการกำหนด. ตัวอย่างเช่น ถ้าต้องการไม่ให้ echo สิ่งที่พิมพ์จะใช้เครื่องหมาย - นำหน้าชื่อที่ต้องการตั้ง, `stty -echo`. ถ้าต้องการกำหนดเทอร์มินอลให้ echo อีกครั้งก็ใช้เอาเครื่องหมาย - ออก.

tee

```
tee [-a] [filename]
```

อ่านข้อมูลจาก stdin แล้วส่งต่อให้ sdtout และบันทึกลงไฟล์ถ้ามีการระบุชื่อไฟล์เป็นอาร์กิวเมนต์.

`-a`	บันทึกข้อมูลต่อเพิ่มจากไฟล์ที่มีอยู่.
filename	ชื่อไฟล์ที่ต้องการบันทึกข้อมูล.

test

```
test [EXPRESSION]
```

ทดสอบไฟล์และเปรียบเทียบค่าต่าง ๆ. คำสั่งนี้มีผลเหมือนคำสั่ง `[` แต่คำสั่ง `[` ต้องการเครื่องหมาย `]` ลงท้ายตอนจบคำสั่งเสมอ.

EXPRESSION	นิพจน์ที่ต้องการทดสอบ.
`!` *EXPRESSION*	กลับค่าจากจริงให้เป็นเท็จหรือจากเท็จให้เป็นจริง.
EXPRESSION1 `-a` *EXPRESSION2*	เป็นจริงถ้านิพจน์ทั้งสองเป็นจริง (AND).
EXPRESSION1 `-o` *EXPRESSION2*	เป็นจริงถ้านิพจนใดนิพจน์หนึ่งเป็นจริง (OR).
STRING1 `=` *STRING2*	เป็นจริงถ้าสายอักขระแรกเหมือนกับสายอักขระที่สอง.
STRING1 `!=` *STRING2*	เป็นจริงถ้าสายอักขระแรกแตกต่างจากสายอักขระที่สอง.
`-z` *STRING*	เป็นจริงถ้าขนาดของสายอักขระเป็นศูนย์.
`[-n]` *STRING*	เป็นจริงถ้าขนาดของสายอักขระไม่เป็นศูนย์. จะเขียน `-n` ห
INTEGER1 `-eq` *INTEGER2*	เป็นจริงถ้าจำนวนเต็มทั้งสองมีค่าเท่ากัน.
INTEGER1 `-ne` *INTEGER2*	เป็นจริงถ้าจำนวนเต็มทั้งสองมีค่าไม่เท่ากัน.
INTEGER1 `-gt` *INTEGER2*	เป็นจริงถ้าจำนวนเต็มแรกมีค่ามากกว่าจำนวนเต็มจริงตัวที่ส
INTEGER1 `-lt` *INTEGER2*	เป็นจริงถ้าจำนวนเต็มแรกมีค่าน้อยกว่าจำนวนเต็มจริงตัวที่ส
INTEGER1 `-ge` *INTEGER2*	เป็นจริงถ้าจำนวนเต็มแรกมีค่ามากกว่าหรือเท่ากับจำนวนเต็ สอง.
INTEGER1 `-le` *INTEGER2*	เป็นจริงถ้าจำนวนเต็มแรกมีค่าน้อยกว่าหรือเท่ากับจำนวนเต็ สอง.
`-f` *FILE*	ตรวจสอบว่ามีไฟล์ *FILE* จริงและเป็นไฟล์ธรรมดา (เช่นไ ไวส์).
FILE1 `-nt` *FILE2*	ตรวจสอบว่าไฟล์ *FILE1* ใหม่กว่า *FILE2* หรือไม่.

FILE1 -ot *FILE2*	ตรวจสอบว่าไฟล์ *FILE1* เก่ากว่า *FILE2* หรือไม่.
-d *FILE*	ตรวจสอบว่า *FILE* คือไดเรกทอรีหรือไม่.
-e *FILE*	ตรวจสอบว่า *FILE* มีจริงหรือไม่.
-h *FILE*	ตรวจสอบว่า *FILE* คือซอฟต์ลิงค์หรือไม่.
-r *FILE*	ตรวจสอบว่าไฟล์มีจริงและสามารถอ่านได้หรือไม่.
-s *FILE*	ตรวจสอบว่าไฟล์มีจริงและขนาดมากกว่าศูนย์.
-w *FILE*	ตรวจสอบว่าไฟล์มีจริงและสามารถแก้ไขได้.
-x *FILE*	ตรวจสอบว่าไฟล์มีจริงและสามารถกระทำการได้.

tty

```
tty
```

แสดงชื่อไฟล์ดีไวซ์ของเทอร์มินอลที่ใช้.

type

```
type name
```

ตรวจสอบคำสั่งว่าเป็นคำสั่งประเภทใด (คำสั่งประกอบภายใน, โปรแกรมคำสั่ง, ฟังก์ชัน ฯลฯ). คำสั่งนี้เป็นคำสั่งประกอบภายในเชลล์ bash.

name ชื่อคำสั่งที่ต้องการตรวจสอบ.

users

```
users [file]
```

แสดงชื่อล็อกอินของผู้ที่ล็อกอินอยู่ในระบบ. ข้อมูลของผู้ที่ล็อกอินมาจากไฟล์ /var/run/utmp โดยปริยาย.

file ไฟล์ /var/run/utmp หรือ /var/log/wtmp.

w

```
who [-]
```

แสดงผู้ที่ล็อกอินอยู่และรายงานว่ากำลังทำอะไร. ในบรรทัดแรกเป็นหัวเรื่องแสดงเวลา

ปัจจุบัน, ระยะเวลาที่เครื่องปฏิบัติการ, จำนวนผู้ใช้ที่ล็อกอิน, โหลดโดยเฉลี่ยนของเครื่องในระยะเวลา 1, 5 และ 15 นาทีที่ผ่านมา.

- `-s` แสดงข้อมูลแบบสั้น. ไม่แสดงเวลาล็อกอิน, เวลา JCPU และเวลา PCPU. JCPU คือเวลาที่หน่วยประมวลใช้ไปกับโปรเซสที่รันอยู่ในเทอร์มินอลนั้นไม่ว่าจะเป็นโปรเซส foreground หรือ background. PCPU คือเวลาที่หน่วยประมวลผลใช้ไปกับโปรแกรมที่รันอยู่ในเทอร์มินอล (ชื่อโปรแกรมที่อยู่ในคอลัมน์ what).
- `-f` แสดงชื่อโฮสหรือ IP address ของผู้ที่ล็อกอิน.

whatis

```
whatis [keyword] ...
```

ค้นหาชื่อคำสั่งจากฐานข้อมูล whatis.

- *keyword* คีย์เวิร์ดที่ต้องการค้นหา.

whereis

```
whereis [-bms] command ...
```

แสดงไฟล์ไบนารี, รหัสต้นฉบับ, และคู่มือใช้งานของคำสั่ง.

- `-b` แสดงไฟล์ไบนารี.
- `-m` แสดงไฟล์คู่มือใช้งาน.
- `-s` แสดงรหัสต้นฉบับ.
- *command* ชื่อคำสั่ง.

which

```
which [-a] command
```

แสดง full path ของคำสั่ง.

- `-a` แสดง full path ของคำสั่งทุกตัวที่ตรวจสอบได้จากตัวแปรสภาพแวดล้อม PATH.
- *command* ชื่อคำสั่ง.

who

```
who [-Hu]
```

แสดงผู้ที่ล็อกอินอยู่และรายละเอียดอื่น ๆ.

- `-H` แสดงหัวข้อ (header) ในแต่ละคอลัมน์.
- `u` แสดงรายชื่อผู้ใช้และข้อมูลอื่น ๆ.

whoami

```
whoamai
```

แสดง ID ของผู้ใช้. เช่นถ้าผู้ใช้เป็น root ก็จะให้ผลเป็น root ทางหน้าจอ.

xargs

```
xargs [command [arguments]]
```

รันคำสั่งโดยใช้ข้อมูลจาก stdin เป็นอาร์กิวเมนต์.

command	คำสั่งที่ต้องการสั่ง. ถ้าไม่ระบุจะรันคำสั่ง `echo` โดยปริยาย.
arguments	อาร์กิวเมนต์ของคำสั่งถ้าต้องการระบุ.
`-p`	สงวนสิทธิ์การใช้ไฟล์ที่ตั้งไว้ให้เหมือนเดิม.

ข.9 เชลล์

basename

```
basename filename [suffix]
```

สกัดส่วนที่เป็นชื่อไฟล์จริง ๆจาก *filename*. ถ้าระบุ *suffix* ด้วยจะตัดส่วนที่เป็นส่วนขยายชื่อไฟล์ออกจากชื่อด้วย.

filename	ชื่อไฟล์หรือไดเรกทอรี. จะเป็น full path หรือไม่ก็ได้. ถ้าเป็นไดเรกทอรี, จะแสดงชื่อไดเรกทอรี.
suffix	ส่วนขยายชื่อไฟล์ที่ต้องการตัดออก.

bg

```
bg [job]
```

เปลี่ยนสภาพจ็อบที่หยุดชั่วคราวให้กระทำการแบบ background.

job หมายเลขจ็อบเช่น %4 หมายถึงจ็อบที่ 4. ถ้าไม่ระบุจะเชลล์จะทำให้จ็อบตัวที่ถูกหยุดตัวสุดท้ายเป็นจ็อบแบบ background.

dirname

```
basename filename
```

สกัดส่วนที่เป็นไดเรกทอรีจาก *filename*. ถ้า *filename* เป็นชื่อโดดๆ, ผลลัพธ์คือ . (ไดเรกทอรีปัจจุบัน).

filename ชื่อไฟล์หรือไดเรกทอรี.

clear

```
clear
```

เคลียร์หน้าจอเทอร์มินอล. ถ้าเป็นเชลล์ bash จะใช้คีย์ C-l ก็ได้.

echo

```
echo [-en] [string ...]
```

แสดงข้อมูลเท็กซ์, คำทางหน้าจอที่รับมาจากอาร์กิวเมนต์.

-e แปลคำที่ขึ้นต้นด้วยอักษร \ ให้มีความหมายพิเศษ. เช่น \t คือ Tab.
-n ไม่ขึ้นบรรทัดใหม่หลังจากจบการทำงาน (หลังจากแสดงคำ).
string คำมากกว่าหนึ่งคำหรือไม่มีก็ได้.

eval

```
eval EXPR
```

ตีความและกระทำการอาร์กิวเมนต์ที่ได้รับ.

EXPR วลีที่เชลล์สามารถตีความได้.

exit

```
exit
```

จบการทำงานของเชลล์.

export

```
export [VAR ...] [VAR=VAL ...]
```

คำสั่งภายในเชลล์ใช้สำหรับกำหนดตัวแปรให้เป็นตัวแปรสภาพแวดล้อม. ถ้าสั่งคำสั่งโดยไม่มีอาร์กิวเมนต์จะแสดงตัวแปรสภาพแวดล้อมและค่าที่กำหนดไว้ในเชลล์นั้น.

VAR ...	ชื่อตัวแปรที่ต้องการ export. ถ้าไม่ระบุจะเป็นการดูตัวแปรสภาพแวดล้อมทั้งหมด.
VAR=VAL ...	กำหนดตัวแปรสภาพแวดล้อมและค่าในทีเดียว.

expr

```
expr expr
```

ตีความนิพจน์ ***expr***

arg1 + arg2	แสดงผลบวก.
arg1 - arg2	แสดงผลลบ.
*arg1 * arg2*	แสดงผลคูณ. ให้ระวังเครื่องหมาย * เวลาเขียนในเชลล์พรอมต์.
arg1 / arg2	แสดงผลหาร.
arg1 % arg2	แสดงเศษของการหาร.

factor

```
factor number ...
```

หาตัวประกอบของจำนวนเต็ม ***number***.

false

```
false
```

ส่งค่าจบการทำงานเป็นเท็จซึ่งหมายถึงจำนวนเต็ม 1. ไม่แสดงผลใด ๆบนหน้าจอ.

fg

```
fg [job]
```

เปลี่ยนสภาพของจ็อบให้เป็นจ็อบแบบ foreground และทำให้เป็นจ็อบที่ดำเนินงานในปัจจุบัน.

job หมายเลขจ็อบที่ต้องการทำให้จ็อบ foreground. ถ้าไม่ระบุจะหมายถึงจ็อบตัวล่าสุดที่เป็น background.

help

```
help [-s] [pattern] ...
```

ใช้แสดงการใช้คำสั่งภายในของเชลล์.

-s แสดงค่าวิธีใช้อย่างง่ายโดยไม่มีคำอธิบาย.

pattern รูปแบบที่ต้องการค้นหา. ไม่จำเป็นต้นเป็นชื่อคำสั่งก็ได้. ถ้าไม่ระบุอะไรจะแสดงรายการความช่วยเหลือที่มี.

jobs

```
jobs
```

แสดงรายการจ็อบที่กระทำการอยู่ในเชลล์นั้น.

mesg

```
mesg [y|n]
```

แสดงสถานะของเทอร์มินอลว่าสามารถรับข้อมูลจากผู้ใช้อื่นได้หรือไม่. ถ้ามีอาร์กิวเมนต์จะเป็นการตั้งค่าอนุญาติ (y) หรือไม่อนุญาต (n) การส่งข้อความเข้ามาในเทอร์มินอลที่ใช้อยู่.

y อนุญาตให้รับข้อความเข้ามาในเทอร์มินอลที่ใช้.

n ไม่อนุญาตให้รับข้อความเข้ามาในเทอร์มินอลที่ใช้.

nice

```
nice [-n pri] command arg ...
```

จัดความสำคัญของคำสั่งที่จะใช้.

-n *pri*	ตั้งค่าความสำคัญของโปรเซส. *pri* คือตัวเลขตั้งแต่ -20 (สำคัญสูงสุด) จนถึง 19 (สำคัญต่ำสุด). ถ้าไม่ระบุจะตั้งความสำคัญเป็น 10 โดยปริยาย.
command	คำสั่งที่ต้องการกระทำ.
arg	อาร์กิวเมนต์ของคำสั่ง.

nohup

```
nohup command arg ...
```

รันคำสั่งโดยที่เพิกเฉยกับสัญญาณ SIGHUP ทำให้คำสั่งทำงานได้ต่อไปหลังจากล็อกเอาท์.

command	คำสั่งที่ต้องการกระทำ.
arg	อาร์กิวเมนต์ของคำสั่ง *command*.

renice

```
renice pri [-p pid] [-g pgrp] [-u user]
```

เปลี่ยนความสำคัญของโปรเซส. root เท่านั้นที่สามารถสั่งคำสั่งนี้ได้.

pri	ตั้งค่าความสำคัญของโปรเซส. *pri* คือตัวเลขตั้งแต่ -20 (สำคัญสูงสุด) จนถึง 19 (สำคัญต่ำสุด).
-p *pri*	ระบุหมายเลขโปรเซส *pid* ที่ต้องการเปลี่ยนความสำคัญ.
-g *pgrp*	เปลี่ยนความสำคัญของโปรเซสที่รันโดยกลุ่ม *pgrp*.
-u *user*	เปลี่ยนความสำคัญของโปรเซสที่รันโดยผู้ใช้ *user*.

select

```
select NAME [in WORDS ... ;] do COMMANDS; done
```

แสดงตัวเลือกเป็นข้อ ๆให้เลือกและเก็บค่าคำตอบลงในตัวแปร *NAME* ที่กำหนดไว้.

seq

```
seq [-s string] [-f format] [-w] num
seq [OPTION] start stop
seq [OPTION] start incr stop
```

แสดงลำดับของจำนวนเต็มตามที่กำหนด. ถ้ามีอาร์กิวเมนต์เพียงตัวเดียวจะแสดงลำดับเริ่มตั้งแต่ 1 ถึงจำนวน *num* ที่กำหนด.

`-s` *string*	ใช้สายอักขระ *sting* เป็นตัวแบ่ง. ถ้าไม่ใช้ตัวเลือกนี้จะใช้ `\n` เป็นตัวแบ่งระหว่างจำนวน (แสดงบรรทัดต่อบรรทัด).
`-f` *format*	แสดงจำนวนตามแบบคำสั่ง `printf`. เช่น `-f 'a%03g'` จะให้ผลเป็น a001, a002 เป็นต้น.
`-w`	เติมเลข 0 เพื่อให้จำนวนที่แสดงมีความกว้างเท่ากัน. ตัวอย่างเช่น 09, 10.
start	จำนวนเต็มเริ่มต้น.
stop	จำนวนเต็มตัวสุดท้าย.
incr	จำนวนที่ใช้เพิ่มลำดับ.

source

```
source filename
```

คำสั่งภายในของเชลล์. อ่านและกระทำคำสั่งที่อยู่ในไฟล์ที่เป็นอาร์กิวเมนต์ในเชลล์ที่ทำงานอยู่. คำสั่งที่เหมือนกับ `source` คือ `.` (จุด).

filename	ชื่อไฟล์ที่มีคำสั่งที่ต้องการกระทำการในเชลล์ที่ทำงานอยู่.

true

```
true
```

ส่งค่าจบการทำงานเป็นจริงซึ่งหมายถึงจำนวนเต็ม 0. ไม่แสดงผลใด ๆบนหน้าจอ.

unalias

```
unalias [-a] [name] ...
```

ยกเลิก alias.

`-a`	ยกเลิก alias ที่ตั้งไว้ทั้งหมด.
name	ยกเลิก alias เฉพาะชื่อ *name* ที่กำหนดเป็นอาร์กิวเมนต์.

umask

```
umask [value]
```

ตั้งหรือดูค่า umask.

value ค่าบิตที่ไม่อนุญาต.

unset

```
unset name
```

ลบตัวแปรหรือฟังชันที่สร้างในเชลล์.

name ชื่อตัวแปรหรือฟังชัน.

uname

```
uname [-a]
```

แสดงข้อมูลรายละเอียดเกี่ยวกับเคอร์เนลที่ใช้อยู่. ถ้าไม่มีอาร์กิวเมนต์จะแสดงแค่ชื่อเคอร์เนล.

-a แสดงรายละเอียดทั้งหมด.
-n แสดงชื่อเน็ตเวิร์ก (node name) ซึ่งได้แก่ชื่อโฮส.
-r แสดง kernel release ซึ่งหมายถึงรุ่นของเคอร์เนล เช่น 2.6.5-gentoo-r1 เป็นต้น.
-v แสดงรุ่นของเคอร์เนล. หมายถึงเวลาที่สร้างเคอร์เนล.
-m แสดงสถาปัตยกรรมของเครื่อง.
-p แสดงข้อมูลเบื้องต้นของหน่วยประมวลผล.

write

```
write user [tty]
```

เขียนส่งข้อความจาก stdin ไปหาผู้ใช้ที่ล็อกอินอยู่ในระบบ.

user ผู้ใช้ที่ต้องการส่งข้อความไปหา.
tty ชื่อเทอร์มินอลที่ต้องส่งข้อความ. ถ้าไม่ระบุจะส่งข้อความไปหาทุกเทอร์มินอลที่ผู้ใช้ล็อกอินอยู่.

yes

```
yes [string]
```

แสดงตัวอักษร “y” ไปเรื่อย ๆบรรทัดต่อบรรทัดถ้าไม่มีอาร์กิวเมนต์. ถ้ามีอาร์กิวเมนต์ *string* จะแสดงอาร์กิวเมนต์ที่ได้รับแทน “y”.

ข.10 ระบบ X วินโดว์

bdftopcf

```
bdftopcf [-o output.pcf] [font.bdf]
```

แปลงข้อมูลฟอนต์จาก bdf (Bitmap Distribution Format) ให้เป็น pcf (Portable Compiled Format).

`-o output.pcf`	ระบุชื่อไฟล์หลังจากการแปลงข้อมูล. ถ้าไม่ใช้ตัวเลือกนี้, คำสั่งจะแสดงผลออกทาง stdout.
font.bdf	ชื่อไฟล์ฟอนต์. ถ้าไม่ระบุชื่อไฟล์จะใช้ข้อมูลนำเข้าจาก stdin.

fc-cache

```
fc-cache [-f] [-s] [-v]
```

สร้างฐานข้อมูลไฟล์ `fonts.cache` สำหรับระบบจัดการฟอนต์ fontconfig.

- `-f` บังคับให้อัปเดตฐานข้อมูลอีกครั้ง.
- `-s` สร้างฐานข้อมูลสำหรับไดเรกทอรีที่เก็บฟอนต์โดยรวมของระบบเท่านั้น.
- `-v` แสดงข้อมูลการทำงานของคำสั่งทางหน้าจอ.

mkfontscale

```
mkfontscale [-o fonts.scale] [-e enc_dir]          [-a enc] [directory]
```

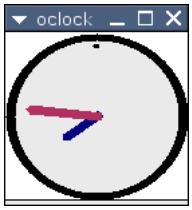
สร้างไฟล์ `fonts.scale` สำหรับเตรียมการติดตั้งฟอนต์.

`-o` *fonts.scale*	ระบุชื่อไฟล์ผลลัพธ์. ถ้าไม่ระบุตัวเลือกนี้จะสร้างไฟล์ `fonts.scale` ในไดเรกทอรีที่สั่งคำสั่ง.
`-e` *enc_dir*	ระบุชื่อไดเรกทอรีที่มีไฟล์ `encodings.dir`. โดยปรกติ, ไฟล์นี้จะอยู่ใต้ไดเรกทอรี `/usr/share/fonts/encodings`.

`-a` *`end`*	ระบุรหัสอักขระเพิ่มเติมตามต้องการ.
`directory`	ไดเรกทอรีที่เก็บฟอนต์. ถ้าไม่ระบุจะถือว่าคือไดเรกทอรีที่สั่งคำสั่ง.

oclock

```
oclock [-transparent]
```



แสดงนาฬิกาทรงกลม.

`-transparent`	แสดงพื้นหลังของนาฬิกาใส.

startx

```
startx [ [ client ] options ... ] [ -- [ server ] options ... ]
```

เชลล์สคริปต์เริ่มต้นระบบ X วินโดว์. เชลล์สคริปต์นี้จะตั้งค่าตัวแปร, เตรียมการต่าง ๆ และสั่งคำสั่ง `xinit` ต่อไป.

`client`	ไฟล์โปรแกรมสำหรับรันเป็นไคล์เอ็นต์แสดงในหน้าจอ.
`options`	ตัวเลือกของโปรแกรมไคล์เอ็นต์.
`--`	ตัวเลือกที่อยู่หน้าเครื่องหมายนี้จะเกี่ยวกับข้องกับไคล์เอ็นต์, ตัวเลือกที่อยู่หลังเครื่องหมายนี้จะเกี่ยวกับเซิร์ฟเวอร์.
`server`	ไฟล์โปรแกรมสำหรับรันเป็นเซิร์ฟเวอร์.
`options`	ตัวเลือกของเซิร์ฟเวอร์.

xclock

```
xclock [-d] [-update seconds]
```

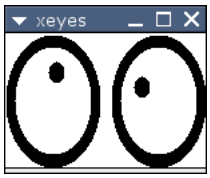
แสดงนาฬิกาทรงเหลี่ยมหรือนาฬิกาดิจิตอล.

`-d`	แสดงนาฬิกาดิจิตอล.
`-update`	อัปเดทหน้าจอทุก ๆ *`seconds`* วินาที.

xeyes

```
xeyes [toolkit_options]
```

แสดงลูกตามองตามตำแหน่งของเมาส์ในหน้าจอ.



`-outline` *`color`* ระบุสีที่ใช้สำหรับขอบตา.

xfd

```
xfd [-fa fontname] [-fn fontname]
```

แสดงอักขระต่างโดยระบุชื่อฟอนต์.

`-fa` ระบุชื่อฟอนต์ *`fontname`* แบบ fontconfig.
`-fn` ระบุชื่อฟอนต์ *`fontname`* แบบ XLFD.

xhost

```
xhost +|-[hostname]
```

ควบคุมการแสดงหน้าต่างในระบบ X วินโดว์.

`+`	อนุญาตให้แสดงผลบนหน้าจอ.
`-`	ไม่อนุญาตให้แสดงผลบนหน้าจอ.
`hostname`	ชื่อโฮสหรือ IP แอดเดรสของเครื่องที่ต้องการอนุญาตให้เข้าถึง X เซิร์ฟเวอร์เฉพาะราย. ถ้าไม่ระบุจะหมายถึงเครื่องใด ๆไม่เจาะจง.

xlsfonts

```
xlsfonts [-l] [-ll] [-lll]
```

แสดงชื่อฟอนต์ในระบบ X วินโดว์.

`-l` แสดงค่าเฉพาะของฟอนต์นอกเหนือจากชื่อฟอนต์อย่างเดียว. ตัวเลือกนี้จะใช้เวลานานในการประมวลผล.
`-ll` แสดงคุณสมบัติของฟอนต์เพิ่มเติมจากตัวเลือก `-l`.
`-lll` แสดงค่าเมตริกของอักขระเป็นข้อมูลเพิ่มเติมจากตัวเลือก `-ll`.

xset

```
xset [q] [fp+ fontpath] [fp rehash]
```

แสดงหรือปรับแต่งพารามิเตอร์ต่าง ๆเฉพาะบุคคลในระบบ X วินโดว์.

`q`	แสดงค่าพารามิเตอร์ต่าง ๆ.

`fp+` *`fontpath`* เพิ่ม *fontpath* ต่อท้ายรายการฟอนต์ไดเรกทอรีที่มีอยู่.
`fp rehash` ให้ X เซิร์ฟเวอร์อ่านฐานข้อมูลฟอนต์อีกครั้ง.

xterm

```
xterm [-fa fontname] [-fs size] [-e program]
      [-fn fontname] [-ls] [-u8]
```

เทอร์มินอลเอมิวเลเตอร์มาตรฐาน.

- `-fa` ระบุชื่อฟอนต์ *`fontname`* ตามรูปแบบชื่อฟอนต์ในไลบรารี Freetype.
- `-fs` ระบุขนาดฟอนต์ *`size`* ตามรูปแบบที่ใช้ในไลบรารี Freetype.
- `-e` ระบุโปรแกรม *`program`* ที่ต้องการรันในเทอร์มินอลทันทีหลังจากเริ่มทำงานเช่น `xterm -e top`.
- `-fn` ระบุชื่อฟอนต์ *`fontname`* ตามรูปแบบ XLFD.
- `-ls` ใช้ล็อกอินเชลล์.
- `-u8` ตีความข้อมูลนำเข้าจากแป้นพิมพ์เป็นข้อมูลที่ลงรหัสอักขระ UTF-8.

xwd

```
xwd [-out output] [-root] [-display name]
```

จับภาพหน้าจอในระบบ X วินโดว์ในฟอร์แมตพิเศษ xwd.

`-out` ชื่อไฟล์ผลลัพธ์ที่ต้องการบันทึก *`output`*. ถ้าไม่ใช้ตัวเลือกนี้จะแสดงผลลัพธ์ทาง stdout.
`-root` จับหน้าจอทั้งหมด. ถ้าไม่ระบุตัวเลือกนี้จะต้องใช้พอยเตอร์เลือกหน้าต่างที่ต้องการจับภาพ.
`-display` ระบุชื่อหน้าจอ *`name`*.

ภาคผนวก ค

ไฟล์ปรับแต่งระบบที่อยู่ใน /etc

ภาคผนวก ง
Debian GNU/Linux

หนังสือเล่มนี้เลือกใช้ Debian GNU/Linux เป็นลินุกซ์ดิสทริบิวชันอ้างอิงเนื่องจากเหตุผลหลาย ๆอย่างและในบทนี้จะแนะนำการติดตั้ง Debian และการจัดการแพ็กเกจซอฟต์แวร์เพื่อที่จะให้ผู้ใช้คุ้นเคยและนำไปใช้งานได้จริงต่อไป.

ง.1 ติดตั้ง Debian GNU/Linux

ขั้นตอนการติดตั้งลินุกซ์ไม่ว่าจะเป็นดิสทริบิวชันใดก็ตามโดยทั่วไปมีขั้นตอนดังนี้.

1. ดาวน์โหลดซีดีในรูปของไฟล์อิมเมจ ISO
2. เขียนซีดีรอม และบูตเครื่องคอมพิวเตอร์จากซีดีรอม
3. ติดตั้งระบบปฏิบัติการลินุกซ์ด้วยโปรแกรมติดตั้งของดิสทริบิวชันที่ใช้
4. ปรับแต่งและอัปเดทซอฟต์แวร์

ง.2 ดาวน์โหลดซีดี

การติดตั้งลินุกซ์โดยทั่วไปมักจะใช้ซีดีรอมเป็นสื่อในการติดตั้ง. ผู้ใช้สามารถซื้อหรือจะดาวน์โหลดด้วยตัวเองจากเว็บไซต์ของดิสทริบิวชันต่าง ๆ.

ซีดีที่ดาว์นโหลดมาจะอยู่ในรูปของไฟล์ที่เรียกว่า*ดิสก์อิมเมจ (disk image)* และมักจะมีส่วนขยายชื่อไฟล์เป็น `.iso`.

ในเว็บไซต์หน้าดาว์นโหลดซีดี Debian [1] จะมีบอกวิธีการดาว์นโหลดหลายแบบเช่น ดาวน์โหลดด้วย jigdo ซึ่งเป็นโปรแกรมพิเศษสำหรับดาว์นโหลดไฟล์ ISO ของ Debian, ดาวน์โหลดโดยใช้ Bittorrent. แต่วิธีที่เข้าใจง่ายที่สุดคือดาว์นโหลดผ่านทาง FTP หรือ HTTP โดยไปที่หน้าดาว์นโหลด Debian CD/DVD ผ่านทาง HTTP/FTP[2] ก่อน. จากนั้นก็เลือก FTP หรือ HTTP ที่ใกล้ตัวเพื่อความรวดเร็วในการดาว์นโหลด.

ส่วนขยายชื่อไฟล์ `.iso` เป็นแค่ชื่อที่ให้คนอ่านแล้วเข้าใจ. จริง ๆแล้วไม่จำเป็นต้องเป็น `.iso`. บ้างก็ใช้ส่วนขยายชื่อไฟล์เป็น `.raw`, `.img` หรือไม่ใส่ส่วนขยายชื่อไฟล์.

[1] http://www.debian.org/CD

[2] http://www.debian.org/CD/http-ftp/

ตัวอย่างเช่นถ้าดาว์นโหลดจาก ftp://cdimage.debina.org/debian-cd/3.1_r0a/i386/iso-cd/ จะมีไฟล์ .iso หลายตัวให้เลือก. ให้เลือกดาว์นโหลดไฟล์ debian-31r0a-i386-binary-1.iso ไฟล์เดียวก็พอเพียงสำหรับการติดตั้ง. ไฟล์อิมเมจนี้สามารถใช้ติดตั้งโดยไม่มีเน็ตเวิร์กได้และมีแพ็กเกจพื้นฐานที่จำเป็นรวมถึงสภาพแวดล้อมเดสก์ท็อปให้ด้วย. ไฟล์ debian-31r0a-i386-businesscard.iso เป็นไฟล์อิมเมจสำหรับ CD ขนาดนามบัตรซึ่งแพ็กเกจที่อยู่อิมเมจก็จะน้อยลง. และไฟล์ debian-31r0a-i386-netinst.iso เป็นไฟล์อิมเมจสำหรับติดตั้งโดยผ่านเน็ตเวิร์กซึ่งแพ็กเกจที่ติดตั้งจะดาว์นโหลดผ่านเน็ตเวิร์กขณะติดตั้ง.

ง.2.1 การตรวจสอบดิสก์อิมเมจ

ไฟล์ดิสก์อิมเมจที่ดาว์นโหลดมามักจะมีขนาดใหญ่ประมาณ 700 MB เพื่อเขียนลงซีดีรอมหนึ่งแผ่น. การดาว์นโหลดไฟล์ใหญ่แบบนี้อาจจะเกิดความผิดพลาดในการโอนถ่ายข้อมูล, ซึ่งบางครั้งนำไฟล์ที่ดาว์นโหลดไปเขียนซีดีแล้วใช้งานไม่ได้. ในบางกรณีที่ผู้ใช้ไม่ได้ดาว์นโหลดไฟล์จากเว็บไซต์ของดิสทริบิวชันอย่างเป็นทางการ, อาจจะมีผู้ไม่ประสงค์ดีแอบใส่โปรแกรมที่ไม่พึงประสงค์ลงไปในไฟล์ติดตั้งด้วย. ด้วยเหตุผลเหล่านี้เองผู้ใช้จึงควรตรวจสอบไฟล์ดิสก์อิมเมจที่ดาว์นโหลดมาก่อนใช้งาน.

ผู้ใช้สามารถตรวจสอบไฟล์ที่ดาว์นโหลดมาด้วยโปรแกรม md5sum.

```
$ md5sum debian-31r0a-i386-binary-1.iso↵
b49a7955be5e286eff873a9bd157793a  debian-31r0a-i386-binary-1.iso
```

โปรแกรม md5sum จะแสดงค่า*เช็คซัม (checksum)* ของไฟล์ที่ต้องการตรวจสอบ. เมื่อได้ค่าเช็คซัมแล้วก็นำไปเทียบกับค่าเช็คซัมที่ดิสทริบิวชันแสดงไว้ว่าตรงกันหรือไม่. ปรกติไซต์ที่ดาว์นโหลดจะมีไฟล์ MD5SUMS ให้ดาว์นโหลดด้วย. ไฟล์นี้จะมีค่าเช็คซัมของไฟล์อิมเมจไว้ให้ดูเปรียบเทียบว่าค่าตรงกันหรือไม่. ถ้าไม่ตรงกันแสดงว่าไฟล์ที่ดาว์นโหลดมามีข้อมูลไม่ตรงกับต้นฉบับจริง.

สำหรับผู้ที่ใช้วินโดวส์และต้องการตรวจค่าเช็คซัม, ให้ใช้โปรแกรม MD5summer [53] ซึ่งเป็นซอฟต์แวร์เสรี.

ง.3 การเขียนซีดี

โปรแกรมที่ใช้เขียนซีดีบนลินุกซ์มีหลายโปรแกรมให้เลือกใช้เช่น xcdroast, gtoaster แต่โปรแกรมเหล่านี้ต่างก็เป็น frontend ของโปรแกรม (คำสั่ง) cdrecord ทั้งนั้น. เมื่อดาว์นโหลดัและตรวจสอบไฟล์ที่ดาว์นโหลดแล้ว, ให้ใช้คำสั่ง cdrecord.

cdrecord dev=0,0,0 เป็นการระบุดีไวส์ซีดี. -eject ให้คอมพิวเตอร์ดีดตัวซีดีออกจากไดรว์เมื่อเขียนซีดีเสร็จ. และ speed=4 เป็นการใช้ความเร็ว 4 เท่า.

```
# cdrecord dev=0,0,0 -eject speed=4 samila-5.5-i386-cd1.iso↵
```

ผู้ที่ใช้วินโดวส์สามารถใช้โปรแกรมเขียนไฟล์ .iso ลงซีดีได้เช่น Nero, Easy CD Creator ฯลฯ. หรือถ้าเป็นซอฟต์แวร์เสรีสามารถใช้ BurnAtOnce[3], CDBruenerXP Pro[4]

[3] http://www.burnatonce.com

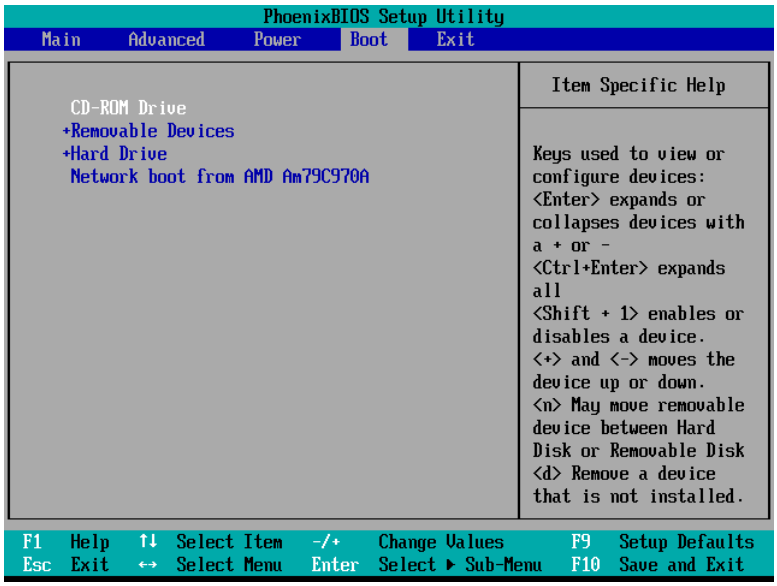
[4] http://www.cdburnerxp.se

ฯลฯ.

ง.4 การติดตั้งลินุกซ์

ก่อนอื่นให้ตรวจสอบ BIOS ของคอมพิวเตอร์ให้เครื่องคอมพิวเตอร์บูตเครื่องจากซีดีได้. จากนั้นให้ใส่ซีดีแล้วเปิดเครื่องใหม่, ถ้าไม่มีข้อผิดพลาดใด ๆก็จะสามารถบูตจากแผ่นซีดีได้.

ตั้งค่า BIOS



เข้าหน้าจอ BIOS และเปลี่ยนให้เครื่องบูตจาก CDROM. วิธีตั้งให้เครื่องเข้า BIOS อาจจะกด F1, Del, ESC แล้วแต่กรณี. หลังจากที่เข้า BIOS แล้งแก้เมนูการบูตให้บูตจาก CDROM, บันทึกการแก้ไขแล้วรีบูต.

เข้าสู่โปรแกรมติดตั้ง

หนังจากที่ใส่แผ่น CD แล้วบูตจากจะมีหน้าจอดังภาพ. ให้กดคีย์ Enter หรือกดคีย์ F1 เพื่อดูคำอธิบายช่วยเหลือ.

... to be continued

บรรณานุกรม

[1] Robert M. Free. Ansi/vt100 terminal control. Website, 1996.
http://www.fh-jena.de/~gmueller/Kurs_halle/esc_vt100.html
เว็บไซต์แหล่งอ้างอิง ANSI escape sequence ต่าง ๆเช่น การควบคุมเคอร์เซอร์, สี, เทอร์มินอล ฯลฯ.

[2] Linus Torvalds and David Diamod. *Just For Fun*. Harper Collins, 2001.
หนังสือเล่มนี้ไม่ใช่หนังสือคอมพิวเตอร์แต่เขียนเกี่ยวกับความเป็นมา, เรื่องราว, เบื้องหน้าเบื้องหลังของระบบปฏิบัติการลินุกซ์จากปากคำของผู้สร้างซึ่งได้แก่นาย Linus เอง.

[3] กรมทรัพย์สินทางปัญญา. Website.
http://www.ipthailand.org
เว็บไซต์ของกรมทรัพย์สินทางปัญญาของไทย.
มีข้อมูลเกี่ยวกับลิขสิทธิ์และสิทธิบัตร.

[4] The linux kernel archives. Website.
http://www.kernel.org
เป็นแหล่งรวมรหัสต้นฉบับของลินุกซ์เคอร์เนลตั้งแต่ต้นจนถึงปัจจุบัน.

[5] Free Software Foundation. *GNU General Public License*, version 2 edition, June 1991.
http://www.gnu.org/copyleft/gpl.html
เนื้อหาฉบับเต็มของหนังสืออนุญาตแบบ GPL.

[6] Free Sofware Foundation. Various licenses and comments about them. Website.
http://www.gnu.org/licenses/license-list.html
อธิบายเกี่ยวกับหนังสืออนุญาตแบบต่าง ๆและเปรียบเทียบความแตกต่าง.

[7] Free Software Foundation. Boycott amazon! Website, 2001.
http://www.gnu.org/philosophy/amazon.html
เป็นเว็บไซต์เพื่อการต่อต้าน Amazon.com เนื่องจากทาง Amazon จดสิทธิบัตรเกี่ยวกับวิธีการซื้อขายสินค้าทางอินเทอร์เน็ตซึ่งเป็นที่วิภาษณ์วิจารย์อย่างกว้างขวาง.

[8] Free software foundation. Website.
http://www.gnu.org

เว็บไซต์ขององค์กร Free Software Foundation มีซอฟต์แวร์ให้ดาว์นโหลดและเอกสารให้อ่านมากมาย.

[9] Free Software Foundation. The free software definition. Website. http://www.gnu.org/philosophy/free-sw.html
อธิบายคำจำกับความของคำว่า Free Software โดยมีการแบ่งระดับของความเสรีในระดับต่าง ๆ.

[10] Open Source Initiative OSI. The open source definition. Website. http://www.opensource.org/docs/definition.php
เว็บไซต์เกี่ยวกับคำจำกับความของคำว่า Open Source.

[11] Red Hat. History of linux at red hat. Website, 2003. http://fedora.redhat.com/about/history
เป็นเว็บไซต์ที่เขียนเกี่ยวกับประวัติของ Red Hat Linux ตั้งแต่เริ่มต้นจนถึงปัจจุบัน. มีระบุเดือนปีของการออกรุ่นต่าง ๆและชื่อรหัสของแต่ละรุ่น.

[12] Mark J Cox. End of life for red hat linux 7.1, 7.2, 7.3, 8.0. Technical report, Red Hat, December 2003.
https://listman.redhat.com/archives/redhat-watch-list/2003-December/msg00004.html
เมลล์เป็นทางการแจก Red Hat ประกาศการหมดอายุขัยของ Red Hat Linux.

[13] Red Hat Inc. Red hat enterprise linux. Website. http://www.redhat.com/software/rhel
เว็บไซต์แนะนำสินค้าและบริการของ Red Hat Enterprise Linux.

[14] Fedora project. Website. http://fedora.redhat.com/
เว็บไซต์แหล่งข้อมูลของ Fedora Project ที่สนับสนุนโดย Red Hat.

[15] Bdale Garbee, Hartmut Koptein, Nils Lohner, Will Lowe, Bill Mitchell, Ian Murdock, Martin Schulze, and Craig Small. *A Brief History of Debian*. Debian Project, 2.4 edition, January 2003. http://www.debian.org/doc/manuals/project-history
เป็นเอกสารจาก Debian Project เกี่ยวกับประวัติของโปรเจคโดยละเอียด.

[16] Fedora linux home page. Website. http://www.fedora.us
เว็บไซต์ดั้งเดิมของ Fedora Project ก่อนที่จะได้รับการสนับสนุนเป็นทางการจาก Red Hat.

[17] ภัทระ เกียรติเสรี. Knoppix thai. Website, มีนาคม 2004. http://linux.thai.net/plone/Members/ott/knoppix-th

เว็บไซต์แนะนำ Knoppix ภาษาไทยและที่ดาว์นโหลด.

[18] ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC). Lexitron thai <-> english dictionary. Website.
http://lexitron.nectec.or.th
เว็บไซต์เป็นทางการของพจานุกรมอิเล็กทรอนิกส์ LEXiTRON.
ผู้ที่สนใจสามารถดาว์นโหลดตัวซอฟต์แวร์หรือใช้บริการทางเว็บก็ได้.
ผู้ใช้ยังสามารถแนะนำคำศัพท์ใหม่, ส่งเรื่องราวสาระเกี่ยวกับภาษา ฯลฯ
ให้แก่ทีมงานได้ด้วย.

[19] ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC). อ่านไทย arn-thai. Website.
http://www.nectec.or.th/sll/thaiocr/menuth.htm
เว็บไซต์เป็นทางการของโครงการอ่านไทย.

[20] Distrowatch.com, put the fun back into computing. use linux. Website.
http://www.distrowatch.com
เป็นที่รวมข้อมูลของดิสทริบิวชันต่าง ๆทั่วโลก, รวมถึงบทความรีวิว,
ข่าวล่าสุดของดิสทริบิวชันต่าง ๆ.

[21] Linux tle. Website.
http://www.opentle.org
เว็บไซต์หลักของ Linux TLE. สามารถดาว์นโหลด Linux TLE ได้จากที่นี่.
นอกจากนั้นยังมีเว็บบอร์ดสนับสนุนผู้ใช้, ข่าว, บทความ ฯลฯ.

[22] บูรพาลินุกซ์ (burapha linux). Website.
http://www.buraphalinux.org
เป็นเว็บไซต์หลักของบูรพาลินุกซ์แนะนำโครงการ.
มีเอกสารการใช้ลินุกซ์ภาษาไทยประกอบด้วย.

[23] Thai linux working group. Website.
http://linux.thai.net
เว็บไซต์ของ Thai Linux Working Group
มีเว็บบอร์ดสื่อสารกัน, ข้อมูลต่าง ๆที่เป็นประโยชน์มากมายเป็นภาษาไทย.
เว็บบอร์ดของที่นี่จริง ๆแล้วเป็นการเชื่อมเว็บกับ mailing list ที่อยู่ที่ yahoogroup-s.com และนิวส์กรุปที่อยู่ที่ thaigate.nii.ac.jp.

[24] Berny Goodheart and James Cox. *The Magic Garden Explained: The Internals of UNIX System V Release 4.* Prentice Hall, 1994. หนังสือเรียนเกี่ยวกับระบบปฏิบัติการยูนิกซ์. อธิบายถึงโครงสร้างภายใน.
เนื้อหาไม่เกี่ยวกับลินุกซ์โดยตรงแต่หลักการคล้าย ๆกัน.

[25] Uresh Vahalia. *UNIX Internal: The New Frontiers*. Prentice Hall, 1996. หนังสือเรียนเกี่ยวกับระบบปฏิบัติการยูนิกซ์. อธิบายถึงโครงสร้างภายใน. เนื้อหาไม่เกี่ยวกับลินุกซ์โดยตรงแต่หลักการคล้าย ๆกัน.

[26] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1992. เป็นหนังสือเรียนเกี่ยวกับระบบปฏิบัติการโดยเน้นระบบปฏิบัติการ Minix ซึ่งศาสตราจารย์ Andrew Tanenbaum เป็นคนสร้างเอง. หนังสือเล่มนี้เป็นแรงบรรดาลใจให้นาย Linus Torvalds สร้างระบบปฏิบัติการลินุกซ์. หลังจากที่ลินุกซ์เป็นที่รู้จักทางนิวส์กรุป, มีการถกเถียง (เชิงทะเลาะ) กันระหว่างนาย Linus Torvalds กับศาสตราจารย์ Andrew Tanenbaum เป็นที่เลื่องลือในประวัติศาสตร์.

[27] The linux document project. Website.
`http://www.tldp.org`
เว็บไซต์ที่รวบรวมเอกสารที่เกี่ยวกับลินุกซ์. มีเอกสารที่เรียกว่า HOWTO เขียนโดยอาสาสมัครที่สนใจการใช้งานลินุกซ์ด้านต่าง ๆ. นอกจากเอกสาร HOWTO แล้วยังมีเอกสาร Guide เป็นหนังสือเกี่ยวกับการใช้งานลินุกซ์ในด้านต่าง ๆเผยแพร่ฟรี.

[28] Giles Orr. The bash prompt howto. Website, November 2003.
`http://www.gilesorr.com/bashprompt/howto/book1.html`
เป็นเอกสารที่อธิบายเทคนิควิธีการตั้งค่าพรอมต์ของ bash. ในเอกสารอธิบายความรู้พื้นฐานเกี่ยวกับเชลล์, เทอร์มินอลและแสดงตัวอย่างการตั้งค่าพรอมต์ชั้นสูง (ไม่ธรรมดา) ด้วย.

[29] Brian W. Kernighan and Rob Pike. *The Unix Programming Environment*. Prentice Hall, 1984.
เป็นหนังสือที่น่าอ่านอย่างยิ่งสำหรับผู้ที่เริ่มใช้และต้องการเรียนรู้การใช้งานยูนิกซ์.
เนื้อหาในหนังสือบางอย่างล้าสมัยไปแล้วแต่โดยทั่วไปยังคงความเป็นอมตะอยู่และใช้ได้กับลินุกซ์เช่นกั

[30] Filesystem Hierarchy Standard Group. Filesystem hierarchy standard. Website, 1994-2004.
`http://www.pathname.com/fhs`
เว็บไซต์แหล่งอ้างอิงเกี่ยวกับโครงสร้างระบบไฟล์ของลินุกซ์.
ผู้อ่านสามารถอ่านรายละเอียดของไดเรกทอรีและไฟล์ที่พึงมีในระบบได้.

[31] รวมเอกสารจากหลายแหล่ง. linux/documentation. ไฟล์.
`/usr/src/linux/Documentation`
ในรหัสต้นฉบับของลินุกซ์เคอร์เนลจะมีเอกสารแบบข้อความธรรมดาเป็นไฟล์หลายฉบับที่เกี่ยวข้องกับ
ข้อมูลที่อยู่ในไฟล์เหล่านี้มีประโยชน์มากสำหรับผู้ที่ต้องการเรียนรู้เรื่องเกี่ยวกับเคอร์เนลและลินุกซ์อย่

[32] Dale Dougherty และ Arnold Robbins. *sed & awk*. O'Reilly & Associates, 1997.
เป็นหนังสือที่อธิบายสอนวิธีการใช้คำสั่งประมวลผลข้อมูลเท็กซ์ที่สำคัญได้แก่ `sed` และ `awk`. ระยะหลังมีการทำเป็นดิจิตอลและหาอ่านได้ตามอินเทอร์เน็ต.

[33] The gzip home page. Website.
`http://www.gzip.org/`
เป็นเว็บไซต์อย่างเป็นทางการของโปรแกรม `gnuzip`. มีคำแนะนำโปรแกรม, ที่ดาว์นโหลดและคำถามที่ถามบ่อย.

[34] The bzip2 and libbzip2 home page. Website.
`http://sources.redhat.com/bzip2/`
เป็นเว็บไซต์อย่างเป็นทางการของโปรแกรม `bzip2`. มีคำอธิบาย, เอกสาร, แหล่งดาว์นโหลดของโปรแกรม `bzip2` และ libbzip2 ซึ่งเป็นไลบรารีที่เกี่ยวข้องกัน.

[35] Info-zip home page. Website.
`http://www.info-zip.org/`
เป็นเว็บไซต์ของโปรแกรม unzip/zip ที่ใช้ในลินุกซ์. นอกจากโปรแกร `unzip` สำหรับยูนิกซ์แล้วยังมีโปรแกรม zip สำหรับวินโดวส์และ Mac ด้วย.

[36] Jonathan B. Postel. Rfc 821 - simple mail transfer protocol. Website, August 1982.
`http://www.faqs.org/rfcs/rfc821.html`
เอกสาร RFC ที่อธิบายเกี่ยวกับโปรโตคอลรับส่งเมล.

[37] Machtelt Garrels. Bash guide for beginners. Website, October 2004.
`http://www.tldp.org/LDP/Bash-Beginners-Guide/html/index.html`
หนังสืออิเล็กทรอนิกส์หนึ่งใน The Linux Documen-t Project แนะนำการเขียนเชลล์สคริปต์ด้วย bash. เหมาะสำหรับผู้ใช้ลินุกซ์อย่างจริงจังหรือผู้ดูแลระบบ.

[38] Mendel Cooper. Advanced bash-scripting guide. Website, October 2004.
`http://www.tldp.org/LDP/abs/html/index.html`
หนังสืออิเล็กทรอนิกส์หนึ่งใน The Linux Documen-t Project แนะนำการเขียนเชลล์สคริปต์ด้วย bash. เหมาะสำหรับผู้ที่เขียนเชลล์สคริปต์เป็นแล้วต้องการจะศึกษาการใช้งานชั้นสูงต่อไป.

[39] Marco Cesati Daniel P.Bovet. *Understanding The Linux Kernel*. O'Reilly & Associates, Inc, 2001.
หนังสืออธิบายลินุกซ์เคอร์เนลเหมาะสำหรับผู้ที่ต้องการเรียนรู้เคอร์เนลอย่างจริงจัง.

[40] The IEEE and The Open Group. The single unix specification version 3. Website, 2004.
http://www.unix.org/version3
เอกสารกำหนดมาตรฐานต่าง ๆของระบบปฏิบัติการยูนิกซ์.
มาตรฐานที่กำหนดประกอบด้วย Base Definition, Shell & Utilities, System Interfaces และ Rationale.

[41] Ingo Molnar Ulrich Drepper. The native posix thread library for linux. Website, 2003.
http://people.redhat.com/drepper/nptl-design.pdf
เอกสารเกี่ยวกับ NPTL โดยผู้พัฒนาไลบรารี. อธิบายประวัติ, ปัญหา, การออกแบบไลบรารี thread สำหรับลินุกซ์.

[42] Paul E. Kimball. *The X Toolkit Cookbook*. Prentice Hall International, Inc, 1995.
หนังสือสอนเขียนโปรแกรม GUI ใช้บน X วินโดว์ด้วยภาษาซี. ไลบรารีที่ใช้ได้แก่ Xt, Athena และ Motif ซึ่งไม่นิยมใช้ในปัจจุบัน.

[43] Ph.D Thaweesak Koanantakool. The keyboard layouts and input method of the thai language. Website.
http://www.nectec.or.th/it-standards/keyboard_layout/thai-key
เอกสารวิชาการเกี่ยวกับประวัติแป้นพิมพ์ภาษาไทยและคุณสมบัติต่าง ๆ.

[44] สำนักงานมาตรฐานผลิตภัณฑ์อุตสาหกรรม กระทรวงอุตสาหกรรม. การกำหนดตำแหน่งอักขระไทย บนผังแป้นพิมพ์คอมพิวเตอร์. Website, 1995.
http://www.nectec.or.th/it-standards/std820/std820.htm
เอกสารรายละเอียดเกี่ยวกับมาตรฐานผังแป้นพิมพ์ TIS-820.2538.

[45] Kazutaka Yokota. Mouse support in x11r6.8.1. Website, 2002.
http://www.x.org/X11R6.8.1/doc/mouse.html
เอกสารเกี่ยวกับปรับแต่งเมาส์สำหรับระบบ X วินโดว์.

[46] RealVNC Ltd. Realvnc. Website.
http://www.realvnc.com/
เว็บไซต์ของบริษัทพัฒนาซอฟต์แวร์ VNC.
สามารถดาว์นโหลดโปรแกรมเซิร์ฟเวอร์และไคลเอ็นต์ได้จากที่นี่ทุกแพล็ตฟอร์ม.

[47] Juliusz Chroboczek. xfsft: Truetype font support for x. Website, 1998-1999.
http://www.dcs.ed.ac.uk/home/jec/programs/xfsft/

เว็บไซต์ที่เกี่ยวกับโมดูล xfsft ของ X เซิร์ฟเวอร์ซึ่งในช่วงแรก ๆเป็นโมดูลช่วยทำให้ X เซิร์ฟเวอร์ใช้ฟอนต์ทรูไทป์ได้. ปัจจุบันโครงการนี้ถูกรวมไว้ในตัว X เซิร์ฟเวอร์แล้ว.

[48] The freetype project. Website.
http://www.freetype.org/
เว็บไซต์ของโครงการ FreeType.
เป็นไลบรารีพื้นฐานสำหรับจัดการกับฟอนต์ทรูไทป์และใช้โดยโครงการสำคัญ ๆหลายโครงการ.

[49] Standard for thai character codes for computers. Website, สิงหาคม พ.ศ.2533.
http://www.nectec.or.th/it-standards/std620/std620.htm
เอกสารรหัสอักขระไทยที่ใช้กับคอมพิวเตอร์โดยมาตรฐานผลิตภัณฑ์อุตสาหกรรม.

[50] F. Yergeau. Rfc 2279 - utf-8, a transformation format of iso 10646. Website, January 1998.
http://www.faqs.org/rfcs/rfc2279.html
เอกสาร RFC ที่อธิบายเกี่ยวกับการเอ็นโค้ดแบบ UTF-8.

[51] George Coulouris. Bits of history. Website, September 1998.
http://www.dcs.qmul.ac.uk/~{}george/history
เว็บไซต์ของ George Coulouris ผู้สร้างบรรณาธิกรณ์ em เล่าถึงความเป็นมาของ vi.

[52] Jamie Zawinski. Emacs timeline. Website, Feb 2004.
http://www.jwz.org/doc/emacs-timeline.html
เว็บไซต์ของ Jamie Zawinski แฮ็กเกอร์ที่มีชื่อเสียงจากการสร้าง Lucid Emacs และ Netscape. เอกสารฉบับนี้แสดงความเป็นมาของบรรณาธิกรณ์ Emacs.

[53] Luke Pascoe. Home of md5summer, windows md5 sum generator. Website.
http://www.md5summer.org/
ซอฟต์แวร์เสรีแบบ GUI สำหรับคำนวณค่าเช็คซัม MD5 สำหรับวินโดวส์. ใช้ตรวจสอบไฟล์ .iso หลังจากที่ดาว์นโหลดเรียบร้อยแล้ว.

[54] Simon Josefsson (บรรณาธิการ). Rfc 3548 - the base16, base32, and base64 data encodings. Website, July 2003.
http://www.faqs.org/rfcs/rfc3548.html
เอกสาร RFC ที่อธิบายเกี่ยวกับการเข้ารหัสข้อมูลแบบ Base16, Base32 และ Base64.

ดรรชนี

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

ก

ข

ค

จ

ช

ซ

ด

ต

ถ

ท

ธ

น

บ

ป

พ

ฟ

ภ

ม

ย

ล

ว

ศ

ส

ห

อ

ฮ

ดรรชนีคำสั่ง, โปรแกรม

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z

รวมคำศัพท์คอมพิวเตอร์

A

address space

หน่วยความจำเสมือน (virtual memory) ที่โปรเซสสามารถใช้ได้. เคอร์เนลจะแม็ป (map) address space เข้ากับหน่วยความจำจริง (physical memory) เวลาใช้งาน.

archive

การเก็บไฟล์, เอกสารสำคัญไว้. ถ้าเป็นคำนามจะหมายถึงเอกสารที่เก็บไว้เช่นเอกสารสำรองหรือเอกสารสำคัญ.

assembly language

ภาษาแอสเซมบลี. ภาษาคอมพิวเตอร์ชั้นต่ำ, มีความขึ้นอยู่กับสถาปัตยกรรมที่ใช้มากทำให้ยากต่อการพอร์ตโปรแกรมไปใช้กับสถาปัตยกรรมอื่น ๆ.

B

base64

วิธีการเข้ารหัสข้อมูลไบนารีให้เป็นเท็กซ์ ASCII โดยใช้อักขระ 65 ตัวได้แก่ A-Z, a-z, 0-9, +, / และ = [54].

batch

การประมวลผลข้อมูลโดยรวบรวมกระทำตามลำดับที่กำหนดไว้.

Bezier

เส้นโค้ง Bézier เป็นเส้นโค้งคณิตศาสตร์แบบพารามิเตอร์. นิยมใช้ในโปรแกรมกราฟิกต่าง ๆเช่น Gimp, Inkscape ฯลฯ. รูปทรงของฟอนต์ทรูไทป์เป็นเส้นโค้ง Quadratic Bezier ซึ่งอันดับ (เลขยกกำลัง) ของพารามิเตอร์เป็น 2. ส่วนฟอนต์ PostScript ใช้เส้นโค้ง Cubic Bezier มีอันดับของพารามิเตอร์เป็น 3.

boot loader

buffer overflow

เป็นอาการที่โปรแกรมไม่สามารถรันข้อมูลเข้าทำให้ล้นพื้นที่ที่รองรับ (buffer). ในกรณีที่โปรแกรมนั้นออกแบบมาไม่ได้, อาจจะทำให้ระบบเกิดความเสียหายหรือเป็นลู่ทางให้ผู้ที่รันโปรแกรมที่มีค่าบิต suid สั่งคำสั่งหรือกระทำการที่ root กระทำได้.

bug

บั๊ก, ข้อบกพร่อง. ข้อบกพร่องของซอฟต์แวร์หลังจากที่เจ้าของซอฟต์แวร์นั้นประกาศออกตัวซอฟต์แวร์นั้น

byte

หน่วยของข้อมูล. จำนวนหนึ่งไบต์สามารถแสดงหรือเก็บค่าได้ตั้งแต่ 0 ถึง 255.

C

cache

การเก็บข้อมูลบางอย่างไว้ช่วยคราวเพื่อความรวดเร็วในการเข้าถึงข้อมูลนั้นภายหลัง. ถ้าใช้คำว่า cache อย่างเดียวไม่สามารถบอกได้ว่าเป็น cache ของอะไร. ลินุกซ์เคอร์เนลใช้หน่วยความจำในการ cache ข้อมูลหลายอย่างเช่น buffer cache, page cache, inode cache, directory cache และ swap cache.

checksum

วิธีการตรวจสอบความผิดพลาดของข้อมูล, หรือค่าที่ได้จากการตรวจสอบ. ค่าเหล่านี้สามารถบอกได้ว่าข้อมูลที่ได้รับเช่นข้อมูลผ่านทางเน็ตเวิร์กมีข้อผิดพลาดหรือไม่. วิธีการคำนวณและหาค่าเหล่านี้มีหลายวิธีเช่น cyclic redundancy checks, cryptographic message digest. วิธีแบบ cryptographic message digest ยังมีหลายวิธีเช่น MD5 เป็นต้น.

C language

ภาษาซี. ภาษาคอมพิวเตอร์ที่มนุษย์สามารถอ่านทำความเข้าใจได้. เป็นภาษาชั้นสูงที่มีไวยกรณ์แน่นอน. โดยปรกติผู้ใช้จะคอมไพล์รหัสต้นฉบับที่เขียนด้วยภาษาซีเพื่อแปลงเป็นภาษาเครื่องกลที่คอมพิวเตอร์เข้าใจซึ่งเรียกว่าโปรแกรม. ภาษาซีสร้างโดย Dennis Ritchie นักวิจัยของ Bell Laboratories ในราวปี ค.ศ. 1972. ภาษาซีเป็นภาษาใช้งานทั่วไป (general purpose) และใช้ในสร้างระบบปฏิบัติการยูนิกซ์ในเวลาต่อมา. ลินุกซ์เคอร์เนลและโปรแกรมใช้งานส่วนใหญ่ก็ใช้ภาษาซีเขียนเช่นกัน.

code

ส่วนที่เป็นข้อมูลสำหรับหน่วยประมวลผลสั่งคำสั่ง.

command

คำสั่ง. คำหรือตัวอักษรที่พิมพ์ทางแป้นพิมพ์ส่งต่อให้เชลล์แปลความหมายและกระทำการต่อไป. ความหมายทั่วไปยังหมายถึงโปรแกรมหรือชื่อโปรแกรมที่เรียกใช้ผ่านเชลล์.

command line interface

ระบบอินเทอร์เฟสที่ใช้คีย์บอร์ดและหน้าจอแสดงตัวอักษรเป็นหลัก. ถ้าเป็นงานที่ไม่ต้องการกราฟฟิกจะเป็นอินเทอร์เฟสที่สะดวกมากและปฏิบัติการได้เร็วกว่า graphical user interface.

core

core หรือเรียกอีกอย่างว่า core dump เป็นไฟล์ที่บันทึกเนื้อหาของหน่วยความจำเวลาเวลาโปรแกรมทำงานล้มเหลว (crash). ผู้พัฒนาสามารถใช้ข้อมูลที่ได้จากไฟล์นี้ในการหาข้อบกพร่องของโปรแกรมได้.

csv

เป็นคำย่อของ comma separated values. เป็นรูปแบบไฟล์ใช้บันทึกรายการ (record) เป็นบรรทัด ๆ. ภายในบรรทัดสามารถมีค่าได้หลายคอลัมน์ (field) และมักใช้เครื่องหมาย comma เป็นต้นแบ่งคอลัมน์. ไฟล์แบบนี้สามารถสร้างได้ง่าย ๆ ด้วยบรรณาธิกรณ์ทั่วไปหรือโปรแกรม spreadsheet. ในระบบลินุกซ์มีไฟล์แบบนี้เหมือนกันเช่นไฟล์ `/etc/passwd`, `/etc/group` แต่จะใช้เครื่องหมาย colon เป็นตัวแบ่งคอลัมน์.

CVS

Concurrent Versions Systems. เป็นระบบที่ช่วยเก็บและควบคุมเวอร์ชันของรหัสต้นฉบับ. มีประโยชน์อย่างยิ่งโดยเฉพาะการพัฒนาซอฟต์แวร์ร่วมกันหลายคนโดยผ่านทางเน็ตเวิร์ก, ไม่มีข้อจำกัดเรื่องที่อยู่ของนักพัฒนาซอฟต์แวร์.

daemon

โปรเซสแบบ background ที่มีโปรเซส ID แม่ (PPID) เป็น 1 (โปรเซส `init`). โปรเซสเหล่านี้มักจะเป็นโปรเซสที่ทำงานโดยอัตโนมัติถูกสร้างตอนที่ระบบเริ่มทำงาน. โปรเซสเหล่านี้บางตัวอาจเป็นโปรเซสที่ช่วยดูแลระบบ, เซิฟร์เวอร์ เช่น `cron`, `sendmail` เป็นต้น.

data

ส่วนที่เป็นข้อมูลตัวแปรส่วนกลาง (global variable) ของโปรเซส.

debug

การกำจัดข้อมผิดพลาดของโปรแกรม. ข้อผิดพลาดนี้มักเป็นข้อผิดพลาดที่ไม่คาดคิดจนกระทั่งใช้งานจริงที่เรียกว่า runtime error.

Desktop Environment

ระบบ, สภาพแวดล้อมที่เสนอกราฟฟิกอินเทอร์เฟสสำหรับผู้ใช้โดยที่อินเทอร์เฟสเหล่านั้นมีความเข้ากันได้ดีเช่นรูปร่างหน้าตาเหมือนหรือคล้ายกัน, ทำงานร่วมกันได้ ฯลฯ. ตัวอย่าง Desktop Environment ที่นิยมได้แก่ GNOME, KDE เป็นต้น.

dump terminal

เป็นเทอร์มินอลโบราณรุ่นแรก ๆ. ไม่มีความสามารถในการจัดแต่งหน้าจอ, ไม่มีสี (ขาวดำ) หรือมีขีดจำกัดอื่น ๆ. กล่าวคือเป็นเทอร์มินอลแบบง่าย ๆ, มีเพียงคุณสมบัติเบื้องต้นที่พอจะใช้งานได้เท่านั้น.

event

อีเวนท์. เหตุการณ์ต่าง ๆที่เกิดขึ้น. เป็นคำศัพท์ที่ใช้ในระบบ GUI. ตัวอย่างเช่น การคลิ้กเมาส์ทำให้เกิดอีเวนท์ (เหตุการณ์) การคลิ้ก. ตัว X เซิร์ฟเวอร์จะส่งข้อมูลเกี่ยวกับเหตุการณ์นั้นไปให้ไคลเอ็นต์.

expression

นิพจน์. ประโยคที่แสดงความหมายอย่างใดอย่างหนึ่ง.

F

FAQ (Frequently Asked Questions

การรวมคำถามและคำตอบที่ถามกันบ่อยสรุปไว้เพื่อเป็นข้อมูลสำหรับคนอื่นที่มีคำถามเดียวกัน.

file

ไฟล์, แฟ้มข้อมูล. ไฟล์คือข้อมูลซึ่งอาจจะเก็บบันทึกในหน่วยความจำถาวรเช่น ฮาร์ดดิกส์เป็นต้น. ข้อมูลที่เรียกว่าไฟล์นี้มีลำดับ, กล่าวคือข้อมูลที่บันทึกก่อนจะอยู่ในช่วงต้นของไฟล์. ข้อมูลที่เก็บล่าสุดจะอยู่ในช่วงท้ายของไฟล์. เรียกการเก็บข้อมูลแบบนี้ว่า first in first out.

file descriptor

ตัวเลขจำนวนเต็มที่ระบบปฏิบัติการใช้อ้างอิงไฟล์ที่เปิดไว้. โดยปรกติจะมีการกำหนดค่า file descriptor ให้กับ stdin, stdout และ stderr โดยปริยายเป็น 0, 1 และ 2 ตามลำดับ.

file system

ระบบไฟล์. วิธีการที่ระบบปฏิบัติใช้ในการเข้าถึงข้อมูลที่อยู่ในหน่วยความจำถาวร. ระบบไฟล์ที่ใช้ในลินุกซ์มีหลายประเภทได้เช่น ext2, ext3, xfs ฯลฯ.

FQDN

เป็นชื่อย่อของคำว่า Fully Qualified Domain Name. ชื่อเป็นทางการที่ตั้งให้กับคอมพิวเตอร์. ชื่อนี้จะประกอบด้วยชื่อโฮสและชื่อโดเมน. ชื่อ FQDN นี้จะบันทึกกับระบบ DNS ด้วย.

front-end

โปรแกรมที่เรียกใช้โปรแกรมเฉพาะต่ออีกทีหนึ่ง. เช่นโปรแกรม `mkfs` เรียกใช้โปรแกรม `mkfs.ext2` หรือ `mkfs.xfs` อีกทอดหนึ่งแล้วแต่ตัวเลือกที่ใช้กับ `mkfs`. โปรแกรม front-end เหล่านี้เป็นเพียงอินเทอร์เฟส, ไม่ได้เป็นตัวที่ทำงานหลักเอง.

G

gif

เป็นคำย่อของ Graphics Interchange Format ฟอร์แมตไฟล์รูปภาพบิตแมปสี. มีการอัดบีบข้อมูล, ใช้ดรรชนีสีแสดงสีที่มีอยู่ในรูป. ไฟล์รูปแบบนี้เป็นนิยมใช้ในเว็บ แต่หลังจากที่มีปัญหาสิทธิบัตรเกี่ยวกับการบีบอัดข้อมูล, ไฟล์รูปแบบอื่นเช่น jpg และ png ก็เป็นที่นิยมถัดมา.

graphical user interface (GUI)

อินเทอร์เฟสแบบกราฟฟิกที่แสดงหน้าต่าง, ปุ่มกด, เมนู ทางหน้าจอ. ผู้ใช้จะใช้เมาส์หรือคีย์บอร์ดในการโต้ตอบกันระบบ.

graphic mode

กราฟฟิกโหมด. สภาพที่ระบบปฏิบัติการใช้อินเทอร์เฟสแบบกราฟฟิกติดต่อกับผู้ใช้. โดยทั่วไปคอมพิวเตอร์แบบกราฟฟิกโหมดสามารถผลเป็นแบบหน้าต่าง, สร้างปุ่ม, เมนู ฯลฯ. ผู้ใช้โต้ตอบกับคอมพิวเตอร์ด้วยคีย์บอร์ดและเมาส์.

header file

H

high-level language

ภาษาชั้นสูง. หมายถึงภาษาคอมพิวเตอร์ที่มนุษย์อ่านแล้วทำความเข้าใจได้, ไวยกรณ์ไม่ขึ้นกับเครื่องคอมพิวเตอร์ที่ใช้งาน. ตัวอย่างของภาษาชั้นสูงได้แก่ C, C++, Java เป็นต้น.

home directory

โฮมไดเรกทอรี. ไดเรกทอรีที่ผู้ใช้เป็นเจ้าของ. เป็นไดเรกทอรีเริ่มต้นเวลาล็อกอินเข้าสู่ระบบหรือเรียกใช้เทอร์มินัลเอมิวเลเตอร์.

HTML (Hyper Text Markup Language)

มาตรฐานสำหรับเขียนเอกสารที่เผยแพร่ทาง World Wide Web. รูปแบบของไฟล์เป็นเนื้อหาเท็กซ์ที่มนุษย์อ่านแล้วเข้าใจได้. ใช้การเขียนกำกับส่วนที่เป็นหัวข้อ, เนื้อหา, รูปภาพ ฯลฯ. โดยปรกติจะใช้เว็บบราวเซอร์ (web browser) ดูไฟล์ HTML. เว็บบราวเซอร์จะทำหน้าที่จัดหน้าจอภาพตามที่เขียนกำกับไว้ในไฟล์.

interface

I

อินเทอร์เฟส. หมายถึงตัวเชื่อมหรือตัวกลางระหว่างของสองสิ่ง. ตัวอย่างเช่นเชลล์เป็นอินเทอร์เฟสแบบคำสั่งบรรทัดระหว่างยูสเซอร์กับเคอร์เนล. X วินโดว์เป็นอินเทอร์เฟสแบบหน้าต่างใช้เมาส์และคีย์บอร์ดในการป้อนข้อมูล

internationalization

ความสามารถที่โปรแกรมสามารถปรับตัวให้เข้ากับสภาพแวดล้อมภาษาและวัฒนธรรมที่ผู้ใช้กำหนด. ตัวอย่างเช่นโปรแกรมสามารถแสดงผลนอกเหนือจากภาษาอังกฤษได้ถ้าสภาพแวดล้อมของผู้ใช้ไม่ใช่ภาษาอังกฤษ. Internationalization มีคำย่อว่า i18n โดยตัวเลข 18 คือจำนวนอักษรระหว่างอักษร i และ n.

interpreter

โปรแกรมแปลคำสั่ง, *ตัวแปรคำสั่ง*. ภาษาคอมพิวเตอร์แบบหนึ่งซึ่งจะรับข้อมูลเท็กซ์, แปลความหมาย, แล้วกระทำการ. ภาษาคอมพิวเตอร์แบบนี้มีข้อดีที่พัฒนาได้รวดเร็ว, ไม่ต้องคอมไพล์ และมักจะมีวิธีการจัดการหน่วยความจำให้โดยอัตโนมัติ. ข้อเสียของภาษาคอมพิวเตอร์แบบนี้คือจะทำงานช้ากว่าโปรแกรมที่สร้างด้วยคอมไพเลอร์. โปรแกรมแปลภาษาที่นิยมใช้กันได้แก่ เชลล์, Perl, Python, Ruby ฯลฯ.

J

jpg

เป็นคำย่อของ Joint Photographic Experts Group. รูปภาพแบบบิตแมปที่มีการอัดบีบข้อมูลและมักใช้กับไฟล์รูปภาพบนอินเทอร์เน็ต.

K

kerning

การปรับระดับ, ตำแหน่งของอักขระเมื่อมีการแสดงผลร่วมกับอักขระอื่นให้เหมาะสมสวยงาม.

key binding

การสร้างความสัมพันธ์ของโปรแกรมระหว่างคีย์ที่ผู้ใช้กดและการกระทำของโปรแกรมที่เตรียมไว้.

L

load average

โหลดโดยเฉลี่ย. ค่าเฉลี่ยของจำนวนโปรเซสที่ active อยู่ในระบบ. ค่านี้จะแสดงให้ทราบว่าระบบยุ่งหรือไม่. ค่านี้ไม่จำเป็นต้องสัมพันธ์กับเปอร์เซ็นต์การใช้งานของหน่วยประมวลผล.

M

machine language

ภาษาเครื่องกล. ข้อมูลที่หน่วยประมวลผลเข้าใจและแปลความหมายเพื่อทำงานได้.

macro

แมโคร. ชื่อที่กำหนดไว้และกระจายต่อไปคำอื่น ๆต่อไป. แมโครใช้กันอย่างกว้างขวางเช่นในภาษาซี, LaTeX ฯลฯ.

magic number

ข้อมูลหรือค่าที่อยู่ในไฟล์, เป็นเอกลักษณ์พิเศษที่บอกประเภทของไฟล์ได้. ตัวอย่างเช่น magic number ของสคริปต์ที่ใช้ในยูนิกซ์หรือลินุกซ์คืออักขระ #

.

Minix

ระบบปฏิบัติการคล้ายเหมือนยูนิกซ์ที่สร้างขึ้นโดยศาสตราจารย์ Andrew Tanenbaum เพื่อการเรียนการสอนเกี่ยวกับระบบปฏิบัติการ

mount

การนำดีไวซ์หน่วยความจำข้อมูลเช่นฮาร์ดดิสก์มาปะติดในระบบโครงสร้างไดเรกทอรี. เช่นการนำซีดีรอมมาปะติด (mount) ไว้กับไดเรกทอรี `/mnt/cdrom` เป็นต้น. การ mount นี้สามารถระบุไฟล์ซิสเต็มของดีไวซ์ได้ด้วยเมื่อ mount ไฟล์ซิสเต็มเข้าสู่โครงสร้างไดเรกทอรีแล้วก็จะใช้งานได้อย่างโปร่งใส, อาจจะไม่สังเกตุเห็นความแตกต่างระหว่างไฟล์ซิสเต็ม.

multi-tasks

มัลติทาส์ก. ความสามารถในการแบ่งเวลาการทำงานของหน่วยประมวลผลเมื่อมีงานหลายอย่างที่ต้องทำ. เป็นผลให้ดูเหมือนว่าหน่วยประมวลผลข้อมูลสามารถทำงานหลายอย่างได้ในเวลาเดียวกัน.

multi-users

มัลติยูสเซอร์. ความสามารถของระบบปฏิบัติการที่ให้ผู้ใช้หลายคนทำงานได้ในเวลาเดียวกัน

— O —

OpenGL

ย่อมาจาก Open Graphics Library เป็นไลบรารี (ซอฟต์แวร์) อินเทอร์เฟสสำหรับฮาร์ดแวร์กราฟิก, ใช้ในการสร้างภาพสามมิติคุณภาพสูงและเป็นมาตรฐานสากล.

operator

ตัวดำเนิการ หรือ *ตัวปฏิบัติการ.* คือเครื่องหมายที่มีความหมายพิเศษใช้สำหรับการประมวลผล. เช่น + เป็นตัวปฏิบัติการทางคณิตศาสตร์สำหรับรวมค่าของจำนวนสองจำนวน. && เป็นตัวปฏิบัติการทางตรรกะ AND.

— P —

page

หน่วยของหน่วยความจำ. ในระบบยูนิกซ์รวมถึงลินุกซ์จะแบ่งหน่วยความจำเป็นกลุ่มขนาดเล็กเรียกว่า page. โดยทั่วไป page จะมีขนาด 4Kb (4096 ไบต์).

pager

เพจเจอร์. โปรแกรมที่ใช้แสดงข้อมูลเท็กซ์ทางหน้าจอเทอร์มินอลเป็นหน้า ๆ. มีความสามารถเลื่อนข้อมูลตามต้องการและมีความสามารถค้นหาคำที่ต้องการได้. เพจเจอร์ที่นิยมใช้ได้แก่ `less`, `more`, `lv` ฯลฯ.

partition

พาร์ทิชัน. พื้นที่ในฮาร์ดดิสก์ที่แบ่งเป็นส่วน ๆ. เมื่อแบ่งพาร์ทิชันแล้วยังไม่สามารถใช้งานได้ทันทีต้องสร้างระบบไฟล์ (file system) ก่อน.

patch

แพทช์. ไฟล์ที่มีเนื้อหาความแตกต่างระหว่างไฟล์ต้นฉบับก่อนเปลี่ยนแปลงกับไฟล์ที่แก้ไขแล้ว. ผู้พัฒนาซอฟต์แวร์จะสร้างไฟล์ patch ด้วยโปรแกรม `diff` และผู้ที่ต้องการใช้ไฟล์ patch นี้จะทำการ patch ไฟล์ดังกล่าวให้แก้ไขไฟล์ต้นฉบับที่ดั้งเดิมให้มีเนื้อหาเป็นเนื้อหาใหม่ที่แก้ไขแล้วด้วยโปรแกรม `patch`.

physical memory

หน่วยความจำจริง. คือจำนวนหน่วยความจำที่มีจริงในระบบหมายถึงตัวฮาร์ดแวร์ (memory). เคอร์เนลจะใช้หน่วยความจำจริงโดยแบ่งเป็น page, มักจะเรียกว่า freame.

pixel

หมายถึงจุดที่ประกอบกันเป็นรูปหรือหน่วยที่บอกขนาดของรูป. ขนาดของจุดไม่มีขนาดสัมบูรณ์ขึ้นอยู่กับอุปกรณ์ที่ใช้แสดง. เพราะฉะนั้นถ้าพูดถึงรูปขนาด 16x16 พิกเซลจะมีขนาดต่างกันเมื่อพิมพ์ออกทางเครื่องพิมพ์เทียบกับหน้าจอ.

port

พอร์ต(กริยา). การดัดแปลงต้นฉบับโปรแกรม, คอมไพล์ให้ใช้บนระบบปฏิบัติการที่แตกต่างกันหรือสถาปัตยกรรมคอมพิวเตอร์ที่แตกต่างกันได้

POSIX

ข้อกำหนดมาตรฐานว่าระบบปฏิบัติการที่สามารถใช้งานได้กับฮาร์ดแวร์ต่างระบบนั้นต้องมีรายละเอียดคุณสมบัติอย่างไร. มาตรฐานนี้กำหนดโดยองค์กร The Institute of Electrical and Electronic Engineers (IEEE).

PostScript

ภาษาคอมพิวเตอร์สำหรับงานกราฟิก, นิยมใช้ในงานพิมพ์ต่าง ๆ. ในลินุกซ์จะมีตัวแปลภาษา PostScript ที่เรียกว่า Ghostscript ซึ่งใช้แปลภาษา PostScript แสดงผลในดีไวซ์ต่าง ๆได้.

process

โปรเซส. สภาพที่โปรแกรมที่กำลังทำงานอยู่. กล่าวคือเนื้อหาของตัวโปรแกรมถูกโหลดจากฮาร์ดดิกส์ไปที่หน่วยความจำและหน่วยประมวลผลข้อมูลแปลคำสั่งเพื่อที่จะกระทำการต่อไป.

program

โปรแกรม. ชุดคำสั่งภาษาเครื่องกล (machine language) ที่สามารถโหลดเข้าไปในหน่วยความจำและหน่วยประมวลผลแปลความภาษาเครื่องกลเพื่อที่จะทำงานต่อไป.

prompt

พรอมต์. เครื่องหมาย, หรืออักษรที่เชลล์แสดงทางหน้าจอเพื่อแสดงว่าพร้อมที่จะรับคำสั่งจากคีย์บอร์ด.

protocol

โปรโตคอล. คือข้อตกลง, วิธีการในการกระทำการอย่างใดอย่างหนึ่ง. เช่น HTTP protol เป็นข้อตกลง, วิธี

PS/2

ย่อมาจากคำว่า Personal System 2 เป็นอินเทอร์เฟสมาตรฐานสำหรับต่ออุปกรณ์แป้นพิมพ์หรือเมาส์สำหรับคอมพิวเตอร์ส่วยบุคคล.

R

raw device

หมายถึงไฟล์ดีไวซ์. ไฟล์ที่ยังไม่ได้ mount เข้าโครงสร้างไฟล์ไดเรกทอรี. ตัวอย่าง raw device ได้แก่ `/dev/hda`, `/dev/cdrom` เป็นต้น.

regular expression

กลุ่มอักขระที่ใช้แสดงเป็นตัวแทนของคำโดยที่กลุ่มอักขระที่ใช้แสดง regular expression นี้มีไวย์กรณ์ที่แน่นอน. regular expression มีประโยชน์ในการเขียนนิยามของคำที่ต้องการค้นหา. regular expression มักจะใช้กับโปรแกรม `grep`, `sed`, `perl` เป็นต้น. ตัวอย่างเช่น "`A.*$`" หมายถึงคำที่ขึ้นต้นด้วย A จะมีอักขระใด ๆหรือไม่มี (ใช้ . แทนอักขระใด ๆ) มากกว่าหนึ่งตัวจนสุดบรรทัด. regular expression นี้แทนตัว A, A3, Asdf ได้เป็นต้น.

RGB

เป็นวิธีการแสดงสีต่างด้วยแสงโดยใช้แม่สีได้แก่สีแดง (Red), สีเขียว (Green) และสีน้ำเงิน (Blue). จะใช้ค่าแปดบิตซึ่งมีค่าตั้งแต่ 0 ถึง 255 แทนความเข้มของแม่สี. เช่นสีแดงมีค่าเป็น 255 0 0, สีเขียวมีค่าเป็น 0 255 0, และสีน้ำเงินมีค่าเป็น 0 0 255. ในทฤษฎีแสง, ถ้านำแม่สีเหล่านี้มาผสมกันจะได้สีขาว.

Serial

มาตรฐานสำหรับสื่อสาร. ด้านหลังของคอมพิวเตอร์ส่วนบุคคลโดยทั่วไปจะมีซีเรียลอินเทอร์เฟส RS-232 อยู่.

S

shared library

ไฟล์ไบนารีที่เป็นส่วนของไลบรารี. โปรแกรมที่สร้างไม่ต้องรวมส่วนที่เป็นไลบรารีเข้าไปในตัวโปรแกรม. เรียกอีกอย่างหนึ่งว่า dynamic library.

shell

เชลล์. โปรแกรมที่เป็นตัวกลางระหว่างผู้ใช้กับเคอร์เนล. มีหน้าที่แปลคำสั่งจากคีย์บอร์ดส่งต่อให้เคอร์เนลทำงานที่ต้องการ. เชลล์ที่นิยมใช้กับลินุกซ์ได้แก่ `bash`, `csh`, `ksh`, `zsh` เป็นต้น.

shell script

เชลล์สคริปต์. ไฟล์ที่รวมคำสั่งที่ใช้ในเชลล์เข้าด้วยกัน. เชลล์จะตีความคำสั่งที่อยู่ในไฟล์นั้นบรรทัดต่อบรรทัด. สะดวกสำหรับการสั่งคำที่เป็นขั้นตอน. เนื่องจากเชลล์มีไวยกรณ์, สามารถสร้างตัวแปร, สามารถควบคุมข้อแม้ต่าง ๆได้จึงถือเป็นภาษาคอมพิวเตอร์แบบ interpreter ด้วย. เชลล์สคริปต์มีบทบาทสำคัญกับลินุกซ์มากเช่น เชลล์สคริปต์ใช้ในการเริ่มต้นโปรแกรมเซิร์ฟเวอร์ต่าง ๆ.

socket

วิธีการที่ใช้เขียนโปรแกรมติดต่อกันผ่านทางเน็ตเวิร์ก. socket นี้จะเกี่ยวข้องกับ TCP, IP และ port.

stack

โครงสร้างข้อมูล (data structure) แบบเข้าก่อนออกหลัง (first-in last-out) สำหรับเก็บตัวแปรเฉพาะที่ (local variable) ในฟังก์ชัน ฯลฯ. พื้นที่สำหรับ stack จะอยู่ท้าย ๆของหน่วยความจำ.

system call

ซิสเต็มคอลล์. ฟังค์ชันภาษาซีสำหรับติดต่อใช้งานเคอร์เนล.

T

roff

ระบบประมวลผลเอกสารที่พัฒนาบนระบบปฏิบัติการยูนิกซ์ตั้งแต่เริ่มต้น. ฟอร์แมตของเอกสารนี้ในปัจจุบันเป็นฟอร์แมตของ man page ที่ใช้กันทั่วไป. ต่อมามีการพัฒนาหลายแขนงได้แก่ nroff, troff, groff เป็นต้น.

tcp

ย่อมาจากคำว่า Transfer Control Protocol เป็นโปรโตคอลมาตรฐานสำหรับควบคุมข้อมูลที่ส่งผ่านทางอินเทอร์เน็ต. มีระบบสร้างคอนเน็กชันระหว่างโฮส, ตรวจเช็กข้อมูล ฯลฯ

terminal emulator

เทอร์มินอลเอมิวเลเตอร์. โปรแกรมที่จำลองหน้าจอแสดงผลตัวอักษรเพื่อใช้กับเชลล์. เทอร์มินอลเอมิวเลเตอร์จะเป็นโปรแกรมที่ใช้ใน X วินโดว์เช่น `xterm`, `gnome-terminal`, `konsole` เป็นต้น.

terminal emulator

เทอร์มินอลเอมูเลเตอร์. โปรแกรมที่ใช้ในการแสดงผลในรูปของตัวอักษรผ่านทางหน้าจอโดยการป้อนข้อมูลเข้าผ่านทางคีย์บอร์ด

text data

ข้อมูลเท็กซ์. ข้อมูลที่หรือไฟล์ที่มนุษย์สามารถอ่านแล้วเข้าใจได้. โดยปรกติข้อมูลเท็กซ์จะหมายถึงไฟล์ที่เขียนด้วยภาษาอังกฤษที่อ่านได้, ไม่มีอักขระควบคุม (control character) ปะปน.

text mode

เท็กซ์โหมด. สภาพของระบบปฏิบัติที่อินเทอร์เฟสเป็นคีย์บอร์ดและหน้าจอเทอร์มินัลเท่านั้น.

thread

สายงานที่อยู่โปรเซสที่สามารถทำงานได้พร้อม ๆกัน. โปรแกรมที่ใช้ thread ได้จะใช้ไลบรารี pthread.

U

unicode

มาตรฐานสำหรับการแสดงข้อมูลเท็กซ์หลายภาษาพร้อม ๆกัน. เดิมทีข้อมูลในคอมพิวเตอร์ไม่ได้คำนึงถึงการใช้ภาษาอื่น ๆทำให้ช่วงแรก ๆคอมพิวเตอร์รองรับข้อมูลเท็กซ์เฉพาะภาษาอังกฤษซึ่งต้องการเนื้อที่สำหรับบันทึกอักขระแค่ 7 บิตก็เพียงพอ. ต่อมาคอมพิวเตอร์ใช้กันอย่างแพร่หลายทำให้ต้องเพิ่มเนื้อที่สำหรับภาษาอื่น ๆด้วยจึงเกิดมาตรฐานยูนิโค้ดขึ้น. อักขระทุกภาษาสามารถแสดงด้วยข้อมูล 16 บิต.

URL (Uniform Resource Locator)

วิธีการอ้างอิงแหล่งข้อมูลเช่นเอกสารทางอินเทอร์เน็ต. ตัวอย่างเช่น http://linux.thai.net:80/plone ประกอบด้วยส่วนต่าง ๆได้แก่ โปรโตคอล (http), ชื่อโดเมน (linux.thai.net), หมายเลขพอร์ต (80) และ path (plone).

USB

คำย่อของ Universal Serial Bus เป็นมาตรฐานสำหรับต่ออุปกรณ์คอมพิวเตอร์ต่าง ๆตั้งแต่แป้นพิมพ์, เมาส์, เครื่องพิมพ์, ฮาร์ดดิสก์ ฯลฯ.

— V —

virtual desktop

เดสก์ท็อปเสมือน. คุณสมบัติของวินโดว์แมนเนเจอร์ที่สามารถขยายเดสก์ท็อปให้มีหลายหน้าทำให้เนื้อที่การใช้งานเดสก์ท็อปกว้างขึ้น. แต่ในความเป็นจริงแล้วมีหน้าจอเพียงหน้าเดียว.

virtual memory

หน่วยความจำเสมือน. พื้นที่หน่วยความจำ (address space) ที่โปรเซสมองเห็นซึ่งโดยทั่วไปจะมีขนาดไม่เท่ากับหน่วยความจำจริง (physical memory). ในระบบสถาปัตยกรรม 32 บิตพื้นที่ที่โปรเซสเห็นจะมีขนาด 4GB.

virtual memory

virtual terminal

เทอร์มินอลเสมือน. เทอร์มินอลในเท็กซ์โหมดของลินุกซ์ที่สามารถเปลี่ยนหน้าจอเป็นหลายหน้าจอได้โดยมีหน้าจอที่เป็นรูปธรรมเพียงหน้าจอเดียว. วิธีเปลี่ยนหน้าจอทำได้โดยกด Ctrl และ Alt ร่วมกับฟังค์ชันคีย์.

VPN (Virtual Private Network)

การใช้เครือข่ายสาธารณะเช่นอินเทอร์เน็ตในการรับส่งข้อมูลโดยมีวิธีการรักษาความปลอดภัยของข้อมูลด้วย. พูดง่าย ๆคือการสร้างเครือข่ายที่มีความปลอดภัยสูง (เช่นการลงรหัส) บนเครือข่ายสาธารณะ.

— X —

X Display Manager

หน้าจอแบบกราฟฟิกใช้ในการล็อกอิน. ก่อนที่จะมี GNOME และ KDE, ระบบ X วินโดว์ใช้ X Display Manager ที่เรียกว่า `xdm` ซึ่งปัจจุบันไม่นิยมใช้กันแล้ว. โปรแกรมที่มาแทน `xdm` ได้แก่ `gdm`, `kdm` ฯลฯ.

XML

Extensible Markup Language. รูปแบบข้อมูลเท็กซ์ใช้คำที่กำหนดไว้เรียกว่า tag กำกับส่วนต่าง ๆให้มีความหมายต่าง ๆ. XML เป็นฟอร์แมตที่สร้างสืบทอดมาจาก SGML (Standard Generalized Markup Language) เช่นเดียวกันกับ HTML (HyperText Markup Language). HTML จะเป็นซับเซ็ตของ XML.

X server

X เซิร์ฟเวอร์. โปรแกรมแบบ server - client ที่มีหน้าที่รับคำขอจาก client วาดหน้าจอ, ผลิตหน้าต่าง, ควบคุมระบบหน้าต่าง ฯลฯ.

X window system

X วินโดว์. ระบบการแสดงผลกราฟฟิกส์ผ่านทางจอภาพในรูปแบบของหน้าต่างหลายบาน. เป็นโครงการของ MIT ที่พัฒนาต่อมาจาก W windows system ของ Stanford. ระบบ X วินโดว์ที่ใช้ในลินุกซ์เป็นโครงการของ Xfree86 ซึ่งพัฒนา X เซิฟร์เวอร์สำหรับวีดีโอการ์ดต่างๆ. สังเกตุว่าเขียนว่า X window ไม่ใช่ X windows.