Dasar-Dasar Slackware Linux

Untuk Slackware Linux 12.0

Daniël de Kok

Dasar-Dasar Slackware Linux: Untuk Slackware Linux 12.0

oleh Daniël de Kok

Tanggal Publikasi Wed Aug 19 05:43:04 WIB 2009

Lisensi

Daftar Isi

Daftar Gambar

Daftar Tabel

Pengantar

Buku ini bertujuan untuk menyediakan sebuah pengantar pada Slackware Linux. Buku ini berfokus pada orang yang tidak memiliki atau memiliki sedikit pengetahuan tentang GNU/Linux, dan membahas installasi Slackware Linux, perintah GNU/Linux dasar, dan juga konfigurasi Slackware Linux. Setelah membaca buku ini, Anda seharusnya siap untuk menggunakan Slackware Linux untuk pekerjaan sehari-hari Anda, dan lebih dari itu. Semoga buku ini juga bisa berguna sebagai sebuah referensi bagi pengguna Slackware Linux tingkat lanjut.

Terima kasih kepada pengembangan yang sangat cepat dari perangkat lunak open source, sekarang terdapat lingkungan desktop dan aplikasi yang komprehensif untuk GNU/Linux. Sebagian besar distribusi dan buku berfokus pada penggunaan GNU/Linux dengan lingkungan tersebut. Saya memilih untuk mengabaikan sebagian besar dari aplikasi grafis untuk buku ini, dan mencoba untuk memfokuskan buku ini untuk membantu Anda, sebagai seorang pembaca, untuk mempelajari menggunakan GNU/Linux pada cara yang lebih tradisional (Unix-like). Saya yakin pendekatan ini lebih handal, dan membantu Anda mempelajari GNU/Linux dengan baik, dan tidak hanya satu distribusi atau lingkungan desktop. Filosofi UNIX dijelaskan pada Gambaran filosofi UNIX [#chap-intro-unix]

Saya berharap setiap orang menikmati waktunya dengan Slackware Linux, dan saya berharap buku ini akan berguna untuk Anda.

Daniël de Kok

Bagian I. Bagaimana memulainya

Daftar Isi

Bab 1. Tentang buku ini

1.1. Ketersediaan

Buku ini ditulis dalam DocBook/XML, dan dikonversi ke HTML dan XSL:FO dengan **xsltproc**. Versi terbaru dari buku ini selalu ada pada :http://www.slackbasics.org/.

1.2. Konvensi

Bagian ini memberikan sedikit ringkasan dari konvensi yang ada pada buku ini.

Nama berkas

Nama berkas atau direktori akan dicetak sebagai: `/path/menuju/berkas`. Sebagai contoh: `/etc/fstab`

Perintah

Perintah akan dicetak sebagai teks tebal. Sebagai contoh: **ls -l**

Hasil keluaran di layar

Hasil keluaran pada layar akan dicetak sebagai berikut:

```
Hallo dunia!
```

Jika perintah dimasukkan pada layar output, maka perintah akan dicetak sebagai teks tebal:

```
$ perintah
Hasil Keluaran
```

Jika sebuah perintah dieksekusi sebagai root, shell akan ditampilkan sebagai “#”. Jika sebuah perintah dieksekusi sebagai pengguna biasa, shell akan ditampilkan sebagai “$”.

Catatan

Beberapa bagian mengandung catatan tambahan. Tidaklah penting untuk melihat informasi pada bagian catatan, tetapi catatan bisa memberikan informasi yang berharga, atau penunjuk pada sebuah informasi. Catatan akan dicetak sebagai berikut:

> **Catatan**
>
> Ini adalah sebuah catatan.

Bab 2. Perkenalan pada Slackware Linux

2.1. Apa itu Linux?

Linux adalah kernel yang bersifat UNIX-like, yang ditulis oleh Linus Torvalds dan juga para pengembang yang lain. Linux berjalan pada banyak arsitektur yang berbeda, misalnya pada mesin IA32, IA64, Alpha, m68k, SPARC dan PowerPC. Kernel terbaru dan juga informasi tentang kernel Linux dapat ditemukan pada website kernel Linux: http://www.kernel.org.

Kernel Linux seringkali disamakan dengan sistem operasi GNU/Linux. Linux hanyalah sebuah kernel, bukan sebuah sistem operasi yang lengkap. GNU/Linux terdiri dari sistem operasi GNU dengan kernel Linux. Bagian berikut ini memberikan deskripsi yang lebih lengkap tentang GNU/Linux.

2.2. Apa itu GNU/Linux?

Pada tahun 1984 Richard Stallman memulai sebuah proyek yang ambisius dengan tujuan yaitu membuat sebuah sistem operasi yang bersifat UNIX-like dan gratis. Nama dari sistem ini adalah GNU, yang merupakan akronim dari “GNU's Not UNIX”. Sekitar tahun 1990an, sebagian besar komponen utama dari sistem operasi GNU mulai ditulis, kecuali kernel. Dua tahun sebelumnya, pada tahun 1988, sudah ditentukan bahwa proyek GNU akan menggunakan mikro kernel Mach 3.0 sebagai fondasi dari kernelnya. Namun, memakan waktu hingga tahun 1991 bagi Mach 3.0 untuk dirilis dibawah lisensi perangkat lunak bebas. Pada tahun yang sama, Linus Torvalds memulai mengisi celah pada sistem GNU dengan menuliskan kernel Linux. GNU/Linux lalu mengacu pada sebuah sistem GNU yang menjalankan kernel Linux.

Kernel GNU, yang diberi nama “HURD” masih dikembangkan ketika buku ini sedang ditulis, dan sudah tersedia sebagai sistem operasi GNU/HURD. Terdapat beberapa kernel lain yang diporting pada sistem operasi GNU. Sebagai contoh, proyek Debian telah mengembangkan sebuah versi dari sistem operasi GNU yang bekerja dengan kernel NetBSD.

2.3. Apa itu Slackware Linux?

Slackware Linux adalah sebuah distribusi GNU/Linux, yang dikelola dan dikembangkan oleh Patrick Volkerding. Sebuah distribusi adalah koleksi yang koheren dari perangkat lunak yang menyediakan sistem GNU/Linux yang dapat digunakan. Volkerding mulai menggunakan GNU/Linux karena memerlukan sebuah interpreter LISP untuk sebuah proyek. Pada masa itu, distribusi GNU/Linux yang dominan adalah Softlanding System Linux (SLS Linux). Slackware Linux dimulai sebagai koleksi pribadi dari hasil perbaikan (patch) yang dilakukan oleh Volkerding untuk SLS Linux. Versi Slackware Linux pertama yang tersedia untuk publik adalah rilis 1.0, yang dirilis pada 16 Juli 1993.

Berbeda dengan kebanyakan distribusi GNU/Linux lainnya, Slackware Linux mengacu pada prinsip KISS (Keep It Simple Stupid). Hal ini berarti Slackware Linux tidak memiliki perangkat grafis yang kompleks untuk mengkonfigurasi sistem. Sebagai hasilnya, kurva pembelajaran dari Slackware Linux bisa jadi cukup tinggi untuk pengguna GNU/Linux yang belum berpengalaman, tetapi mampu menyediakan transparansi dan fleksibilitas yang lebih. Selain itu, Anda juga bisa mendapatkan pemahaman yang lebih dalam tentang GNU/Linux dengan distribusi seperti Slackware Linux.

Aspek lain yang membedakan pada Slackware Linux, yang juga “sesuai” dengan prinsip KISS, adalah pengelola paket Slackware Linux. Slackware Linux tidak memiliki manajamen paket yang kompleks seperti RPM atau dpkg. Paket adalah berkas tgz (tar/gzip) normal, seringkali dengan tambahan script installasi dan juga berkas deskripsi. Untuk pengguna awam, tgz jauh lebih handal dibandingkan RPM, dan menghindari masalah ketergantungan. Fitur lain dari Slackware Linux yang cukup dikenal adalah script inisialisasinya. Berbeda dengan kebanyakan distribusi GNU/Linux

lainnya, Slackware Linux tidak memiliki sebuah direktori untuk setiap runlevel dengan link simbolik pada layanan yang harus dijalankan atau dihentikan pada runlevel tersebut. Slackware Linux menggunakan pendekatan yang lebih sederhana, dimana Anda bisa mengaktifkan atau menonaktifkan layanan dengan memainkan bit executable dari script inisialisasi.

Paket-paket pada Slackware Linux dikompilasi dengan modifikasi sesedikit mungkin. Hal ini berarti Anda bisa menggunakan sebagian besar dokumentasi GNU/Linux pada umumnya.

2.4. Filosofi UNIX

Karena GNU/Linux adalah implementasi ulang yang bebas dari sistem operasi UNIX, disarankan untuk melihat filosofi yang membuat UNIX banyak digunakan. Doug McIlroy meringkas filosofi UNIX dalam tiga aturan sederhana:

- Menulis program yang melakukakan satu hal dan melakukannya dengan baik.
- Menulis program untuk bekerja bersama-sama.
- Menulis program untuk menangani aliran teks, karena merupakan antarmuka yang universal.

Sangatlah aneh jika Anda tidak berniat untuk menulis program untuk GNU/Linux. Namun, meskipun sebagai pengguna, aturan dasar UNIX ini sangatlah berarti bagi Anda. Setelah Anda mampu mengetahui perintah dasar yang merupakan bagian dari UNIX selama beberapa tahun, Anda akan mampu mengkombinasikan beberapa program sederhana untuk menyelesaikan masalah kompleks. Simpan hal ini dalam pikiran Anda selama Anda mempelajari Slackware Linux; cobalah untuk mendapatkan gambarang tentang bagaimana Anda bisa membagi tugas yang kompleks menjadi operasi sederhana yang dikombinasikan.

2.5. Perangkat Lunak Bebas dan open source

Sebagian besar paket-paket pada Slackware Linux dipublikasikan pada lisensi perangkat lunak bebas atau open source. Pada lisensi semacam ini, perangkat lunak dapat digunakan, dipelajari, diubah, dan didistribusikan secara bebas. Secara umum, hal ini berarti perangkat lunak tersedia dan dapat didistribusikan dalam bentuk kode sumber dan juga biner. Meskipun pergerakan perangkat lunak bebas dan open source berbagi banyak lisensi dan prinsip, terdapat sedikit perbedaan antara kedua pergerakan ini. Pergerakan open source lebih cenderung untuk berfokus pada keuntungan ekonomis dan teknis dari adanya penggunaan bersama dari kode sumber, sedangkan pergerakan perangkat lunak bebas menekankan pada sisi etika dari penyediaan kode sumber dan biner secara bebas. Seperti yang ditulis pada website GNU: “Free software is a matter of liberty, not price. To understand the concept, you should think of free as in free speech, not as in free beer.[1]” Dengan semangat software bebas dan open source, kode sumber dari hampir semua paket-paket disertakan pada CD atau DVD set resmi Slackware Linux.

2.6. Fitur Slackware Linux 12.0

- **Linux 2.6.21.5 -** Slackware Linux menggunakan kernel Linux yang modern dan memiliki performa tinggi. Kernel menyertakan dukungan untuk semua kontroller disk modern, LVM, Perangkat Lunak RAID, disk terenkripsi, dan banyak prosessor/core. Secara default, *udev* diaktifkan untuk manajemen otomatis dari node perangkat.
- **HAL** - HAL (Hardware Abstraction Layer) juga sudah disertakan. Perangkat lunak ini menyediakan API yang seragam untuk aplikasi desktop untuk menggunakan perangkat keras. Hal ini menyebabkan proses mount secara otomatis terhadap disk dan CD menjadi lebih mudah pada Xfce dan KDE.
- **X11 7.2.0 -** Ini merupakan versi pertama dari Slackware Linux yang menggunakan X secara modular. Hal ini berarti bahwa komponen X11 terpisah-pisah dalam banyak paket-paket kecil untuk perawatan yang lebih mudah serta proses upgrade yang lebih ringan.

[1] http://www.gnu.org/philosophy/free-sw.html

- **GCC 4.1.2** - Slackware Linux 12.0 menyertakan perangkat pengembangan yang sepenuhnya direvisi berbasis pada GNU Compiler Collection 4.1.2. GCC menyediakan kompiler untuk C, C++, Objective-C, Fortran-77/95, dan Ada 95. Sebagai tambahan, pustaka GNU C versi 2.5 juga sudah digunakan.
- **Apache 2.2.4** - Apache diupgrade pada versi major yang baru. Apache 2.x adalah bentuk penulisan ulang yang cukup signifikan dari seri 1.3.x.
- **K Desktop Environment (KDE) 3.5.7** - Lingkungan KDE yang penuh disertakan, termasuk KOffice, browser web Konqueror, program multimedia, perangkat pengembangan, dan masih banyak aplikasi lain yang berguna.
- **Xfce 4.4.1** - Xfce adalah lingkungan desktop yang ringan dan berbasis pada GTK2. Aplikasi ini menerapkan semangat dari UNIX yaitu modularitas dan reusabilitas.

2.7. Mendapatkan Slackware Linux

Slackware Linux dapat di-download secara bebas dari mirror Slackware Linux yang resmi. Daftar dari mirror Slackware Linux tersedia pada http://www.slackware.com/getslack/.

Anda juga bisa memesan Slackware Linux sebagai sebuah CD atau DVD dari Slackware Store. Banyak toko di Internet yang juga menawarkan Slackware Linux secara murah pada CD-ROM atau DVD, tetapi Anda akan membantu mendukung Slackware Linux secara finansial jika Anda membeli CD atau DVD set resmi Slackware Linux. Slackware Store juga menawarkan layanan Slackware Linux. Seorang pelanggan akan secara otomatis menerima rilis Slackware Linux yang baru dengan harga yang lebih murah.

Jika Anda hendak mendapatkan informasi lebih banyak tentang pembelian Slackware Linux, kunjungi situs Slackware Store pada http://store.slackware.com/.

Bab 3. Sumber-sumber bantuan

Terdapat banyak sekali informasi yang tersedia tentang banyak hal yang berhubungan dengan GNU/Linux. Sebagian besar dokumentasi berlaku pada Slackware Linux, karena perangkat lunak pada distribusi ini sudah dikompilasi dari kode sumber yang dimodifikasi sesedikit mungkin. Bab ini menyediakan beberapa petunjuk pada informasi dan dokumentasi yang dapat ditemukan sistem yang terinstall Slackware Linux, dan pada Internet.

3.1. Pada sistem Anda

Linux HOWTO

Linux HOWTO adalah koleksi dari dokumen yang membahas subyek spesifik yang berhubungan dengan GNU/Linux. Sebagian besar Linux HOWTO tidak dibuat untuk distribusi tertentu, sehingga sangat berguna bagi pengguna Slackware Linux. Paket *linux-howtos* pada set perangkat lunak “f” berisi koleksi HOWTO. Setelah menginstall paket ini, maka HOWTO akan dapat tersedia pada direktori `/usr/doc/Linux-HOWTOs/`. Slackware Linux juga berisi sekumpulan koleksi FAQs yang berhubungan dengan Linux (FAQ adalah dokumen yang menjawab Frequently Asked Questions - Pertanyaan yang Sering Diajukan). Linux FAQ terinstall pada direktori`/usr/doc/Linux-FAQs/`, dan tersedia dari paket *linux-faqs*yang juga bagian dari set “f”.

Halaman manual

Sebagian besar perintah UNIX-like dibahas pada sistem bantuan tradisional UNIX yang disebut *halaman manual*. Anda bisa membaca halaman manual sebuah program dengan perintah **man**. Mengeksekusi **man** dengan nama sebuah perintah sebagai argumen akan menampilkan halaman manual untuk perintah tersebut. Sebagai contoh,

```
$ man ls
```

menampilkan halaman manual dari perintah **ls**.

Jika Anda tidak tahu nama pasti dari sebuah halaman manual atau perintah, Anda bisa mencari halaman manual dengan kata kunci. Opsi *-k* disediakan untuk menggunakan fasilitas ini:

```
$ man -k rmdir
hrmdir               (1)  - remove an empty HFS directory
rmdir                (1)  - remove empty directories
rmdir                (2)  - delete a directory
```

Koleksi halaman manual sangatlah banyak, dan membahas lebih banyak subyek dibandingkan perintah. Berikut ini bagian dari halaman manual:

1 Program yang dapat dieksekusi atau perintah shell
2 System call (fungsi yang disediakan oleh kernel)
3 Library call (fungsi didalam pustaka program)
4 Berkas khusus (biasanya ditemukan pada /dev)
5 Format berkas dan konvensi, misalnya /etc/passwd
6 Permainan
7 Lain-lain (termasuk paket makro dan konvensi), misalnya man(7), groff(7)

8 Perintah administrasi sistem (biasanya untuk root)
9 Rutin Linux [tidak baku]

Jika terdapat lebih dari satu bagian yang memiliki halaman manual dengan nama tertentu, seperti contoh **rmdir**, Anda bisa memilih halaman yang Anda inginkan dengan menambahkan nomor bagian dari halaman manual sebelum nama halaman manual. Sebagai contoh:

```
man 2 rmdir
```

Jika Anda hendak mencetak halaman manual pada printer yang telah Anda setting, Anda bisa mengirimkan hasil keluaran pipe dari **man** ke perintah **lpr**. Ketika opsi `-t` dari **man** digunakan, **man** akan menghasilkan halaman manual dalam format Postscript, dan bukan ASCII. Sebagai contoh, Anda bisa menggunakan perintah berikut untuk mencetak halaman manual**cat**:

```
$ man -t cat | lpr
```

3.2. Di Internet

Terdapat banyak sekali situs dan forum yang berhubungan dengan GNU/Linux dan Slackware Linux di Internet. Tetapi banyak situs seringkali menghilang secepat kehadirannya, dan informasi pada banyak situs menjadi terfragmentasi. Sumber-sumber berikut sudah ada cukup lama dan menyediakan isi yang bagus.

Website Slackware Linux

Website Slackware Linux mungkin cukup ketinggalan, tetapi menyediakan banyak sumber yang berguna:

- Halaman berita yang mengumumkan rilis baru dan menampilkan berita penting lain yang relevan terhadap Slackware Linux.
- Gambaran dari perubahan pada distribusi disediakan pada format yang terstruktur yang disebut *ChangeLog*. ChangeLog disediakan untuk versi pengembangan (current) dan juga rilis stabil terbaru.
- Terdapat dua mailing list yang dapat Anda gunakan. Mailing list *slackware-announce* digunakan untuk mengumumkan rilis Slackware Linux baru, dan update keamanan diumumkan pada *slackware-security*.
- Sejumlah mirror dimana Anda bisa mendownload Slackware Linux. Mirror di perluas per negara. Informasi tambahan seperti protokol download yang didukung mirror, dan kecepatan mirror juga disertakan.
- Dokumentasi dari berbagai jenis, termasuk daftar pertanyaan yang sering diajukan, dan juga buku *Slackware Linux Essentials*.

Situs Slackware Linux tersedia pada : http://www.slackware.com/

LinuxQuestions

LinuxQuestions adalah forum GNU/Linux yang besar dengan banyak anggota yang bisa membantu. Yang menarik adalah sub forum Slackware Linux, dimana Anda bisa meminta bantuan untuk membantu Anda dengan masalah yang Anda hadapi dengan Slackware Linux. Forum LinuxQuestions tersedia pada : http://www.linuxquestions.org/

alt.os.linux.slackware

alt.os.linux.slackware adalah newsgroup Slackware Linux. Anda bisa membaca newsgroups dengan newsreader seperti tin atau slrn, melalui server newsgroup dari penyedia layanan Internet Anda. Pada newsgroup ini, Anda diharapkan sudah membaca semua dokumentasi yang diperlukan sebelum mengirimkan pertanyaan. Jika Anda belum melakukannya, peluang untuk mendapatkan “flamed” cukup besar.

Bab 4. Konsep umum

Bab ini memberikan perkenalan pada beberapa konsep umum UNIX dan GNU/Linux. Sangatlah penting untuk membaca bab ini secara mendalam jika Anda tidak memiliki pengalaman pada UNIX atau GNU/Linux. Banyak konsep yang dibahas pada bab ini digunakan pada buku ini dan juga pada GNU/Linux.

4.1. Multitasking

Perkenalan

Salah satu kekuatan tradisional UNIX adalah multitasking. Multitasking berarti banyak program dapat dijalankan pada saat yang sama. Anda mungkin heran kenapa hal ini penting, karena banyak orang hanya menggunakan satu aplikasi pada waktu yang sama. Multitasking adalah kebutuhan untuk sistem berbasis UNIX. Meskipun Anda belum menjalankan sembarang aplikasi, terdapat program yang berjalan di belakang. Sebagai contoh, beberapa program menyediakan layanan jaringan, sedangkan program yang lain menampilkan prompt login dan menunggu seorang pengguna login pada sebuah terminal (virtual). Program yang berjalan di belakang disebut *proses daemon*[1].

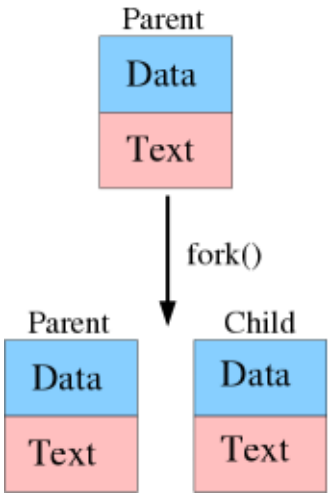
Proses dan thread

Setelah sebuah program dimuat dari media penyimpanan, sebuah instance dari program dijalankan. Instance ini disebut sebuah *proses*. Sebuah proses memiliki memorinya sendiri, disebut *ruang alamat proses (process address space)*. Ruang alaman proses memiliki dua area penting: area *teks* dan area *data*. Area *teks* adalah kode program yang sebenarnya; digunakan untuk memberitahukan sistem apa yang harus dilakukan. Area *data* digunakan untuk menyimpan data konstan dan juga runtime. Sistem operasi memberikan waktu bagi setiap proses untuk dieksekusi. Pada sistem dengan prosesor tunggal, proses tidak sepenuhnya berjalan secara bersamaan. Pada kenyataannya, sebuah penjadwal pada kernel membagi waktu CPU pada semua proses, memberikan sebuah ilusi bahwa proses-proses dijalankan secara bersamaan. Proses ini disebut dengan *time-sharing*. Pada sistem dengan lebih dari satu CPU atau inti CPU, lebih dari satu proses dapat berjalan bersamaan, tetapi konsep time-sharing masih dipakai untuk membagi waktu CPU yang ada pada setiap proses.

Proses-proses baru dibuat dengan menduplikasi proses yang sedang berjalan dengan system call **fork** system call. Gambar 4.1, “Melakukan fork dari sebuah proses” menampilkan skema dari pemanggilan fork(). Proses induk memanggil fork(). Kernel akan merespon terhadap panggilan ini dengan menduplikasi proses, menamai proses lain sebagai *induk (parent)*, dan proses lain sebagai *anak (child)*.

[1] Kata *daemon* tidak boleh disamakan dengan kata *demon*, kata *daemon* mengacu pada supernatural pada mitologi Yunani.

Gambar 4.1. Melakukan fork dari sebuah proses



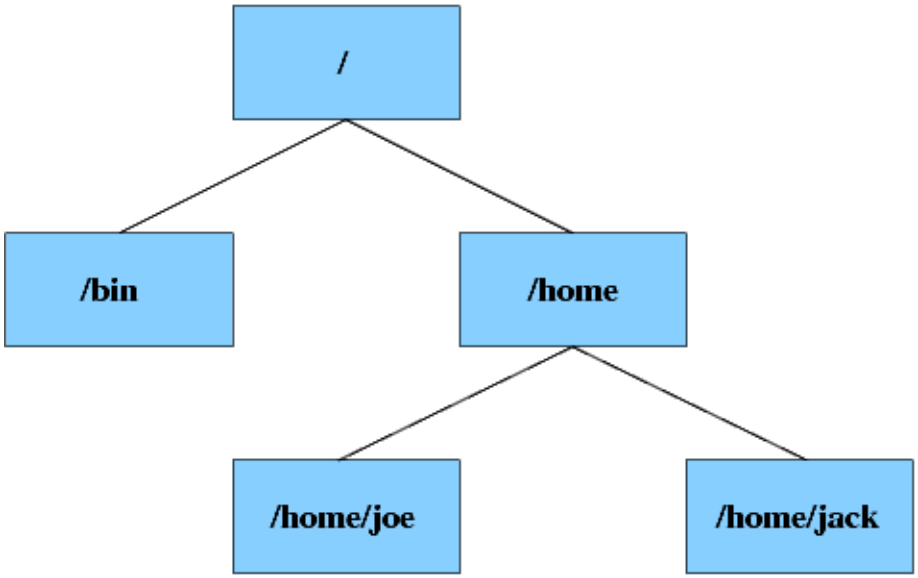
Fork bisa digunakan oleh sebuah program untuk membuat dua proses yang berjalan secara bersamaan pada mesin dengan banyak prosesor. Namun, hal ini seringkali tidak ideal, karena kedua proses akan memiliki alamat ruang prosesnya masing-masing. Duplikasi pertama dari memori proses cukup memakan waktu, dan sangat susah untuk melakukan share data diantara dua proses. Masalah ini diselesaikan oleh sebuah konsep yang disebut *multithreading*. Multithreading berarti banyak instance dari area teks dapat dijalankan pada waktu yang bersamaan, melakukan pertukaran area data. Instance ini disebut thread dapat dieksekusi secara parallel pada banyak CPU.

4.2. Hirarki Sistem Berkas

Struktur

Sistem operasi menyimpan data pada sistem berkas. Sistem berkas adalah sebuah struktur menyerupai pohon dari direktori yang menyimpan berkas, seperti sistem operasi, program pengguna, dan data pengguna. Sebagian besar sistem berkas juga dapat menyimpan berbagai macam meta data tentang berkas dan direktori, misalnya waktu akses dan modifikasi. Pada GNU/Linux hanya terdapat satu hirarki sistem berkas, yang berarti GNU/Linux tidak memiliki huruf pada drive (misalnya A:, C:, D:) untuk sistem berkas yang berbeda, seperti DOS dan Windows. Sistem berkas menyerupai sebuah pohon dengan direktori root (yang tidak memiliki direktori induk)), cabang, dan daun (direktori tanpa sub direktori). Direktori root ditandai dengan garis miring (slash) (“/”). Direktori dipisahkan dengan karakter yang sama.

Gambar 4.2. Struktur sistem berkas



Gambar 4.2, “Struktur sistem berkas” menampilkan struktur dari sistem berkas. Anda bisa melihat bahwa direktori root `/` memiliki dua direktori anak: `bin` dan `home`. Direktori `home` memiliki dua direktori anak, `joe` dan `jack`. Diagram menampilkan nama path penuh dari setiap direktori. Notasi yang sama digunakan untuk berkas. Misalkan terdapat sebuah berkas bernama `memo.txt` pada direktori `/home/jack`, path lengkap dari berkas tersebut adalah `/home/jack/memo.txt`.

Setiap direktori memiliki dua catatan khusus, “`.`”, dan “`..`”. Yang pertama mengacu pada direktori itu sendiri, sedangkan yang kedua mengacu pada direktori induk. Catatan ini bisa digunakan untuk membuat path relatif. Jika Anda bekerja pada direktori `jack`, Anda bisa mengacu direktori `/home/joe` dengan `../joe`.

Melakukan mount

Anda mungkin heran bagaimana mungkin mengakses perangkat atau partisi lain selain partisi hard disk yang menyimpan sistem berkas root. Linux mengijinkan administrator sistem menghubungkan sebuah perangkat pada sembarang direktori pada struktur sistem berkas. Proses ini disebut dengan *mount*. Sebagai contoh, kita bisa melakukan mount terhadap drive CD-ROM pada direktori `/cdrom`. Jika proses mount dilakukan dengan benar, berkas pada CD-ROM bisa diakses pada direktori ini. Proses mount dijelaskan secara detail pada Bagian 8.8, “Melakukan mount sistem berkas”.

Direktori-direktori umum

Filesystem Hierarchy Standard Group sudah mencoba untuk membuat sebuah standar yang mendeskripsikan direktori mana yang harus tersedia pada sistem GNU/Linux. Saat ini, sebagian besar distribusi menggunakan Filesystem Hierarchy Standard (FHS) sebagai panduan. Bagian ini mendeskripsikan beberapa direktori wajib pada sistem GNU/Linux

Harap diperhatikan bahwa GNU/Linux tidak harus memiliki direktori terpisah untuk setiap aplikasi (seperti Windows). Berkas diurutkan berdasarkan fungsi dan jenisnya. Sebagai contoh, program biner untuk program user yang umum disimpan pada `/usr/bin`, dan pustakanya pada `/usr/lib`. Ini adalah gambarang singkat dari direktori yang penting pada sistem Slackware Linux:

- **/bin**: program biner yang esensial yang harus tetap tersedia meskipun `/usr` tidak di-mount.

- **/dev**: berkas perangkat. Ini adalah berkas khusus yang dipakai untuk mengakses perangkat-perangkat.
- **/etc**: Direktori `/etc` berisi semua berkas konfigurasi yang penting.
- **/home**: berisi direktori home untuk setiap pengguna individual.
- **/lib**: pustaka sistem yang penting (seperti glibc) dan modul kernel.
- **/root**: direktori home untuk pengguna *root*.
- **/sbin**: program biner yang penting yang digunakan oleh administrator sistem.
- **/tmp**: Direktori yang dapat ditulis oleh siapa saja untuk berkas temporer.
- **/usr/bin**: menyimpan sebagian besar dari program biner pengguna.
- **/usr/lib**: pustaka yang tidak penting untuk sistem untuk melakukan boot.
- **/usr/sbin**: program biner yang tidak penting untuk administrasi sistem.
- **/var**: berbagai berkas data yang bervariasi, seperti log.

4.3. Perangkat

Perkenalan

Pada GNU/Linux hampir semua direpresentasikan sebagai sebuah berkas, termasuk perangkat. Setiap sistem GNU/Linux memiliki sebuah direktori dengan berkas kuhusus, yang diberi nama `/dev`. Setiap berkas pada direktori `/dev` merepresentasikan sebuah perangkat atau perangkat pseudo. Sebuah berkas perangkat memiliki dua angka khusus yang diasosiasikan dengannya, angka perangkat *major* dan *minor*. Kernel tahu perangkat mana yang direpresentasikan oleh berkas perangkat dengan melihat angka-angka ini. Contoh berikut ini menampilkan angka perangkat untuk berkas perangkat `/dev/zero`:

```
$ file /dev/zero
/dev/zero: character special (1/5)
```

Perintah **file** dapat digunakan untuk menentukan jenis dari sebuah berkas. Berkas dikenali sebagai sebuah berkas perangkat yang memiliki angka perangkat major *1* dan *5* sebagai angka minor.

Jika Anda sudah menginstall paket sumber kernel, Anda bisa menemukan daftar lengkap dari semua perangkat dengan angka major dan minornya pada `/usr/src/linux/Documentation/devices.txt`. Sebuah daftar terbaru juga tersedia secara on-line melalui Arsip Kernel Linux[2].

Kernel Linux menangani dua jenis perangkat: perangkat *karakter* dan *blok*. Perangkat karakter dapat membaca per byte, sedangkan perangkat blok tidak. Perangkat blok membaca per blok (misalnya 4096 byte per satuan waktu). Sebuah perangkat ditentukan sebagai perangkat blok atau karakter oleh sifat alami dari sebuah perangkat. Sebagai contoh, sebagian besar media penyimpanan adalah perangkat blok, dan sebagian besar perangkat masukan adalah perangkat karakter. Perangkat blok memiliki keuntungan terpisah, yaitu mereka dapat disimpan dalam cache. Ini berarti sebagian besar blok yang dibaca atau ditulis disimpan dalam sebuah area khusus dari memori sistem, yang bernama cache. Memori jauh lebih cepat dari media penyimpanan secara umum, sehingga performa bisa ditingkatkan dengan melakukan operasi pembacaan dan penulisan blok pada memori. Tentu saja, perubahan harus dituliskan pada media penyimpanan untuk merefleksikan perubahan yang dibuat pada cache.

[2] ftp://ftp.kernel.org/pub/linux/docs/device-list/

Perangkat ATA dan SCSI

Terdapat dua jenis perangkat blok yang akan kita lihat lebih detail, karena memahami penamaan dari perangkat ini sangat penting untuk mempartisi hard disk dan juga melakukan mount. Hampir semua komputer modern dengan arsitektur x86 menggunakan hard disk dan CD-ROM ATA. Pada Linux, perangkat-perangkat ini diberi nama sesuai urutannya:

```
/dev/hda - perangkat master pada kanal ATA pertama
/dev/hdb - perangkat slave pada kanal ATA pertama
/dev/hdc - perangkat master pada kanal ATA kedua
/dev/hdd - perangkat slave pada kanal ATA kedua
```

Pada sebagian besar komputer dengan hard disk tunggal, hard disk adalah perangkat master pada kanal ATA pertama (`/dev/hda`), dan CD-ROM adalah perangkat master untuk kanal ATA kedua. Partisi hard disk diberi nama sesuai disk yang menampungnya ditambah sebuah angka. Sebagai contoh `/dev/hda1` adalah partisi pertama dari disk yang direpresentasikan oleh perangkat `/dev/hda`.

Hard disk SCSI dan drive CD-ROM mengikuti konvensi penamaan lainnya. SCSI tidak umum dipakai pada mesin biasa, tetapi perangkat USB dan Serial ATA (SATA) juga direpresentasikan sebagai disk SCSI. Notasi berikut digunakan untuk perangkat SCSI:

```
/dev/sda - Disk SCSI pertama
/dev/sdb - Disk SCSI kedua
/dev/sdc - Disk SCSI ketiga
/dev/scd0 - CD-ROM pertama
/dev/scd1 - CD-ROM kedua
/dev/scd2 - CD-ROM ketiga
```

Nama partisi selalu dibuat dengan cara yang sama dengan disk ATA; `/dev/sda1` adalah partisi pertama pada disk SCSI pertama.

Jika Anda menggunakan implementasi perangkat lunak RAID pada kernel Linux, volume RAID tersedia pada `/dev/mdn`, dimana *n* adalah nomor volume yang diawali dengan 0.

Bab 5. Menginstall Slackware Linux

5.1. Boot CD-ROM Installasi

Metode termudah untuk boot sistem installasi dengan menggunakan CD-ROM installasi. CD-ROM installasi Slackware Linux adalah CD yang bersifat bootable, yang berarti BIOS dapat boot dari CD, seperti ia dapat melakukan boot dengan sendirinya, sebagai contoh dari disket. Sebagian besar sistem modern memiliki BIOS yang mendukung boot dari CD-ROM.

Jika CD tidak di-boot ketika Anda sudah memasukkan CD pada drive CD-ROM selama boot sistem, maka urutan boot mungkin tidak benar pada BIOS. Masuk pada BIOS (biasanya dengan menekan tombol <Del> atau <Esc> ketika layar BIOS muncul) dan pastikan CD-ROM berada di daftar yang paling atas dari urutan boot. Jika Anda menggunakan CD-ROM SCSI, Anda mungkin harus menentukan urutan boot pada BIOS SCSI dan bukan pada BIOS sistem. Lihat manual kartu SCSI untuk informasi lebih lanjut.

Ketika CD-ROM di-boot, sebuah layar pre-boot akan muncul. Biasanya, Anda cukup menekan <ENTER> untuk melanjutkan dengan memuat kernel Linux default (`hugesmp.s`). Kernel ini membutuhkan paling tidak CPU Pentium Pro. Anda bisa boot kernel alternatif dengan memasukkan nama kernel pada prompt, dan menekan <ENTER>. Tabel berikut menampilkan kernel yang tersedia dari CD atau DVD Slackware Linux.

Tabel 5.1. Kernel installasi

Linux	Deskripsi
huge.s	Sebelumnya, mereka ada kernel spesifik untuk pengontrol disk yang berbeda. Kernel huge yang baru menyertakan dukungan untuk semua pengontrol ATA, SATA dan SCSI yang umum. Kernel ini tidak memiliki dukungan SMP, dan bekerja pada CPU i486 dan yang lebih baru. Jika Anda memiliki Pentium Pro atau yang lebih baru, disarankan untuk menggunakan kernel *hugesmp.s*, meskipun pada sistem prosesor tunggal.
hugesmp.s	Kernel *hugesmp.s* memiliki dukungan untuk semua pengontrol ATA, SATA, dan SCSI yang umum. Sebagai tambahan, kernel ini memiliki dukungan SMP. Kernel ini direkomendasikan untuk CPU Pentium Pro dan yang lebih baru.
speakup.s	Kernel ini dapat dibandingkan dengan kernel *huge.s*, tetapi menyertakan dukungan untuk perangkat keras sintesa suara.

Setelah melakukan boot sistem installasi, Anda akan ditanya apakah Anda menggunakan layout keyboard khusus atau tidak. Jika Anda menggunakan keyboard US/Internasional umum, yang merupakan paling umum, Anda bisa menekan

<Enter> pada pertanyaan umum. Login sebagai “root”, tidak ada kata sandi yang akan ditanyakan. Setelah login, shell akan dijalankan dan Anda bisa memulai proses installasi Slackware Linux. Prosedur installasi akan dijelaskan pada bab ini.

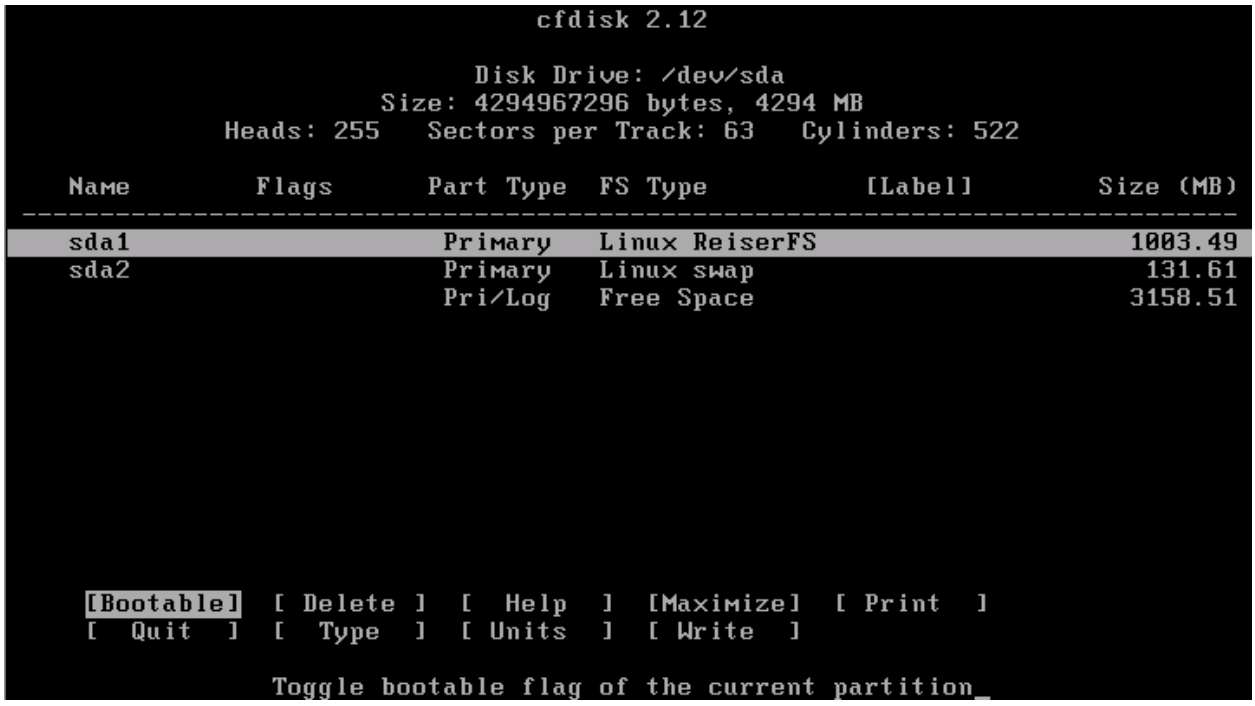
5.2. Mempartisi hard disk

Menginstall Slackware Linux membutuhkan paling tidak satu partisi Linux, membuat partisi swap juga disarankan. Untuk bisa membuat partisi, harus terdapat ruang kosong pada disk. Terdapat beberapa program yang dapat melakukan perubahan ukuran partisi. Sebagai contoh, FIPS bisa merubah ukuran partisi FAT. Program komersial seperti Partition Magic juga bisa mengubah ukuran jenis partisi lain.

Setelah boot CD-ROM Slackware Linux dan melakukan login, terdapat dua program partisi yang bisa Anda gunakan: **fdisk** dan **cfdisk**. **cfdisk** adalah yang termudah, karena dikendalikan oleh antarmuka berbasis menu. Bagian ini menjelaskan program **cfdisk**.

Untuk mempartisi hard disk pertama, Anda cukup menjalankan **cfdisk**. Jika Anda hendak mempartisi disk lain atau disk SCSI, Anda harus menentukan disk mana yang hendak Anda partisi (**cfdisk /dev/device**). Disk ATA memiliki penamaan seperti berikut: `/dev/hdn`, dengan “n” diganti oleh sebuah karakter. Contoh “primary master” diberi nama `/dev/hda`, “secondary slave” diberi nama `/dev/hdd`. Disk SCSI diberi nama dengan cara berikut : `/dev/sdn`, dengan “n” diganti oleh karakter perangkat (disk SCSI pertama = a, disk SCSI keempat = d).

Gambar 5.1. Aplikasi partisi cfdisk



Setelah memulai **cfdisk**, partisi yang ada akan ditampilkan, dan juga jumlah ruang kosong. Daftar partisi bisa dinavigasi dengan tombol panah “panah atas” dan “bawah”. Di akhir layar beberapa perintah akan ditampilkan, yang dapat dipilih dengan tombol panah “kiri” dan “kanan”. Sebuah perintah bisa dieksekusi dengan kunci <Enter>.

Anda bisa memulai mempartisi Linux dengan memilih “Free Space” dan mengeksekusi perintah “New”. **cfdisk** akan bertanya apakah Anda akan membuat partisi primer atau logical. Jumlah partisi primer dibatasi hingga empat. Linux bisa diinstall pada kedua jenis partisi. Jika Anda hendak menginstall sistem operasi lain selain Slackware Linux yang

membutuhkan partisi primer, disarankan untuk menginstall Slackware pada partisi logical. Jenis dari partisi baru akan ditetapkan sebagai “Linux Native”, sehingga tidak perlu menentukan jenisnya.

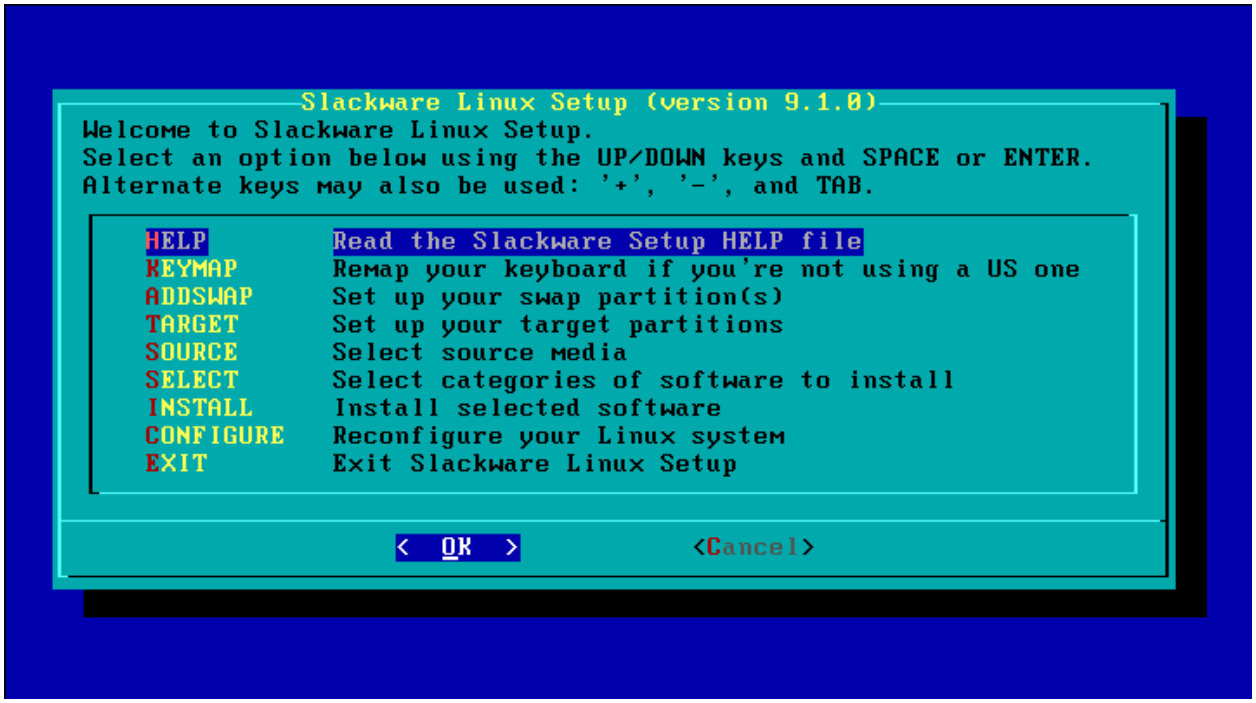
Pembuatan partisi swap melibatkan urutan yang sama dengan partisi Linux normal, tetapi jenis partisi harus diganti menjadi “Linux Swap” setelah partisi dibuat. Ukuran yang disarankan untuk partisi swap bergantung dari jumlah memori utama (RAM) Anda. Jika Anda memiliki hard disk dengan ukuran yang cukup besar, disarankan untuk membuat partisi swap berukuran 256MB atau 512MB, yang cukup untuk kebutuhan normal. Setelah membuat partisi, jenis partisi bisa diganti menjadi “Linux Swap” dengan memilih perintah “Type”. Program **cfdisk** akan bertanya tentang nomor jenis. Partisi “Linux Swap” memiliki nomor 82. Biasanya nomor 82 sudah akan terpilih, sehingga Anda bisa melanjutkan dengan menekan tombol <Enter>.

Jika Anda puas dengan partisi, Anda bisa menyimpan perubahan dengan mengeksekusi perintah “Write”. Operasi ini harus dikonfirmasikan dengan memasukkan **yes**. Setelah menyimpan perubahan, Anda bisa keluar dari **cfdisk** dengan perintah **Quit**. Disarankan untuk melakukan reboot komputer sebelum memulai installasi, untuk memastikan perubahan partisi sudah berhasil. Tekan <ctrl> + <alt> + <del> untuk mematikan Linux dan menjalankan ulang komputer.

5.3. Menginstall Slackware Linux

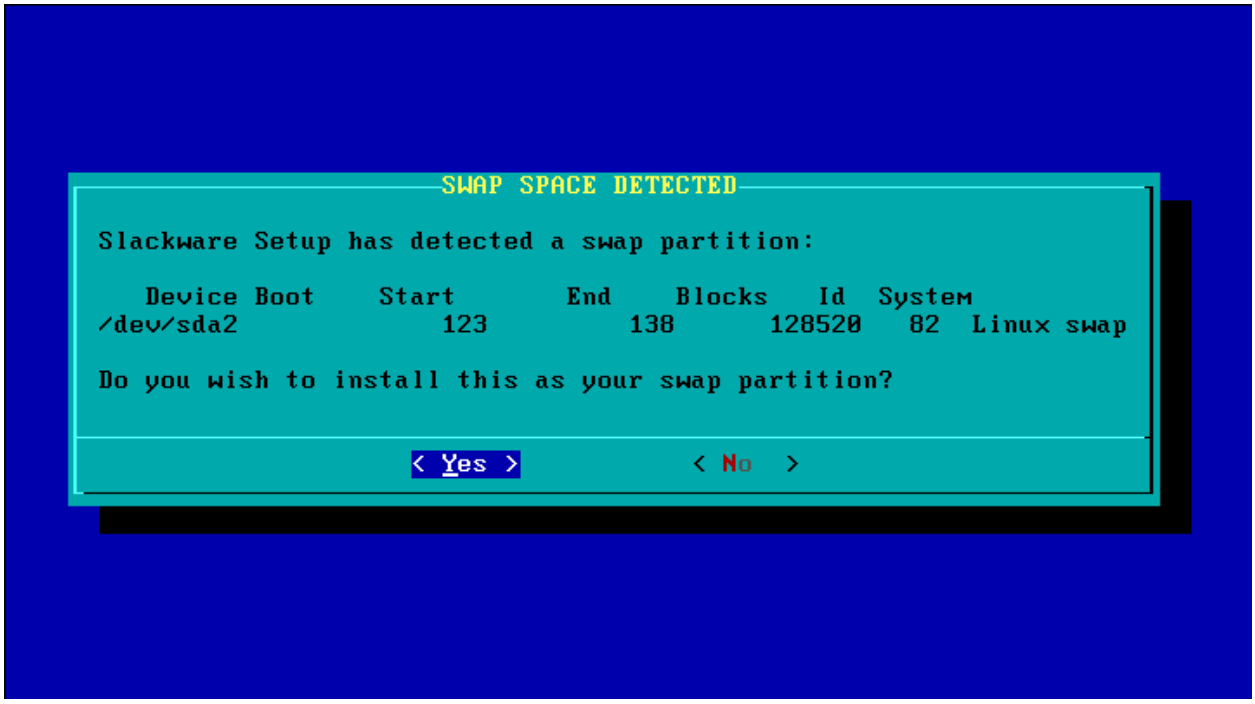
Program installer Slackware Linux dijalankan dengan mengeksekusi setup pada shell installasi. **setup** akan menampilkan sebuah menu dengan beberapa pilihan. Anda bisa melihat tampilan gambar dari installer pada Gambar 5.2, “Aplikasi setup”. Setiap opsi diperlukan untuk installasi Slackware Linux yang lengkap, tetapi setelah Anda mulai, program **setup** akan membimbing Anda pada setiap opsi.

Gambar 5.2. Aplikasi setup



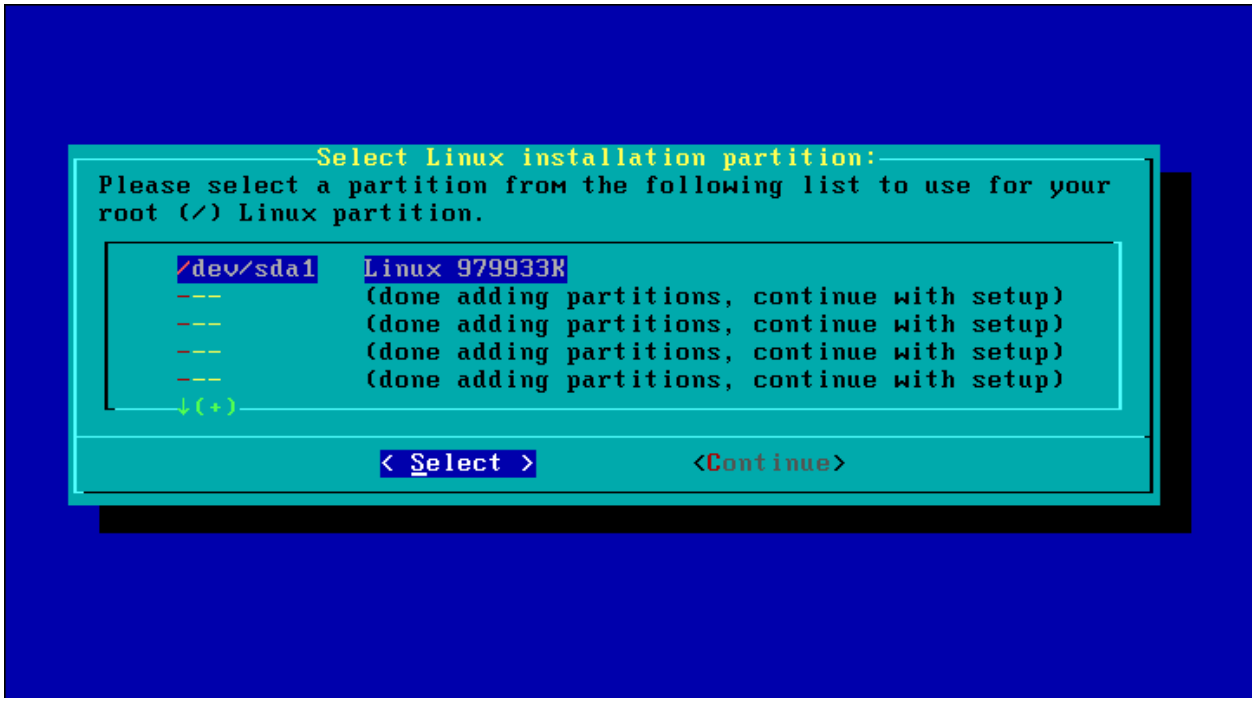
Bagian pertama dari installasi disebut “ADDSWAP”. Aplikasi **setup** akan melihat partisi dengan jenis “Linux Swap”, dan bertanya kepada Anda apakah Anda memformat dan mengaktifkan partisi swap (lihat gambar Gambar 5.3, “Melakukan setting partisi swap”). Biasanya Anda cukup menjawab “Ya”.

Gambar 5.3. Melakukan setting partisi swap



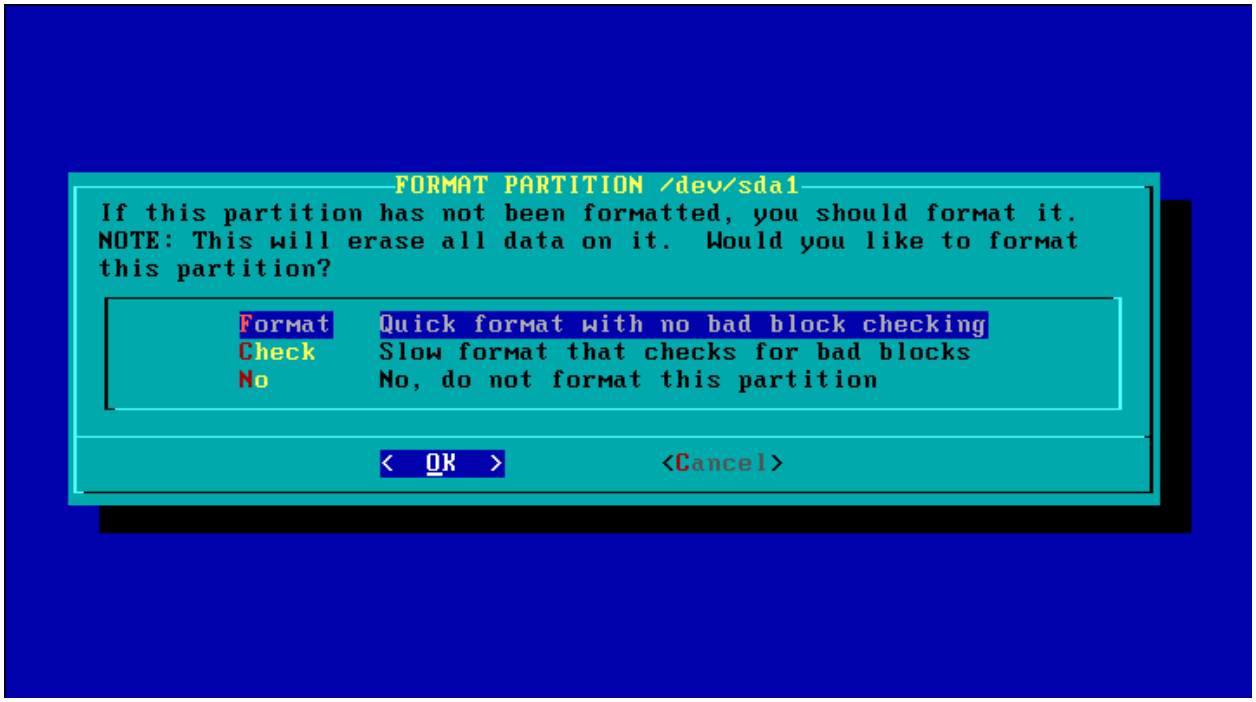
Setelah melakukan setting ukuran swap, menu “TARGET” dijalankan, yang dapat Anda lihat pada Gambar 5.4, “Memilih partisi untuk inisialisasi”. Menu ini digunakan untuk menginisialisasi partisi Slackware Linux. Setup akan menampilkan semua partisi dengan jenis “Linux Native”.

Gambar 5.4. Memilih partisi untuk inisialisasi



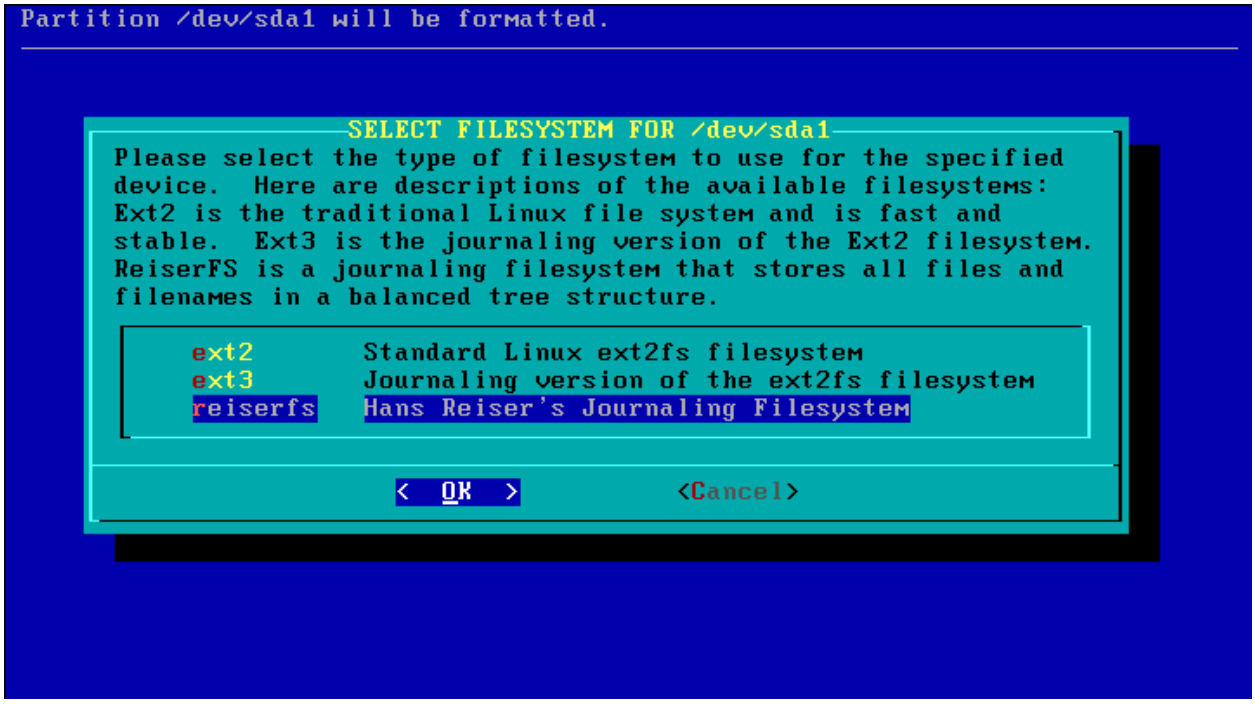
Setelah memilih satu partisi, aplikasi setup akan bertanya apakah Anda hendak memformat partisi atau tidak, dan jika Anda hendak memformatnya, apakah Anda hendak memeriksa disk untuk sektor yang rusak atau tidak (Gambar 5.5, “Memformat partisi”). Menguji disk bisa memakan waktu yang cukup lama.

Gambar 5.5. Memformat partisi



Setelah memilih untuk memformat sebuah partisi, Anda bisa menentukan sistem berkas yang akan digunakan (Gambar 5.6, “Memilih jenis sistem berkas”). Biasanya Anda bisa memilih sistem berkas ext2, ext3, dan reiserfs. Ext2 adalah sistem berkas Linux standar untuk beberapa tahun, tetapi tidak mendukung journaling. Sebuah jurnal adalah berkas khusus atau area pada partisi dimana semua operasi sistem operasi dicatat. Ketika sistem mengalami crash, sistem berkas bisa diperbaiki dengan cepat, karena kernel bisa menggunakan log untuk melihat operasi disk yang dilakukan. Ext3 merupakan sistem berkas yang sama dengan Ext2, tetapi menambahkan kemampuan journaling. Reiserfs adalah sistem berkas yang juga menyediakan journaling. Reiserfs menggunakan balanced tree, yang membuat operasi sistem berkas lebih cepat dibandingkan Ext2 atau Ext3, terutama jika bekerja dengan berkas yang kecil. Kerugiannya adalah Reiserfs lebih baru, dan mungkin lebih tidak stabil.

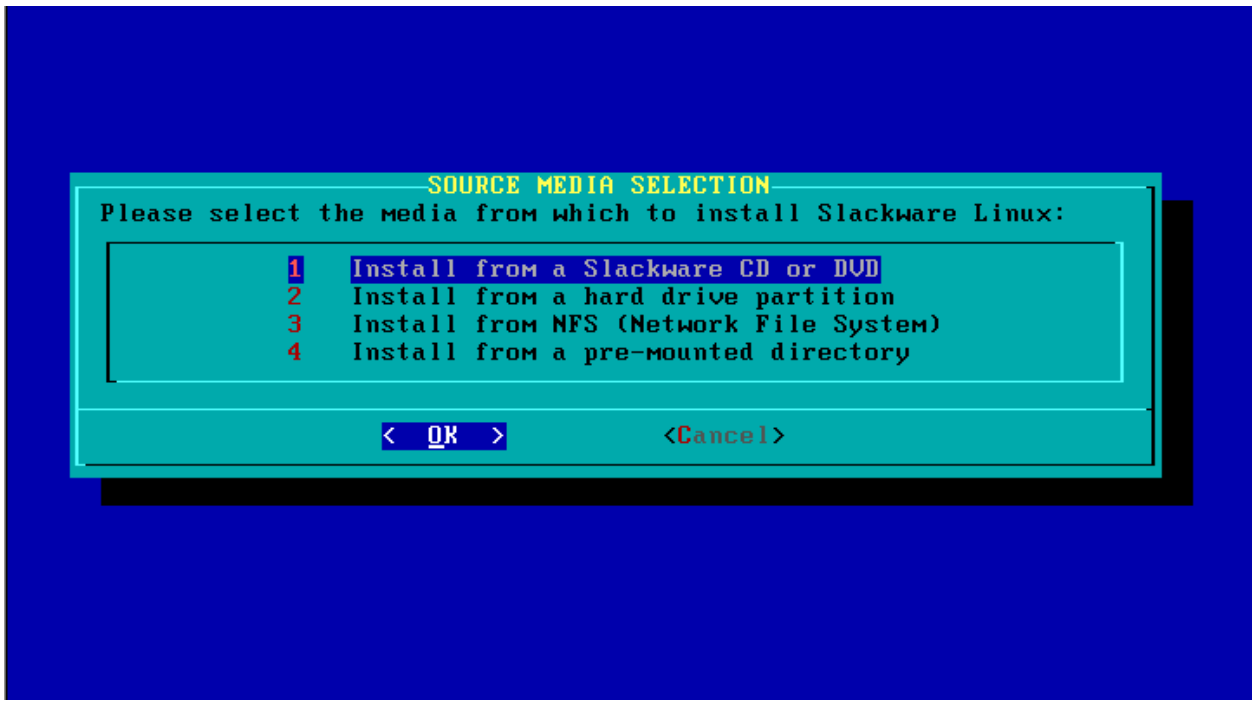
Gambar 5.6. Memilih jenis sistem berkas



Partisi pertama yang diinisialisasi secara otomatis di-mount sebagai partisi root (/). Untuk partisi lain, titik mount bisa dipilih setelah inisialisasi. Anda bisa, misalnya membuat partisi terpisah untuk `/`, `/var`, `/tmp`, `/home` dan `/usr`. Hal ini menyediakan proteksi tambahan ketika crash. Karena `/` jarang berubah setelah installasi jika Anda membuat partisi ini, kemungkinan ia akan berada pada saat operasi penulisan ketika crash menjadi lebih kecil. Selain itu, lebih aman untuk membuat sistem berkas terpisah untuk `/home`. Jika sebuah program memiliki kecacatan keamanan, seorang pengguna bisa menciptakan hard link pada biner dari sebuah program, jika direktori `/home` berada pada sistem berkas yang sama dengan `/{s}bin`, `/usr/{s}bin`, atau `/usr/local/{s}bin`. Pengguna ini akan tetap mampu mengakses biner lama setelah program di-upgrade.

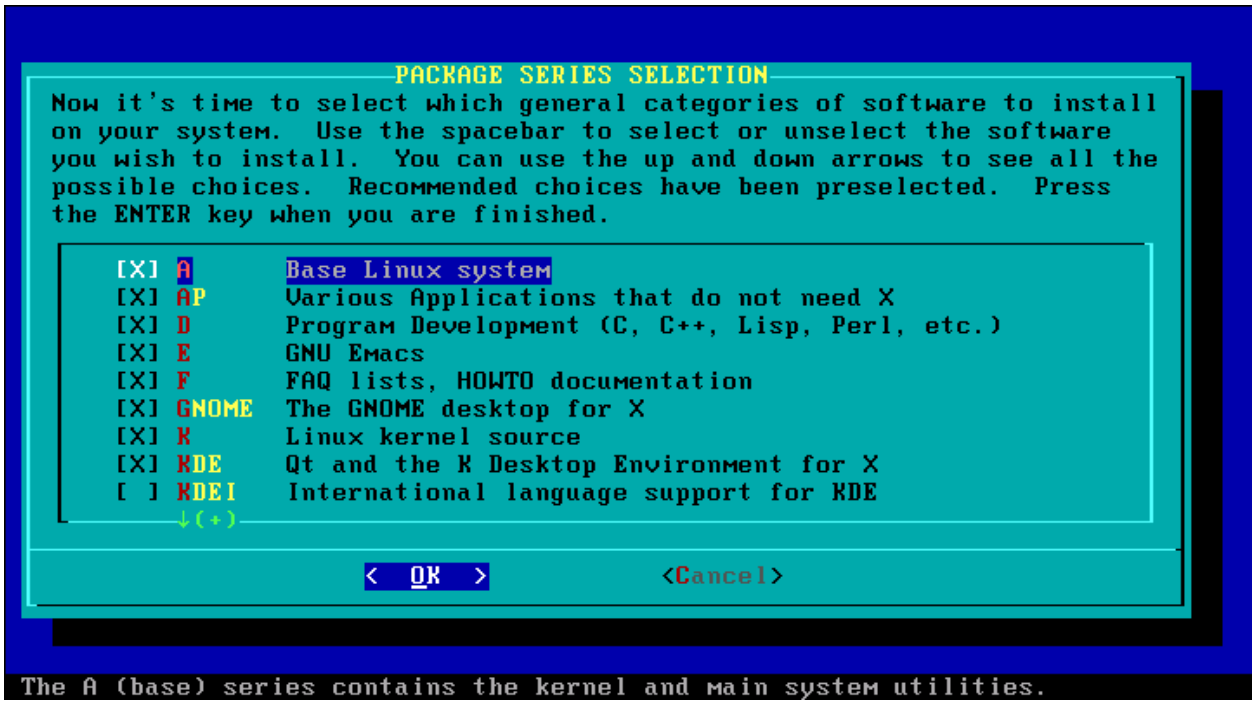
Langkah berikutnya adalah memilih media sumber (Gambar 5.7, “Memilih media sumber”). Dialog ini menawarkan beberapa pilihan, seperti menginstall Slackware Linux dari CD-ROM atau menginstall Slackware Linux via NFS. Slackware Linux biasanya terinstall dari CD-ROM, sehingga ini yang akan kita lihat lebih dalam. Setelah memilih “CD-ROM” Anda akan ditanya apakah Anda mengijinkan setup mencari CD-ROM dengan sendirinya (“Auto”) atau Anda hendak memilih perangkat CD-ROM Anda sendiri (“Manual”). Jika Anda memilih “Manual” aplikasi setup akan menampilkan daftar perangkat. Pilih perangkat yang berisi CD-ROM Slackware Linux.

Gambar 5.7. Memilih media sumber



Setelah memilih sumber installasi, aplikasi setup akan bertanya seri set disk mana yang hendak Anda install (Gambar 5.8, “Memilih set disk”). Sebuah deskripsi singkat dari masing-masing set disk akan ditampilkan.

Gambar 5.8. Memilih set disk



Sekarang hampir saatnya untuk memulai installasi yang sebenarnya. Layar berikutnya akan menanyakan bagaimana cara Anda hendak melakukan installasi. Pilihan yang paling jelas adalah “full”, “menu” atau “expert”. Memilih “full”

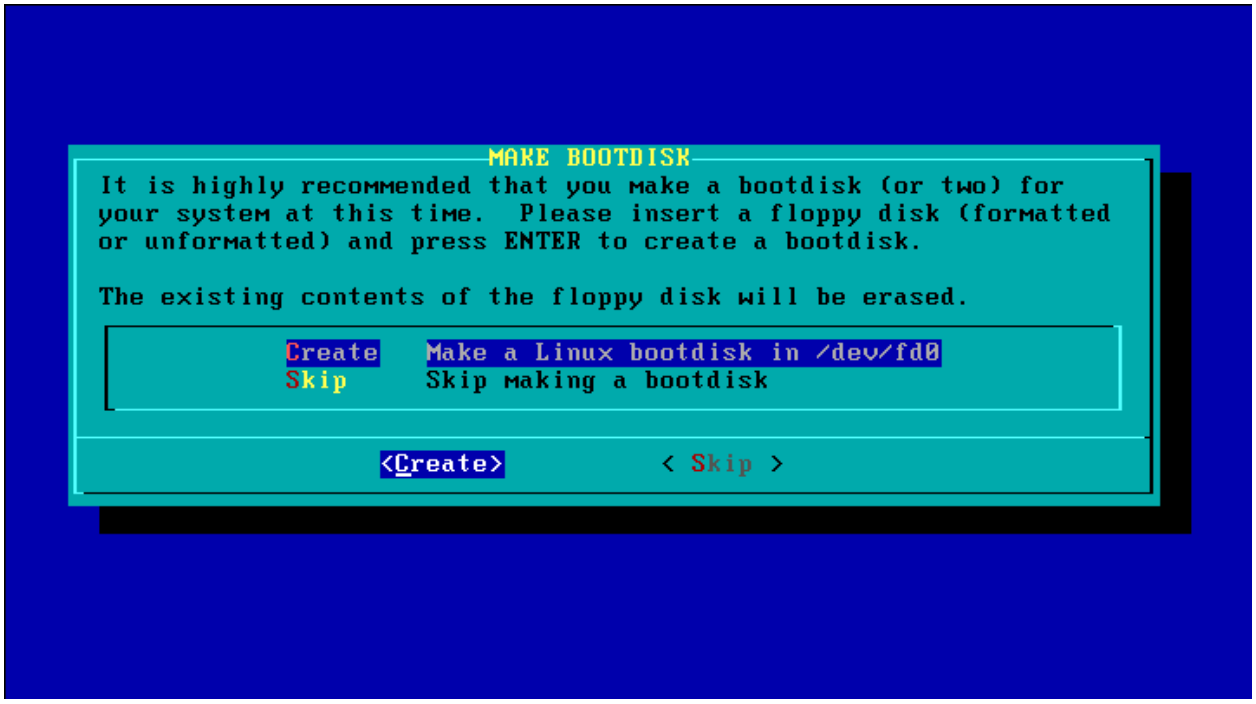
akan menginstall semua paket pada set disk yang dipilih. Ini adalah cara termudah dalam menginstall Slackware Linux. Kerugian dari pilihan ini adalah bisa memakan ruang yang cukup besar. Pilihan “menu” akan bertanya untuk setiap setnya, paket apa yang hendak Anda install. Opsi “expert” hampir sama dengan opsi “menu”, tetapi mengijinkan Anda untuk tidak memilih beberapa paket yang sangat penting dari set disk “a”.

Setelah selesai tahap installasi, aplikasi setup akan mengijinkan And auntuk mengkonfigurasi beberapa bagian dari sistem. Dialog yang muncul pertama kali akan menanyakan dari mana Anda hendak menginstall kernel (lihat Gambar 5.9, “Menginstall kernel”). Biasanya installasi kernel dilakukan dari CD-ROM Slackware Linux, pilihan ini akan memilih kernel yang Anda pilih untuk Slackware Linux. Anda bisa mengkonfirmasi ini, atau memilih kernel lain.

Gambar 5.9. Menginstall kernel

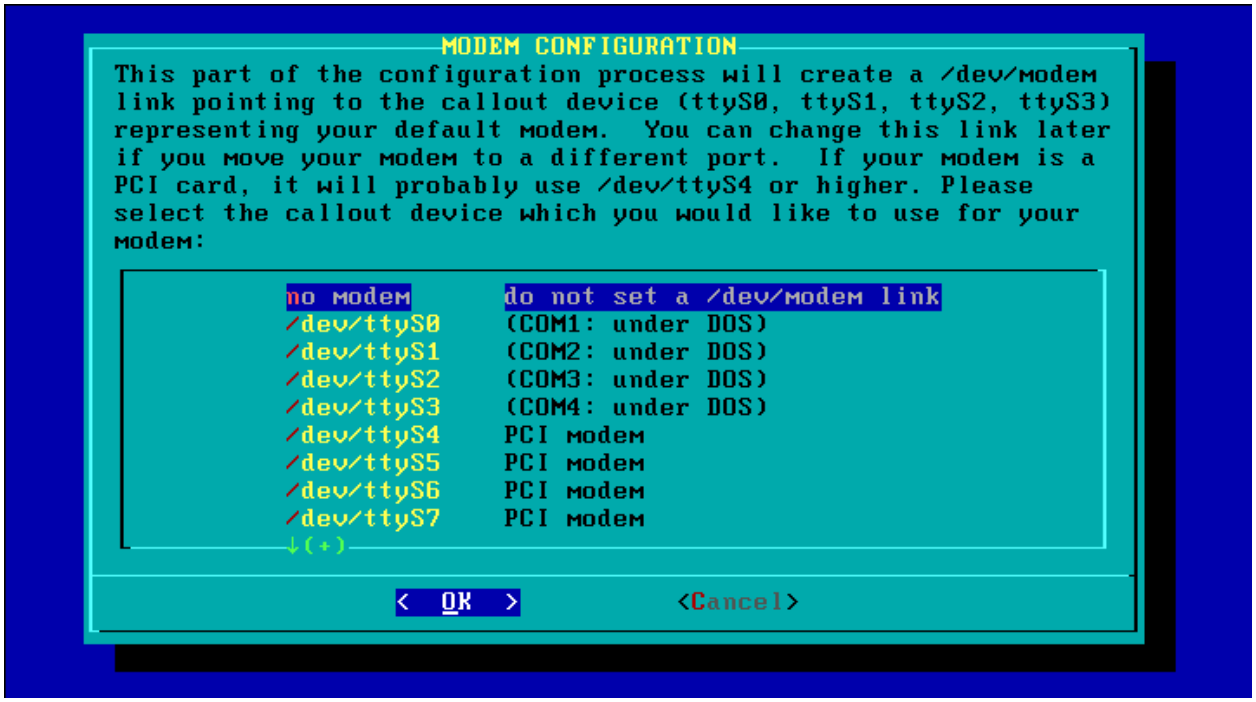
Pada tahap ini Anda bisa memilih untuk membuat disk boot (Gambar 5.10, “Membuat disk boot”). Disarankan untuk membuat disk boot, Anda bisa menggunakannya untuk melakukan boot Slackware Linux jika konfigurasi LILO salah.

Gambar 5.10. Membuat disk boot



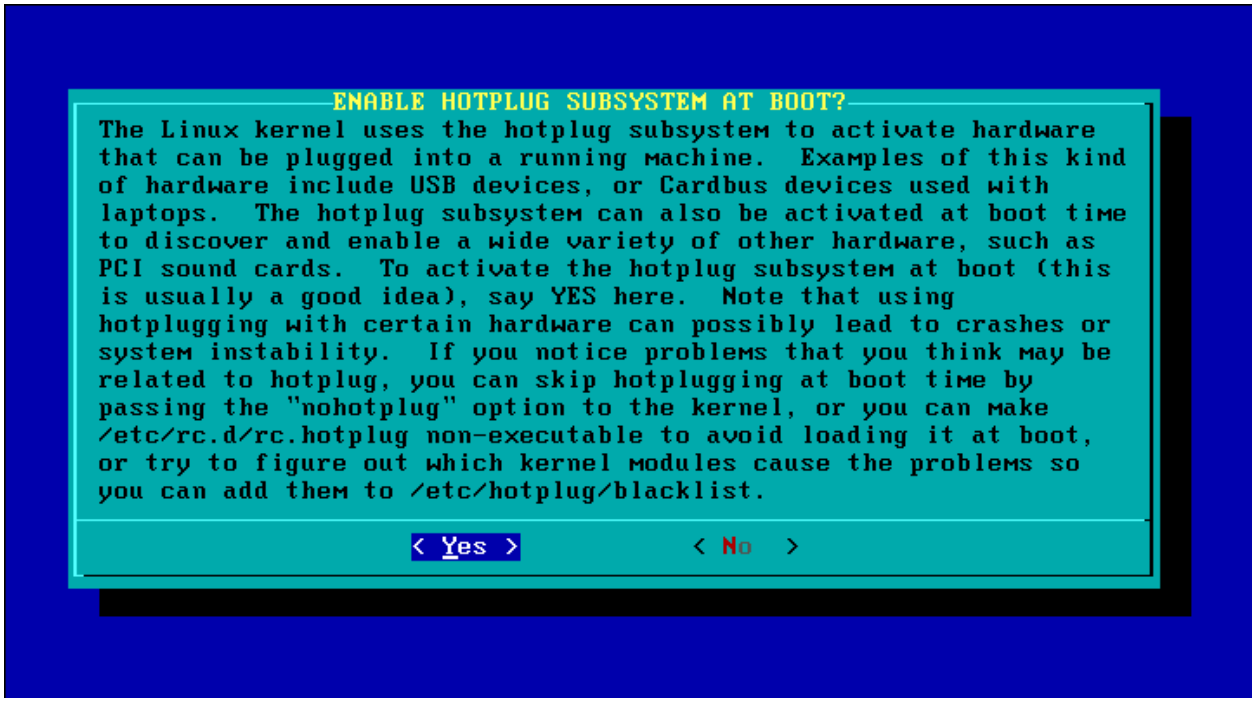
Dialog berikut dapat digunakan untuk membuat sebuah hubungan (link), `/dev/modem`, yang menunjuk pada perangkan modem Anda. (Gambar 5.11, “Memilih modem default”). Jika Anda tidak memiliki sebuah modem, Anda bisa memilih *no modem*.

Gambar 5.11. Memilih modem default



Langkah berikutnya adalah memilh apakah Anda hendak menggunakan hotplug (Gambar 5.12, “Mengaktifkan hotplugging”). Hotplug digunakan untuk mengkonfigurasi secara otomatis perangkat-perangkat USB, PCMCIA, dan PCI yang bersifat pluggable. Secara umum, disarankan untuk mengaktifkan hotplugging, tetapi beberapa sistem mungkin mengalami masalah dengan proses probing dari script hotplug.

Gambar 5.12. Mengaktifkan hotplugging

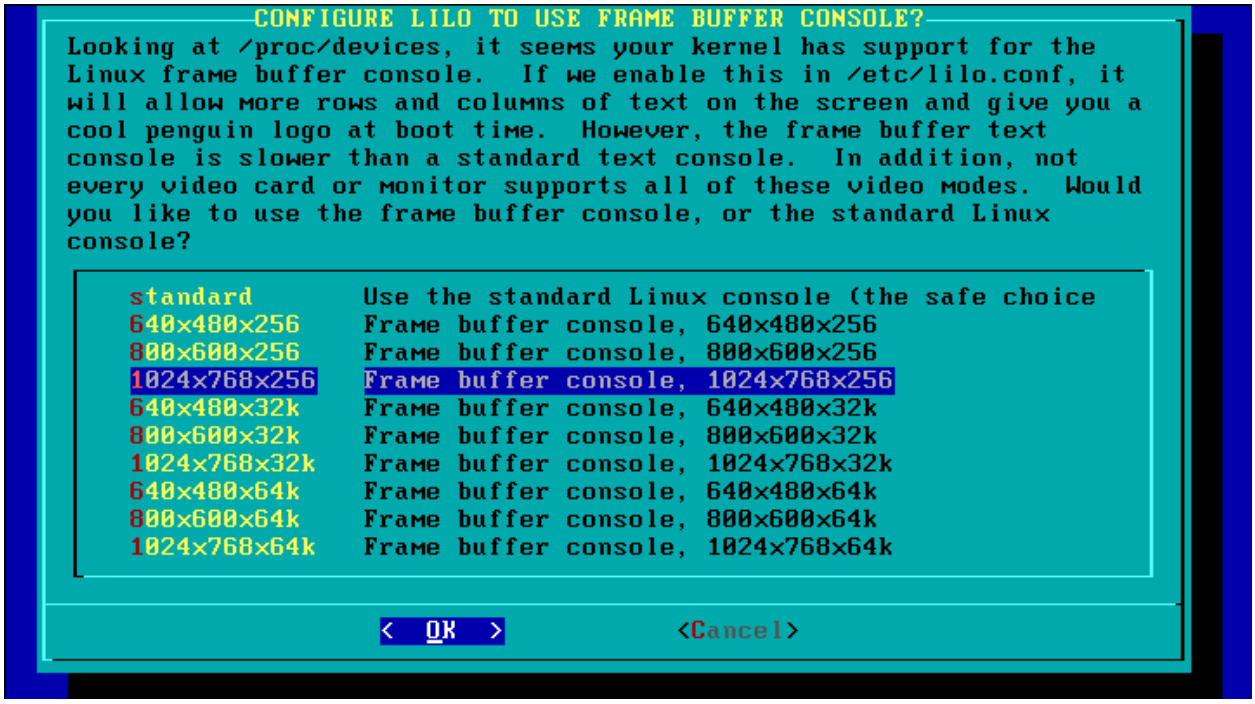


Langkah berikut sangatlah penting, dialog berikutnya akan membantu Anda dalam menginstall LILO, Linux bootloader. Kecuali Anda sudah berpengalaman dalam mengkonfigurasi LILO, disarankan untuk memilih opsi *simple* untuk konfigurasi LILO, yang mencoba mengkonfigurasi LILO secara otomatis (Gambar 5.13, “Memilih jenis installasi LILO”).

Gambar 5.13. Memilih jenis installasi LILO

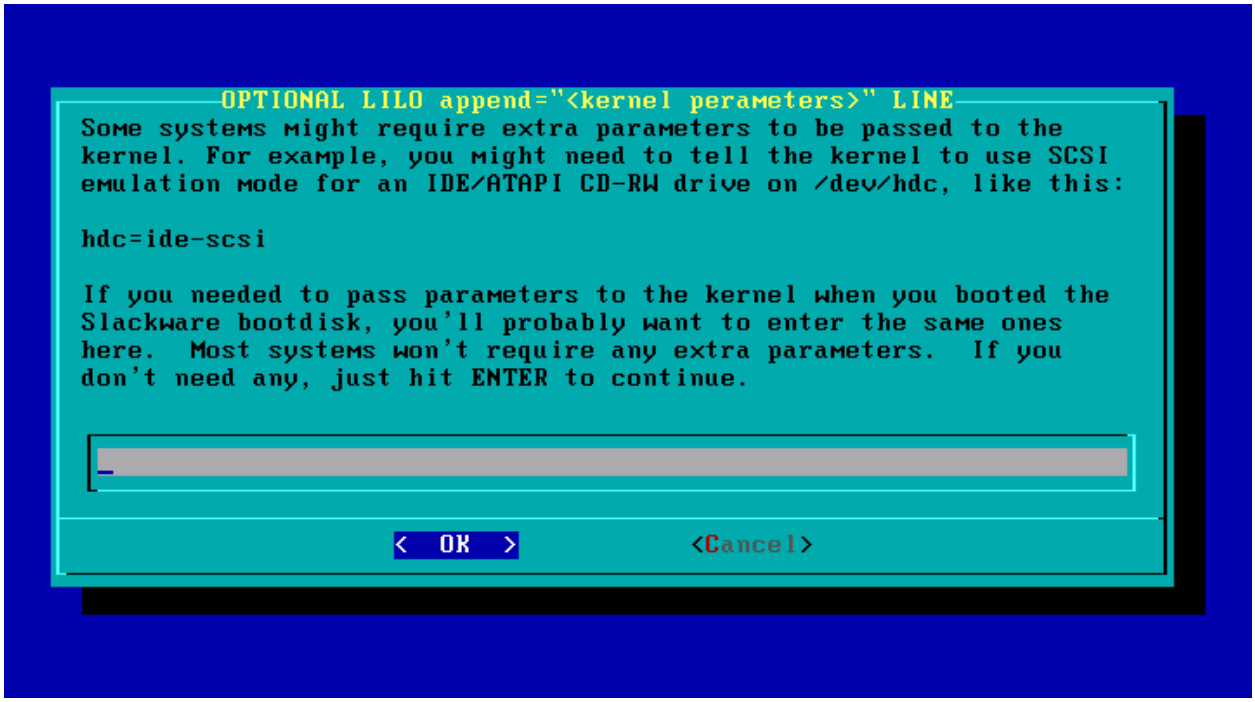
Setelah memilih opsi *simple* utilitas konfigurasi LILO akan menanyakan apakah Anda hendak menggunakan framebuffer atau tidak ((Gambar 5.14, “Memilih resolusi framebuffer”). Menggunakan framebuffer mengijinkan Anda untuk menggunakan konsol pada beberapa resolusi, dengan dimensi lain selain 80x25 karakter. Beberapa orang yang menggunakan konsol secara ekstensif memilih untuk menggunakan framebuffer, yang mengijinkan mereka untuk mempertahankan lebih banyak teks pada layar. Jika Anda tidak menginginkan konsol dengan framebuffer, atau jika Anda tidak yakin, Anda bisa memilih *standard* pada pilihan ini.

Gambar 5.14. Memilih resolusi framebuffer



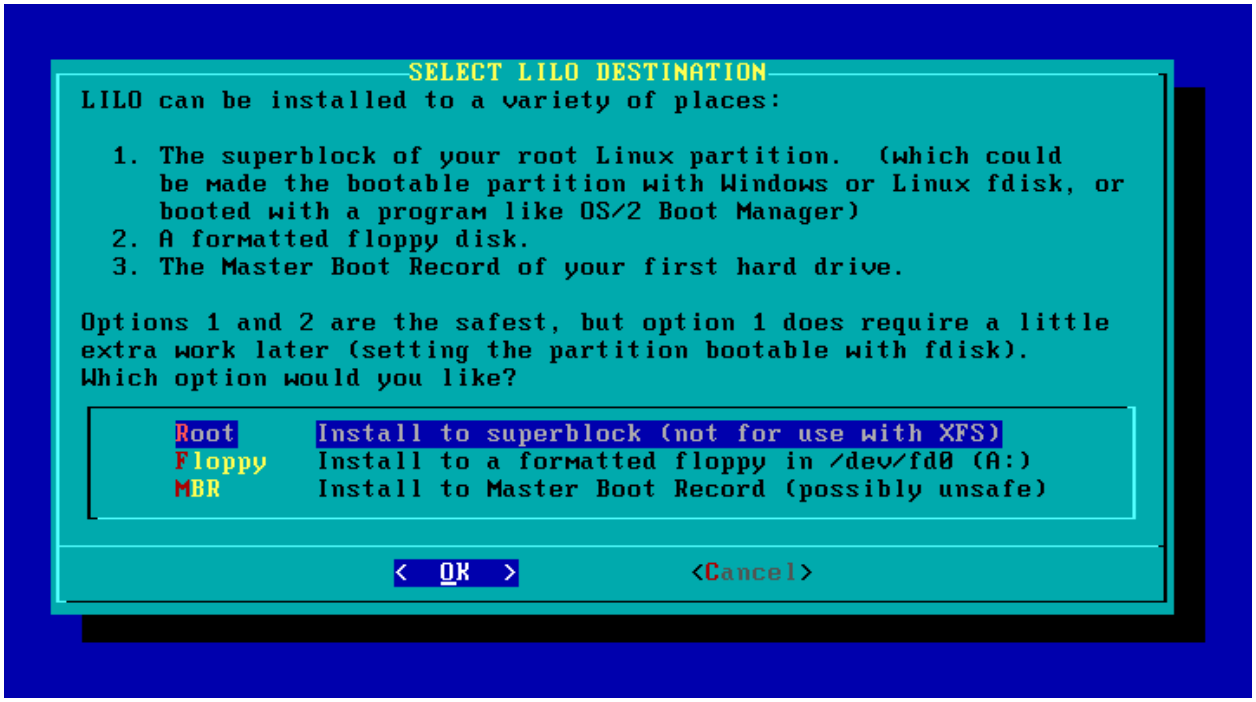
Setelah melakukan setting framebuffer, Anda bisa memberikan parameter tambahan pada kernel (Gambar 5.15, “Menambahkan parameter kernel”). Hal ini biasanya tidak diperlukan, jika Anda tidak ingin menambahkan parameter tambahan, Anda cukup menekan tombol <Enter>.

Gambar 5.15. Menambahkan parameter kernel



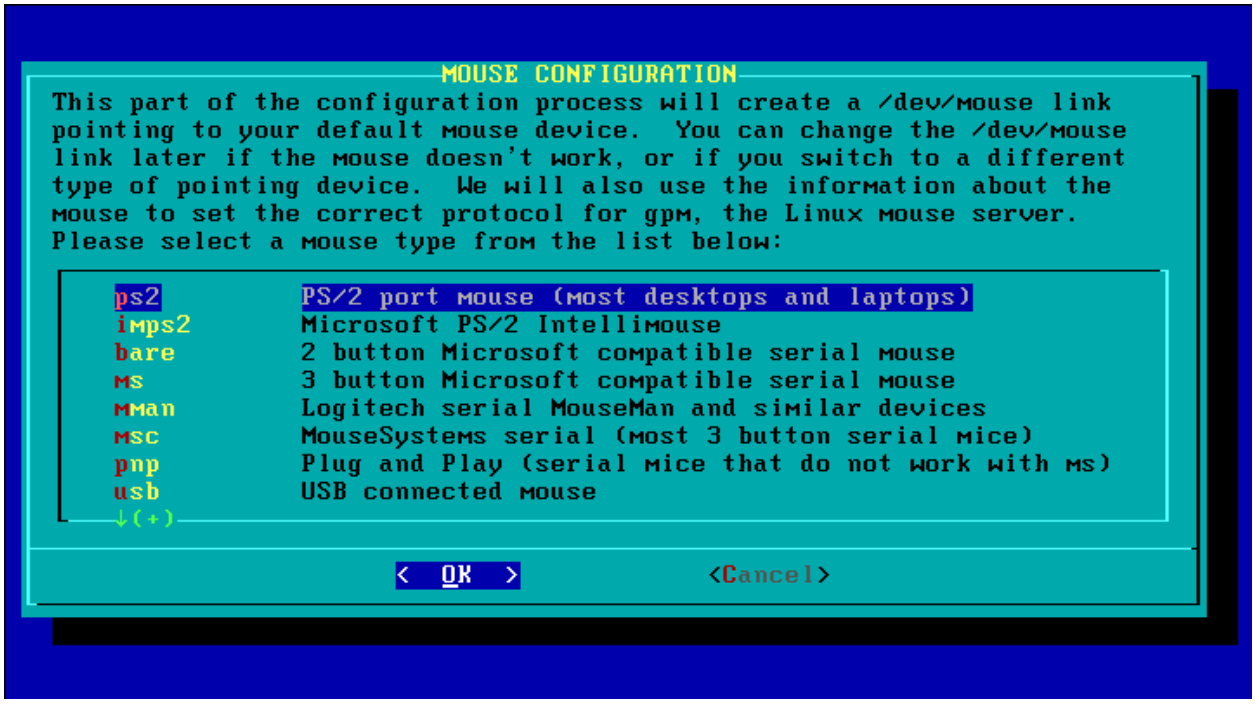
Langkah terakhir dari konfigurasi LILO adalah memilih dimana LILO akan diinstall Gambar 5.16, “Memilih dimana LILO akan diinstall”). *MBR* adalah master boot record, record boot utama dari PC. Gunakan opsi ini jika Anda hendak menggunakan Slackware Linux sebagai satu-satunya sistem operasi, atau jika Anda hendak menggunakan LILO untuk melakukan boot sistem operasi lain. Opsi *Root* akan menginstall LILO pada boot record dari Slackware Linux partisi `/`. Gunakan opsi ini jika Anda hendak menggunakan bootloader lain.

Gambar 5.16. Memilih dimana LILO akan diinstall



Sekarang Anda akan diminta untuk mengkonfigurasi mouse Anda, Pilih jenis mouse dari dialog yang muncul (Gambar 5.17, “Mengkonfigurasi mouse”).

Gambar 5.17. Mengkonfigurasi mouse



Berikutnya Anda akan ditanya apakah program **gpm** harus dimuat pada saat boot atau tidak (Gambar 5.18, “Memilih apakah GPM akan dijalankan atau tidak”). **gpm** adalah daemon yang mengijinkan Anda untuk melakukan operaasi cut dan paste teks pada konsol.

Gambar 5.18. Memilih apakah GPM akan dijalankan atau tidak

Beberapa langkah berikutnya akan mengkonfigurasi konektivitas jaringan. Langkah ini diperlukan untuk hampir semua sistem yang terhubung ke jaringan. Setup Slackware Linux akan bertanya apakah Anda hendak mengkonfigurasi konektivitas jaringan (Gambar 5.19, “Memilih apakah Anda akan mengkonfigurasi konektivitas jaringan”). Jika Anda menjawab “No” Anda bisa melewatkan beberapa langkah yang berhubungan dengan jaringan.

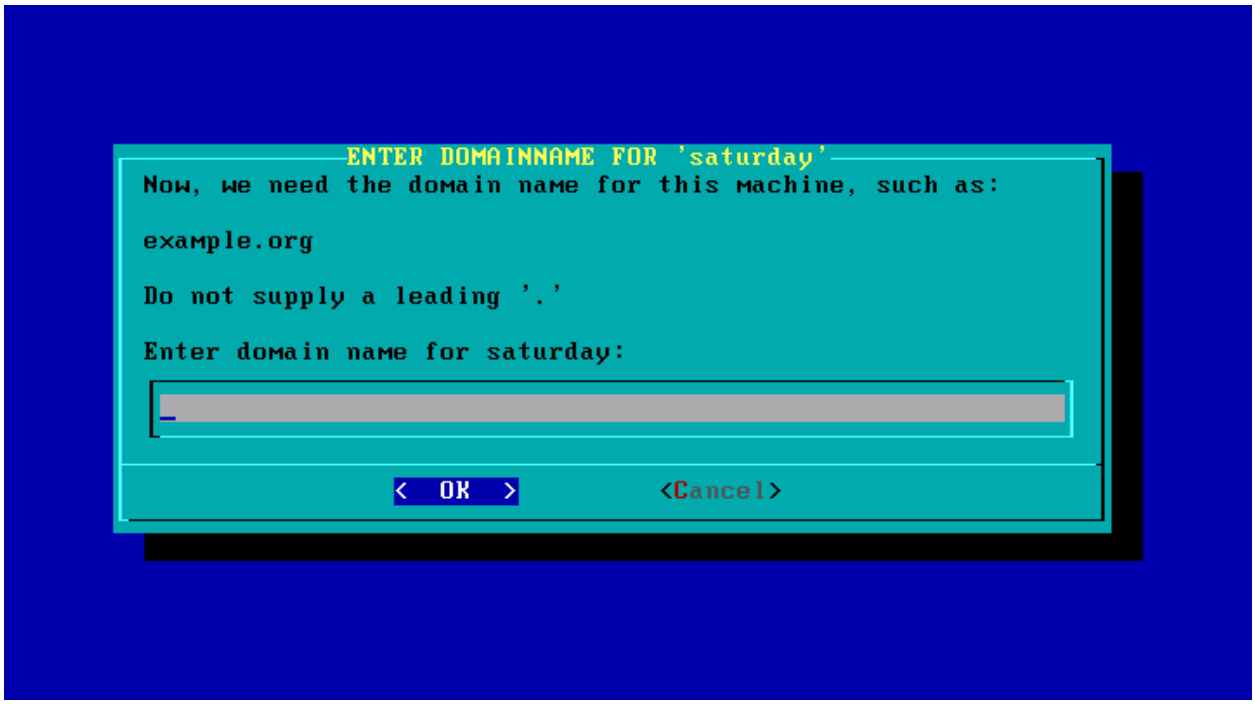
Gambar 5.19. Memilih apakah Anda akan mengkonfigurasi konektivitas jaringan

Anda akan ditanya untuk menentukan nama host (Gambar 5.20, “Melakukan setting nama host”). Haarap dicatat bahwa ini bukan fully qualified domain name (FQDN), hanya bagian yang merepresentasikan host (biasanya karakter sebelum titik pertama dalam bentuk FQDN).

Gambar 5.20. Melakukan setting nama host

Setelah menentukan nama host, Anda bisa mnentukan nama bagian domain dari fully qualified domain name (FQDN) (Gambar 5.21, “Melakukan setting nama domain”).

Gambar 5.21. Melakukan setting nama domain

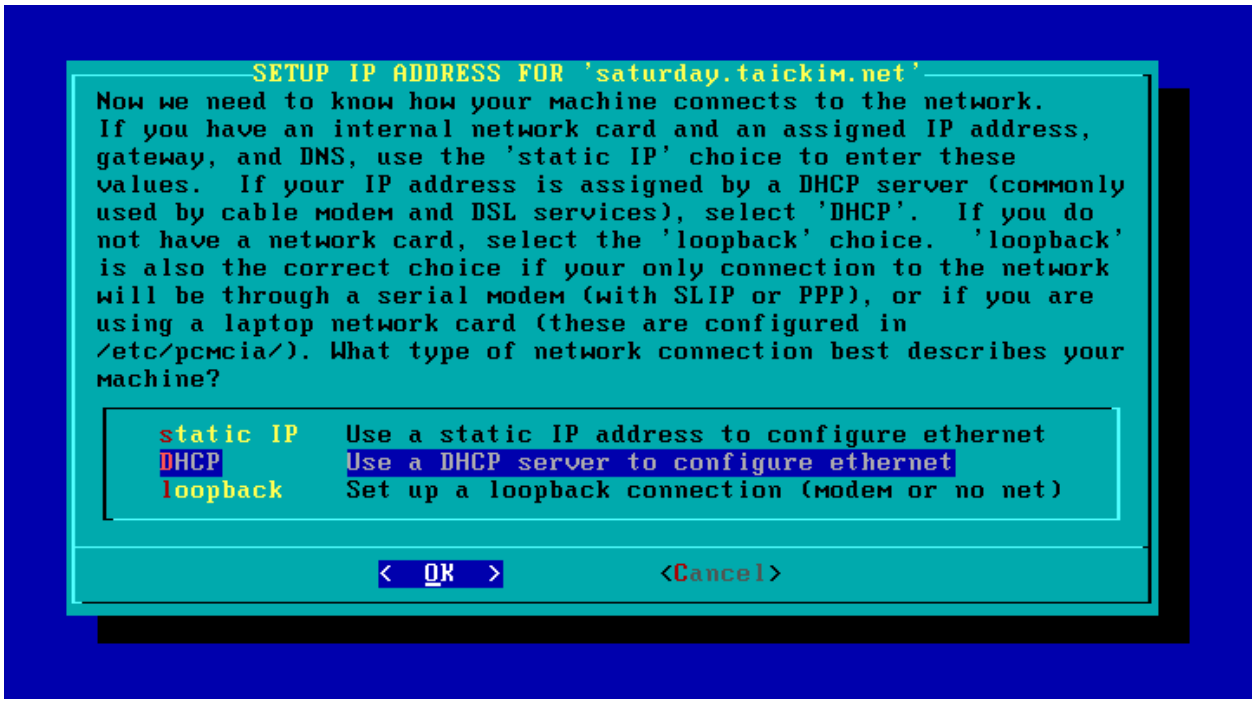


Sisa dari langkah konfigurasi jaringan tergantung dari apakah node pada konfigurasi gambar menggunakan alamat IP statik atau dinamis. Beberapa jaringan memiliki server DHCP yang secara otomatis memberikan alamat IP pada

host di jaringan. Jika ini kasusnya, maka pilih *DHCP* selama langkah installasi (Gambar 5.22, “Konfigurasi alamat IP manual atau otomatis”). Ketika DHCP dipilih, Anda akan ditanya apakah sebuah nama host harus dikirimkan ke server. Biasanya Anda bisa mengosongkan bagian ini. Jika Anda menggunakan DHCP, Anda bisa melewatkan beberapa bagian dari konfigurasi jaringan yang dijelaskan dibawah ini.

Jika jaringan tidak memiliki server DHCP, Anda bisa memilih opsi *static IP*. uamh mengijinkan Anda menentukan alamat IP dan setting yang berhubungan secara manual.

Gambar 5.22. Konfigurasi alamat IP manual atau otomatis



Langkah pertama dari konfigurasi manual adalah menentukan alamat IP dari antarmuka pertama (eth0) dari mesin (Gambar 5.23, “Melakukan setting alamat IP”).

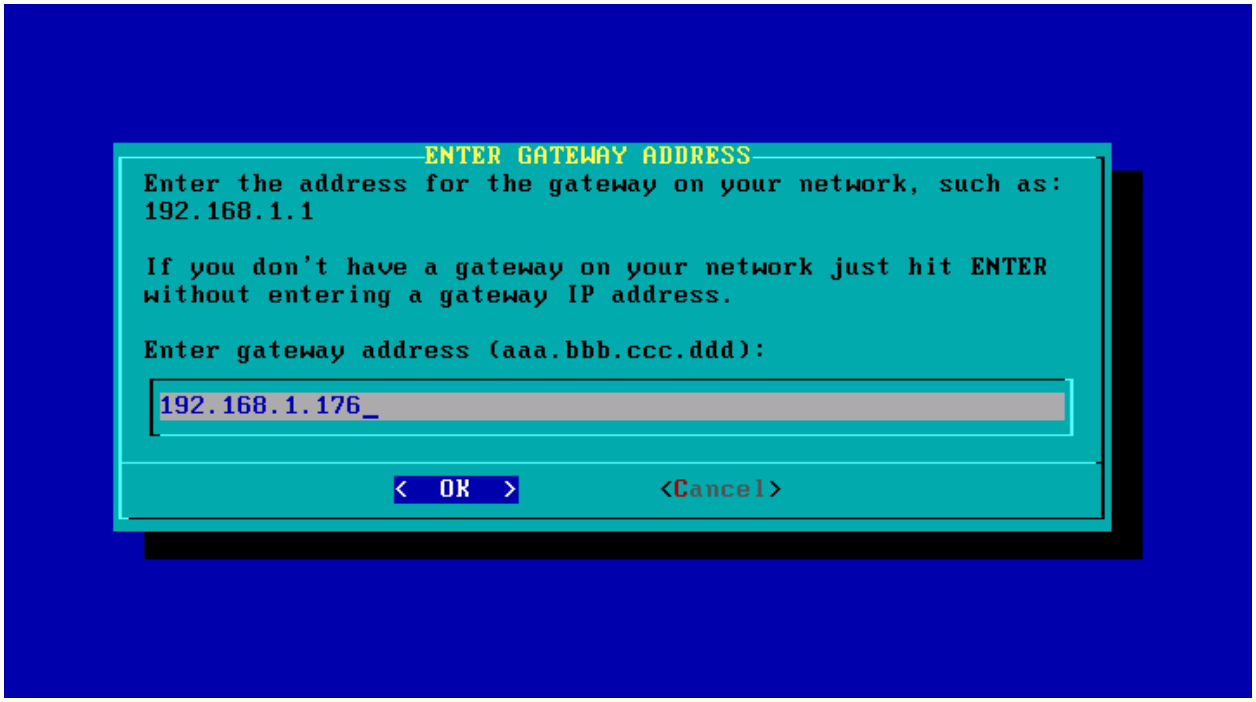
Gambar 5.23. Melakukan setting alamat IP

Setelah melakukan setting alamat IP, Anda akan ditanya untuk memasukan netmask. Netmask biasanya bergantung pada kelas alamat IP (Gambar 5.24, “Melakukan setting netmask”).

Gambar 5.24. Melakukan setting netmask

Berikutnya Anda akan ditanya untuk menentukan alamat dari gateway (Gambar 5.25, “Melakukan setting gateway”). Gateway adalah mesin pada jaringan yang menyediakan akses ke jaringan lain dengan merutekan paket IP. Jika jaringan Anda tidak memiliki gateway, Anda cukup menekan tombol <ENTER>.

Gambar 5.25. Melakukan setting gateway

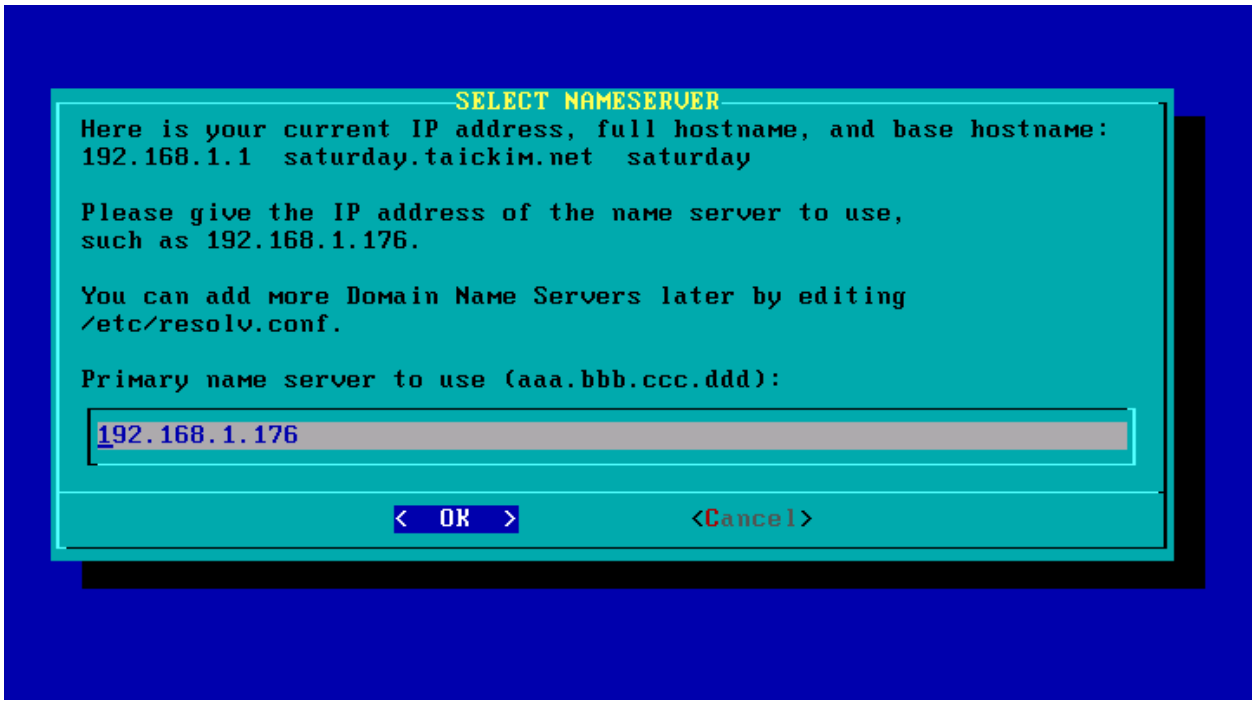


Dialog berikutnya menanyakan kepada Anda apakah Anda hendak menggunakan nameserver atau tidak (Gambar 5.26, “Memilih apakah Anda hendak menggunakan nameserver atau tidak”). Sebuah nameserver adalah sebuah server yang menyediakan alamat IP dari sebuah nama host. Sebagai contoh, jika Anda berkunjung ke *www.slackbasics.org*, nameserver akan “convert” nama *www.slackbasics.org* menjadi alamat IPnya.

Gambar 5.26. Memilih apakah Anda hendak menggunakan nameserver atau tidak

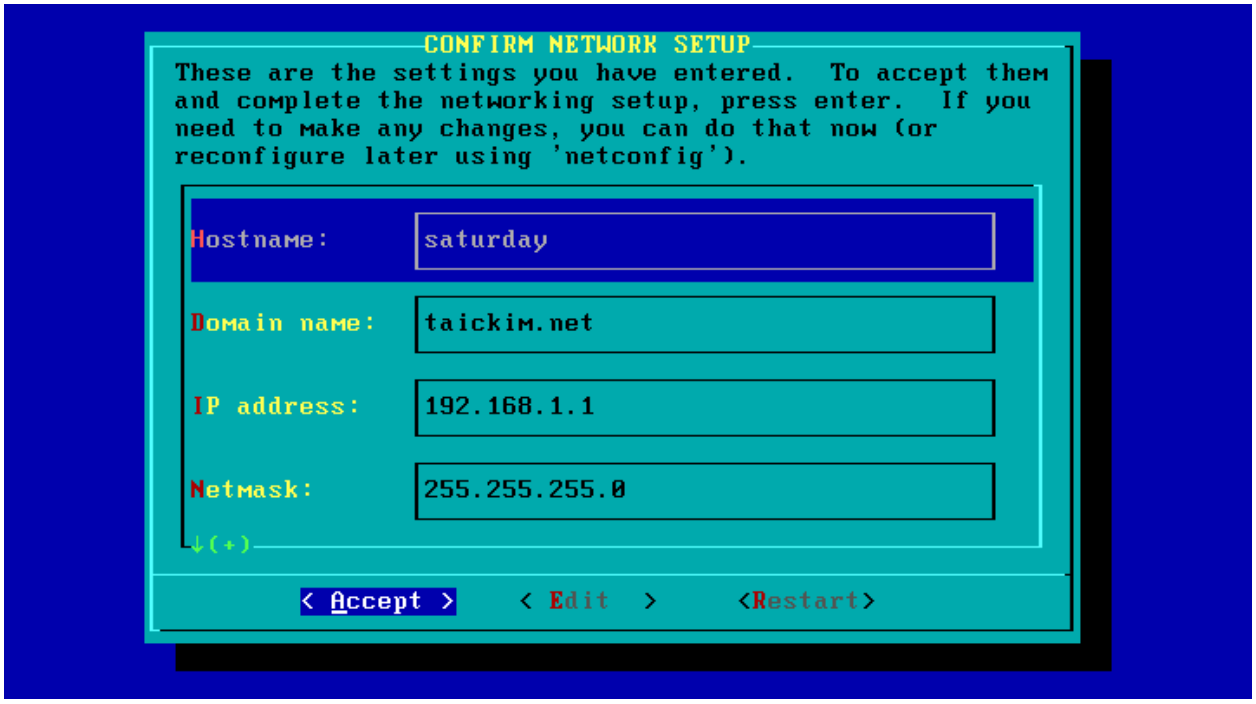
Jika Anda memilih untuk menggunakan nameserver, Anda akan diberi kesempatan untuk menentukan alamat IP dari nameserver (Gambar 5.27, “Melakukan setting nameserver”).

Gambar 5.27. Melakukan setting nameserver



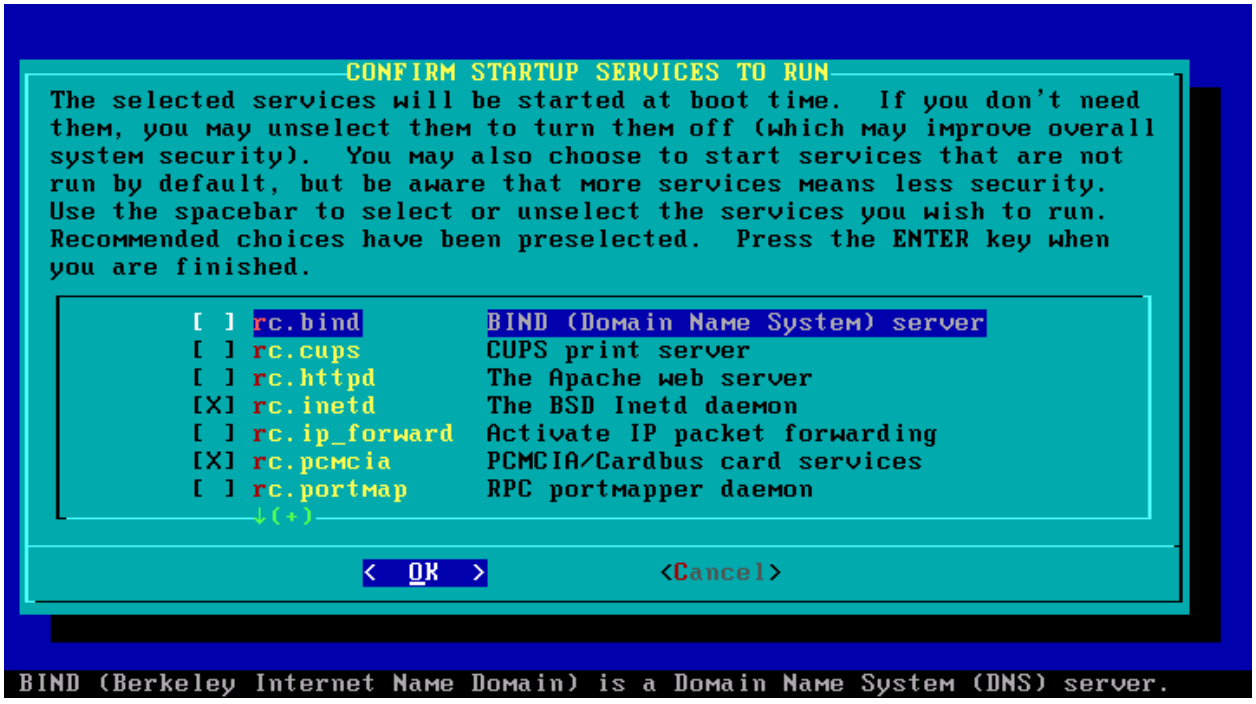
Layar terakhir konfigurasi jaringan menyediakan gambaran dari setting, memberikan kesempatan bagi Anda untuk memperbaiki setting yang memiliki kesalahan (Gambar 5.28, “Mengkonfirmasi setting jaringan”).

Gambar 5.28. Mengkonfirmasi setting jaringan



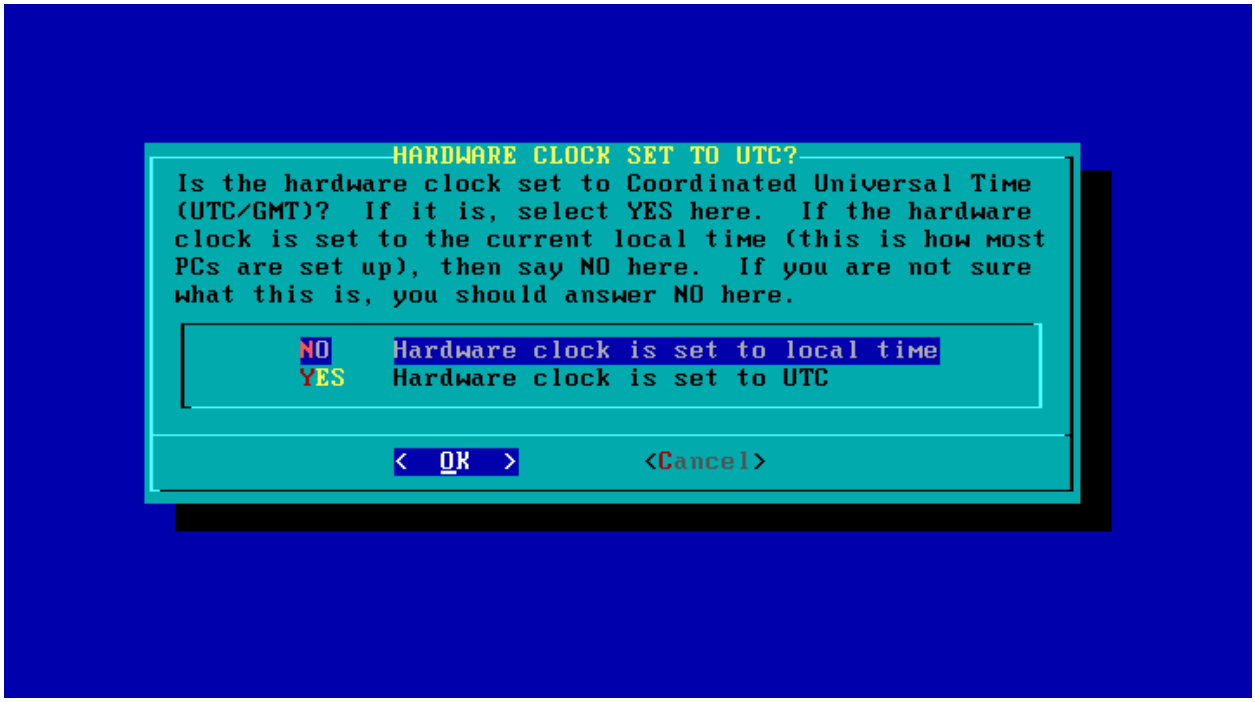
Setelah konfigurasi jaringan, Anda bisa menentukan perangkat mana yang harus dijalankan (Gambar 5.29, ? Mengaktifkan/menonaktifkan layanan yang dijalankan saat startup?). Anda bisa memberikan tanda atau menghilangkan tanda dengan tombol <SPACE>.

Gambar 5.29. Mengaktifkan/menonaktifkan layanan yang dijalankan saat startup



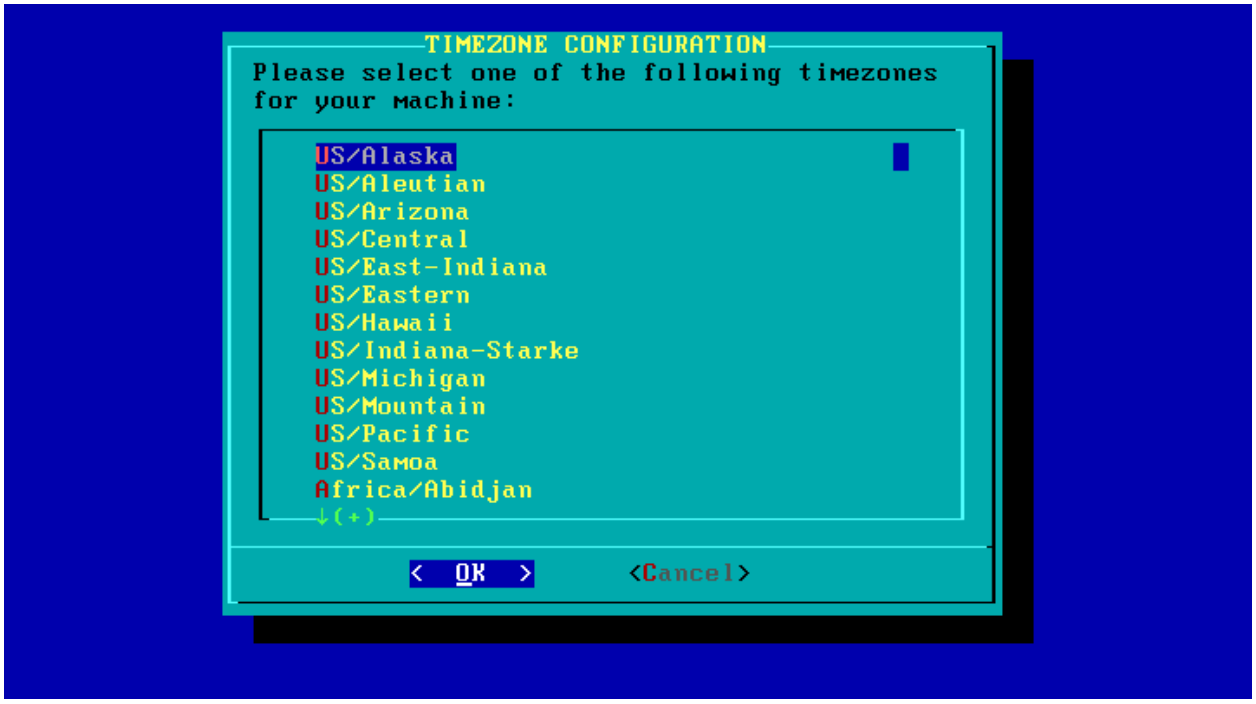
Biasanya, waktu sistem diset ke zona waktu UTC pada sistem berbasis UNIX. Jika ini kasusnya, pilih *Yes* pada langkah selanjutnya (Gambar 5.30, “Memilih apakah jam diset ke UTC”). Jika Anda juga menggunakan sistem operasi yang tidak berbasis UNIX pada sistem yang sama, seperti Windows, disarankan untuk memilih *No*, karena beberapa sistem operasi PC tidak bekerja dengan waktu sistem dan waktu perangkat lunak yang berbeda.

Gambar 5.30. Memilih apakah jam diset ke UTC



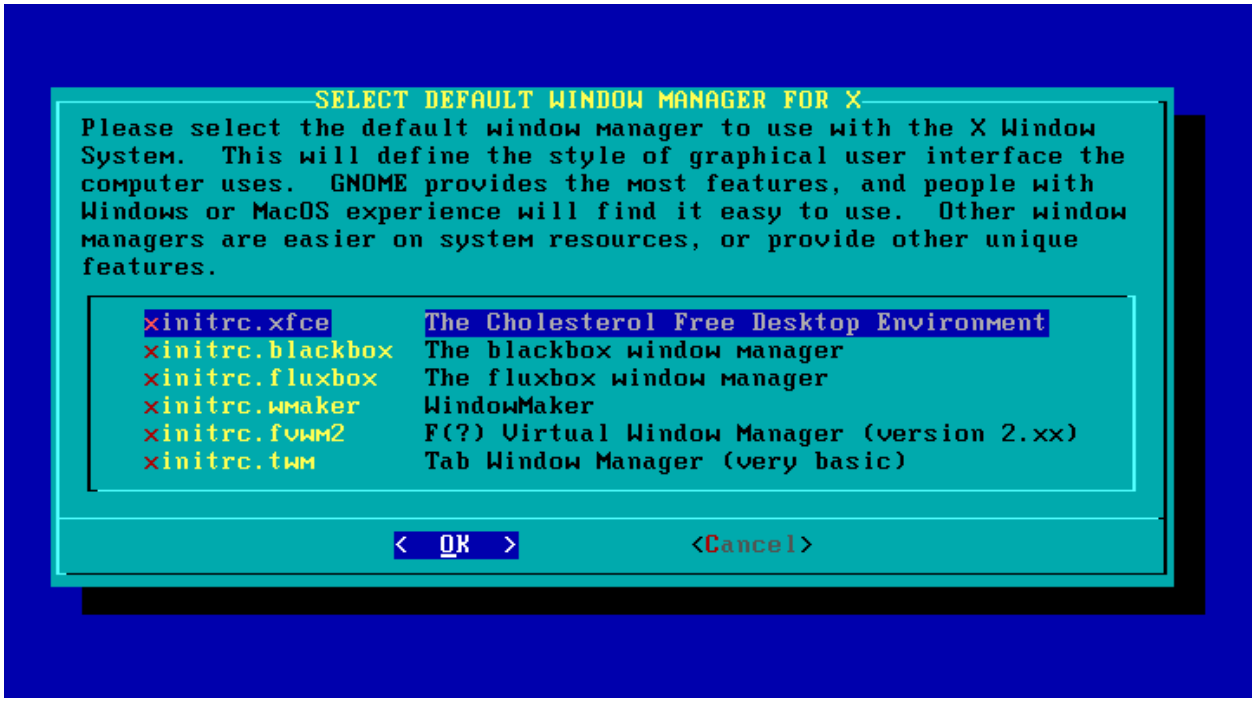
Anda akan diberi kesempatan untuk memilih zona waktu (Gambar 5.31, “Melakukan setting zona waktu”). Hal ini sangat penting terutama pada sistem yang memiliki waktu sistemnya diset ke UTC, tanpa memilih zona waktu yang benar, maka waktu pada perangkat lunak tidak akan sama dengan waktu lokal.

Gambar 5.31. Melakukan setting zona waktu



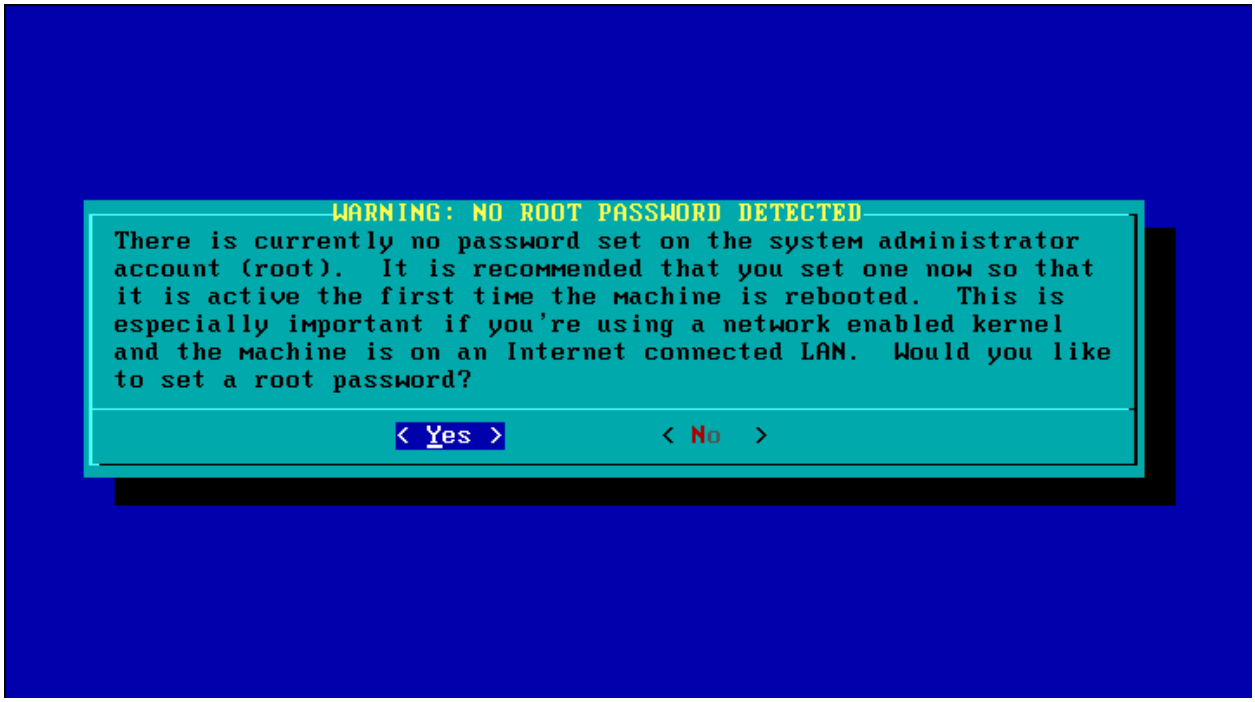
Jika Anda menginstall X Window System Anda bisa menentukan window manager default (Gambar 5.32, “Memilih window manager default”). Fungsionalitas paling dasar dari window manager adalah menyediakan fungsionalitas window dasar seperti title bar. Tetapi beberapa opsi seperti KDE, menyediakan lingkungan desktop yang lengkap.

Gambar 5.32. Memilih window manager default



Langkah terakhir adalah menentukan kata sandi root (Gambar 5.33, “Melakukan setting kata sandi root”). Setup akan menanyakan apakah Anda hendak menentukannya atau tidak. Tidak ada alasan untuk tidak melakukannya, dan tanpa kata sandi root, sistem Anda sangat tidak aman.

Gambar 5.33. Melakukan setting kata sandi root



Pada tahap ini, Anda sudah menyelesaikan installasi Slackware Linux. Anda bisa reboot sistem untuk memulai sistem Slackware Linux baru Anda. Tidak sulit bukan? ;-)

Gambar 5.34. Selesai

Bab 6. Modifikasi Installasi

Seringkali Anda hendak melakukan modifikasi terhadap installasi Slackware Linux, misalnya untuk memahami bagaimana sistem GNU/Linux bekerja, atau mempersiapkan script installasi otomatis. Bab ini membahas langkah-langkah yang diperlukan untuk menginstall Slackware Linux secara manual. Sebuah contoh script installasi juga disediakan pada Bagian 6.5, “Script installasi otomatis”.

6.1. Mempartisi hard disk

Jika Anda sudah melakukan installasi manual, Anda tidak akan mengalami masalah dalam mempartisi hard disk. Anda bisa menggunakan perintah **fdisk** dan **cfdisk** untuk mempartisi disk. Jika Anda membuat script installasi Slackware Linux, maka akan berguna untuk diketahui bahwa Anda bisa menggunakan pipe untuk perintah **fdisk** ke **fdisk**. Sebagai contoh:

```
# fdisk /dev/hda << EOF
n
p
1

+10000M
n
p
2

+512M
t
2
82
w
EOF
```

Perintah-perintah ini menghasilkan partisi Linux utama sebesar 10000MB, dan partisi swap Linux utama sebesar 512MB. Anda bisa menyimpan perintah **fdisk** pada profil disk yang berbeda dan menggunakan salah satu profil sesuai kondisi tertentu (misalnya ukuran disk). Sebagai contoh:

```
# cat /usr/share/fdisk-profiles/smalldisk | fdisk
```

6.2. Menginisialisasi dan me-mount sistem berkas

Setelah membuat minimal partisi swap dan Linux, Anda bisa menginisialisasi sistem berkas dan ruang swap dan menggunakan ruang penyimpanan ini. Pada sistem dengan jumlah memory yang terbatas, Anda harus menginisialisasi dan menggunakan swap terlebih dahulu. Kita akan menggunakan layout partisi yang digunakan pada contoh diatas pada contoh berikutnya. Untuk melakukan setting dan menggunakan swap, jalankan:

```
# mkswap /dev/hda2
```

```
# swapon /dev/hda2
```

Arti dari perintah diatas cukup jelas. **mkswap** menginisialisasi ruang space, dan **swapon** menggunakannya. Anda hanya perlu menjalankan **mkswap** satu kali, tetapi **swapon** harus dijalankan setiap kali sistem dijalankan. Hal ini bisa dilakukan secara otomatis dengan menambahkan sebuah entri untuk partisi swap pada `/etc/fstab`, yang akan kita lakukan nanti.

Untuk saat ini, sangatlah penting untuk menginisialisasi partisi target. Hal ini bisa dilakukan dengan perintah **mkfs**. Anda bisa menentukan sistem berkas yang akan digunakan dengan menambahkan parameter `-t`. **mkfs** akan memanggil perintah **mkfs.filesystem** secara otomatis sesuai dengan sistem berkas yang Anda pilih. Hati-hati bahwa sistem berkas yang dapat digunakan tergantung dari kernel installasi yang Anda gunakan. Jika Anda menggunakan kernel *bare.i*, Anda bisa menggunakan sistem berkas *ext2*, *ext3* dan *reiserfs*.

Untuk menginisialisasi sistem berkas *ext3*, dan melakukan proses mount, Anda harus menjalankan perintah berikut:

```
# mkfs -t ext3 /dev/hda1
# mount /dev/hda1 /mnt
```

Jika Anda sudah menyediakan partisi yang terpisah untuk beberapa direktori tertentu pada sistem berkas root, misalnya `/home`, Anda bisa melakukan inisialisasi dan melakukan proses mount saat ini. Sebagai contoh:

```
# mkfs -t ext3 /dev/hda2
# mkdir /mnt/home
# mount /dev/hda2 /mnt/home
```

Akhirnya, Anda harus melakukan mount terhadap media sumber Anda. Jika Anda menggunakan CD-ROM, hal ini sangatlah mudah. Misalkan `/dev/hdc` adalah berkas untuk CD-ROM, Anda bisa melakukan mount CD-ROM dengan:

```
# mount /dev/hdc /var/log/mount
```

Menggunakan NFS sebagai media installasi membutuhkan beberapa langkah tambahan. Pertama, Anda harus memuat disk jaringan. Anda bisa melakukannya dengan menjalankan perintah **network** dan memasukkan disk jaringan. Anda bisa memuat disk ini dari media lain, misalnya dari memori USB. Misalnya Anda telah melakukan mount dari memori USB pada /var/log/mount, Anda bisa memuat disk jaringan dengan: **network /var/log/mount/network.dsk**. Setelah memuat disk jaringan, Anda harus mengkonfigurasi antarmuka jaringan. Jika server NFS berada pada jaringan yang sama, Anda hanya perlu memberikan alamat IP pada antarmuka jaringan. Misalnya, untuk menggunakan alamat *192.168.2.201*, Anda bisa menjalankan perintah berikut:

```
# ifconfig eth0 192.168.2.201
```

Anda bisa memuat portmapper, yang diperlukan agar klien NFS bisa bekerja dengan benar:

```
# /sbin/rpc.portmap
```

Jika portmapper berjalan dengan baik, Anda bisa melakukan mount terhadap isi NFS. Tetapi, pastikan Anda sudah melakukan unmount semua sistem berkas yang Anda mount pada `/var/log/mount`. Jika tidak ada sistem berkas lain yang di-mount pada `/var/log/mount`, Anda bisa melanjutkan dengan me-mount isi NFS. Misalnya Anda hendak me-mount *192.168.2.1:/home/pub/slackware-current*, Anda harus menjalankan perintah berikut:

```
# mount -r -t nfs -o nolock 192.168.2.1:/home/pub/slackware-current /var/log/mount
```

Jika tidak ada kesalahan yang dicetak ketika isi di-mount, maka isi akan dapat diakses melalui `/var/log/mount`

6.3. Menginstall paket-paket

Semua sudah siap untuk memulai menginstall paket-paket dari media installasi. Karena **installpkg** tersedia dari sistem installasi, Anda bisa menggunakannya untuk menginstall paket-paket Slackware Linux. Untuk menginstall paket-paket pada partisi target yang di-mount pada `/mnt`, tambahkan opsi `-root`. Perintah berikut akan menginstall semua paket dari media sumber:

```
# installpkg -root /mnt /var/log/mount/slackware/*/*.tgz
```

Jika Anda telah membuat berkas tag untuk mendefinisikan paket-paket mana yang harus diinstall, maka Anda bisa menggunakannya sekarang (berkas tag dijelaskan pada Bagian 17.4, “Tagfile”). Misalkan Anda telah menyimpan berkas tag untuk setiap set disk pada `/usr/share/tagfiles/small-desktop`, Anda bisa menginstall paket-paket berdasarkan berkas tag dengan cara berikut:

```
# for p in a ap d e f k kde kdei l n t tcl x xap y; do
   installpkg -infobox -root /mnt \
     -tagfile /usr/share/tagfiles/small-desktop/$p/tagfile \
     /var/log/mount/slackware/$p/*.tgz
done
```

6.4. Konfigurasi pasca-installasi

Bagian berikutnya menjelaskan konfigurasi minimal yang diperlukan untuk menjalankan sistem.

fstab

Salah satu langkah konfigurasi penting adalah membuat berkas **fstab**, sehingga sistem bisa mencari partisi atau isi apa yang harus di-mount. Format dari berkas `/etc/fstab` dijelaskan pada bagian bernama “Berkas fstab”. Pada batas minimal Anda harus menambahkan isian untuk sistem berkas `/`, sistem berkas pseudo `/proc`, `devpts`, dan partisi swap.

Dengan contoh partisi yang digunakan di awal bab ini, Anda bisa membuat `/etc/fstab` seperti ini:

```
#  cat > /mnt/etc/fstab << EOF
/dev/hda2  swap             swap        defaults         0   0
/dev/hda1        /               ext3        defaults         1   1
devpts           /dev/pts        devpts      gid=5,mode=620   0   0
```

```
proc            /proc           proc        defaults        0   0
EOF
```

LILO

Untuk membuat sistem dapat di-boot Anda harus mengkonfigurasi dan menginstall Linux Loader (LILO). Konfigurasi dari LILO dibahas pada Bagian 19.1, “Bootloader”. Untuk bagian ini kita hanya akan menunjukan contoh konfigurasi LILO, yang bisa digunakan untuk skema partisi yang dijelaskan pada bab ini. Langkah pertama adalah membuat berkas `/etc/lilo.conf`:

```
# cat > /mnt/etc/lilo.conf << EOF
boot = /dev/hda
vga = normal
timeout = 50
image = /boot/vmlinuz
 root = /dev/hda1
 label = Slackware
 read-only
EOF
```

Anda bisa menginstall LILO dengan `/mnt` sebagai induk LILO. Dengan `/mnt` sebagai induk, LILO akan menggunakan `/etc/lilo.conf` dari partisi target:

```
# lilo -r /mnt
```

Jaringan

Konfigurasi jaringan pada Slackware Linux dibahas pada Bab 22, *Konfigurasi jaringan*. Bagian ini akan membahas satu contoh dari host yang menggunakan DHCP untuk mendapatkan alamat IP.

Berkas `/etc/networks` berisi informasi tentang jaringan Internet yang dikenali. Karena kita akan mendapatkan informasi dari DHCP, kita cukup menggunakan *127.0.0.1* sebagai jaringan lokal.

```
# cat > /mnt/etc/networks << EOF
loopback        127.0.0.0
localnet        127.0.0.0
EOF
```

Meskipun kita akan mendapatkan nama host dari DHCP, kita akan memberikan nama host sementara:

```
# cat > /mnt/etc/HOSTNAME << EOF
sugaree.example.net
EOF
```

Sekarang nama host sudah dikonfigurasi, nama host dan *localhost* seharusnya bisa di-resolve, dengan menciptakan basis data `/etc/hosts`:

```
# cat > /mnt/etc/hosts << EOF
127.0.0.1 localhost
127.0.0.1 sugaree.example.net sugaree
EOF
```

Kita tidak perlu membuat `/etc/resolv.conf`, karena akan dibuat secara otomatis oleh **dhcpcd**, klien DHCP. Jadi, akhirnya kita bisa melakukan setting pada antarmuka pada `/etc/rc.d/rc.inet1.conf`:

```
# cat > /mnt/etc/rc.d/rc.inet1.conf << EOF
IPADDR[0]=""
NETMASK[0]=""
USE_DHCP[0]="yes"
DHCP_HOSTNAME[0]=""
EOF
```

Anda mungkin perlu membuat salinan dari berkas `rc.inet1.conf` yang lama, karena berisi banyak sekali komentar yang berguna. Atau Anda bisa menggunakan **sed** untuk mengganti opsi spesifik ini:

```
# sed -i 's/USE_DHCP\[0\]=""/USE_DHCP[0]="yes"/' \
   /mnt/etc/rc.d/rc.inet1.conf
```

Melakukan penyesuaian script inisialisasi

Tergantung dari tujuan sistem yang akan diinstall, harus ditentukan script inisialisasi mana yang harus dijalankan. Jumlah layanan yang tersedia tergantung dari paket apa yang telah Anda install. Anda bisa mendapatkan daftar script yang tersedia dengan **ls**:

```
# ls -l /mnt/etc/rc.d/rc.*
```

Jika bit executable (dapat dieksekusi) sudah diaktifkan pada sebuah script, maka akan dijalankan, selain itu tidak. Jelas Anda harus memastikan script yang penting executable, termasuk script untuk run-level tertentu. Anda bisa mengaktifkan bit executable pada sebuah script dengan:

```
# chmod +x /mnt/etc/rc.d/rc.scriptname
```

atau menghapusnya dengan :

```
# chmod -x /mnt/etc/rc.d/rc.scriptname
```

Mengkonfigurasi binding run-time linker dinamis

GNU/Linux menggunakan cache untuk memuat pustaka dinamis. Selain itu banyak program bergantung pada nomor versi generik dari pustaka (misalnya `/usr/lib/libgtk-x11-2.0.so`, dan bukan `/usr/lib/libgtk-x11-2.0.so.0.600.8`). Cache dan symlink pustaka dapat diupdate pada satu perintah **ldconfig**:

```
# chroot /mnt /sbin/ldconfig
```

Anda mungkin tidak tahu perintah **chroot**; ini adalah perintah yang mengeksekusi sebuah perintah dengan root yang berbeda dengan root yang aktif. Pada contoh ini direktori root diganti menjadi `/mnt`, dan dari sana **chroot** menjalankan biner **/sbin/ldconfig**. Setelah perintah selesai, sistem akan kembali ke shell, yang menggunakan root awal. Untuk menggunakan **chroot** dengan perintah lain, perintah **ldconfig** harus dijalankan dahulu minimal satu kali, karena tanpa melakukan inisialisasi, sebagian besar perintah tidak akan dieksekusi, karena ketergantungan pustaka yang tidak dapat diselesaikan.

Menentukan kata sandi root

Sekarang cache pustaka dinamis dan link sudah selesai, Anda bisa menjalankan perintah pada sistem yang terinstall. Kita akan memanfaatkannya untuk menentukan kata sandi *root*. Perintah **passwd** bisa digunakan untuk menentukan kata sandi untuk pengguna yang sudah ada (pengguna *root* adalah bagian dari berkas `/etc/passwd` awal). Kita akan menggunakan perintah **chroot** lagi untuk mengeksekusi perintah pada partisi target:

```
# chroot /mnt /usr/bin/passwd root
```

Menentukan zona waktu

Pada sistem UNIX-like, sangatlah penting untuk menentukan zona waktu dengan benar, karena digunakan oleh banyak bagian dari sistem. Sebagai contoh, zona waktu digunakan oleh NTP untuk melakukan sinkronisasi waktu sistem dengan benar, atau oleh banyak program jaringan lainnya untuk menghitung perbedaan waktu antara klien dan server. Pada Slackware Linux zona waktu bisa ditentukan dengan menghubungkan `/etc/localtime` dengan berkas zona waktu. Anda bisa menemukan zona waktu untuk regional Anda dengan mengunjungi direktori `/mnt/usr/share/zoneinfo`. Sebagian besar zona waktu dapat ditemukan pada sub direktori dari regional masing-masing. Sebagai contoh untuk menggunakan *Europe/Amsterdam* sebagai zona waktu, Anda bisa mengeksekusi perintah berikut:

```
# cd /mnt
# rm -rf etc/localtime
# ln -sf /usr/share/zoneinfo/Europe/Amsterdam etc/localtime
```

Setelah menentukan zona waktu, program masih tidak tahu apakah jam perangkat keras ditentukan ke waktu lokal atau ke standar Coordinated Universal Time (UTC). Jika Anda menggunakan sistem operasi lain pada mesin yang sama yang tidak ditentukan ke UTC, maka lebih baik menentukan jam perangkat keras ke waktu lokal. Pada sistem UNIX-like sangatlah biasa untuk menentukan jam sistem ke UTC. Anda bisa menentukan waktu sistem yang digunakan dengan menciptakan berkas `/etc/hardwareclock`, dan memberikan isi *localtime* didalamnya jika waktu Anda ditentukan ke waktu lokal, atau *UTC* jika menggunakan UTC. Sebagai contoh, untuk menggunakan UTC, Anda bisa menciptakan `/etc/hardwareclock` dengan cara berikut:

```
# echo "UTC" > /mnt/etc/hardwareclock
```

Membuat cache font X11

Jika Anda hendak menggunakan X11, Anda harus melakukan inisialisasi cache font untuk jenis font TrueType dan Type1. Hal ini bisa dilakukan dengan perintah **fc-cache**:

```
# chroot /mnt /usr/bin/fc-cache
```

6.5. Script installasi otomatis

Sangatlah mudah untuk mengkombinasikan langkah-langkah dari installasi modifikasi pada satu script, yang melakukan langkah-langkah secara otomatis. Hal ini ideal untuk membuat installasi server default atau melakukan installasi masal pada klien Linux. Bagian ini berisi contoh script yang ditulis oleh William Park. Sangatlah mudah untuk menambahkan script installasi pada media Slackware Linux, terutama jika Anda menggunakan installasi CD-ROM atau melakukan boot sistem installasi dari flash drive USB.

Sistem installasi tersimpan didalam satu citra terkompres, yang tersedia pada media distribusi sebagai `isolinux/initrd.img`. Anda bisa membuat salinan dari citra ini pada hard disk Anda, dan melakukan dekompresi dengan **gunzip**:

```
# mv initrd.img initrd.img.gz
# gunzip initrd.img.gz
```

Setelah mendekompresi citra, Anda bisa melakukan mount berkas sebagai disk, menggunakan loopback:

```
# mount -o loop initrd.img /mnt/hd
```

Anda bisa menambahkan script pada berkas initrd dengan menambahkannya pada struktur direktori yang tersedia dibawah mount point. Setelah melakukan perubahan yang diperlukan, Anda bisa unmount sistem berkas dan mengkompresnya kembali:

```
# umount /mnt/hd
# gzip initd.img
# mv initrd.img.gz initrd.img
```

Anda bisa meletakkan berkas `initrd.img` yang baru pada media installasi, dan mencoba script.

```
#! /bin/sh
# Copyright (c) 2003-2005 by William Park <opengeometry@yahoo.ca>.
# All rights reserved.
#
# Usage: slackware-install.sh

rm_ln ()  # Usage: rm_ln from to
{
 rm -rf $2; ln -sf $1 $2
}


##############################################################################
echo "Partitioning harddisk..."
```

```
(   echo -ne "n\np\n1\n\n+1000M\n" # /dev/hda1 --> 1GB swap
 echo -ne "n\np\n2\n\n+6000M\n" # /dev/hda2 --> 6GB /
 echo -ne "t\n1\n82\nw\n"
) | fdisk /dev/hda

mkswap /dev/hda1  # swap
swapon /dev/hda1

mke2fs -c /dev/hda2  # /
mount /dev/hda2 /mnt


################################################################################
echo "Installing packages..."

mount -t iso9660 /dev/hdc /cdrom # actually, /var/log/mount
cd /cdrom/slackware
for p in [a-z]*; do  # a, ap, ..., y
 installpkg -root /mnt -priority ADD $p/*.tgz
done

cd /mnt


################################################################################
echo "Configuring /dev/* stuffs..."

rm_ln psaux dev/mouse  # or, 'input/mice' for usb mouse
rm_ln ttyS0 dev/modem
rm_ln hdc dev/cdrom
rm_ln hdc dev/dvd


################################################################################
echo "Configuring /etc/* stuffs..."

cat > etc/fstab << EOF
/dev/hda1   swap         swap     defaults         0  0
/dev/hda2   /            ext2     defaults         1  1
devpts      /dev/pts     devpts   gid=5,mode=620   0  0
proc        /proc        proc     defaults         0  0
#
/dev/cdrom  /mnt/cdrom   iso9660  noauto,owner,ro  0  0
/dev/fd0    /mnt/floppy  auto     noauto,owner     0  0
tmpfs       /dev/shm     tmpfs    noauto           0  0
EOF

cat > etc/networks << EOF
loopback 127.0.0.0
localnet 192.168.1.0
EOF
cat > etc/hosts << EOF
127.0.0.1 localhost
192.168.1.1 node1.example.net node1
```

```
EOF
cat > etc/resolv.conf << EOF
search example.net
nameserver 127.0.0.1
EOF
cat > etc/HOSTNAME << EOF
node1.example.net
EOF

## setup.05.fontconfig
chroot . /sbin/ldconfig  # must be run before other program
chroot . /usr/X11R6/bin/fc-cache

chroot . /usr/bin/passwd root

## setup.06.scrollkeeper
chroot . /usr/bin/scrollkeeper-update

## setup.timeconfig
rm_ln /usr/share/zoneinfo/Canada/Eastern  etc/localtime
cat > etc/hardwareclock << EOF
localtime
EOF

## setup.liloconfig
cat > etc/lilo.conf << EOF
boot=/dev/hda
delay=100
vga=normal # 80x25 char
# VESA framebuffer console:
#   pixel char 8bit 15bit 16bit 24bit
#   ----- ---- ---- ----- ----- -----
#   1600x1200  796 797 798 799
#   1280x1024 160x64 775 793 794 795
#   1024x768 128x48 773 790 791 792
#   800x600 100x37 771 787 788 789
#   640x480 80x30 769 784 785 786
image=/boot/vmlinuz  # Linux
 root=/dev/hda2
 label=bare.i
 read-only
# other=/dev/hda1  # Windows
#     label=win
#     table=/dev/hda
EOF
lilo -r .

## setup.xwmconfig
rm_ln xinitrc.fvwm95  etc/X11/xinit/xinitrc


################################################################################
echo "Configuring /etc/rc.d/rc.* stuffs..."
```

```
cat > etc/rc.d/rc.keymap << EOF
#! /bin/sh
[ -x /usr/bin/loadkeys ] && /usr/bin/loadkeys us.map
EOF
chmod -x etc/rc.d/rc.keymap

## setup.mouse
cat > etc/rc.d/rc.gpm << 'EOF'
#! /bin/sh
case $1 in
 stop)
     echo "Stopping gpm..."
     /usr/sbin/gpm -k
     ;;
 restart)
     echo "Restarting gpm..."
     /usr/sbin/gpm -k
     sleep 1
     /usr/sbin/gpm -m /dev/mouse -t ps2
     ;;
 start)
     echo "Starting gpm..."
     /usr/sbin/gpm -m /dev/mouse -t ps2
     ;;
 *)
     echo "Usage $0 {start|stop|restart}"
     ;;
esac
EOF
chmod +x etc/rc.d/rc.gpm

## setup.netconfig
cat > etc/rc.d/rc.inet1.conf << EOF
IPADDR=(192.168.1.1)  # array variables
NETMASK=(255.255.255.0)
USE_DHCP=()   # "yes" or ""
DHCP_HOSTNAME=()
GATEWAY=""
DEBUG_ETH_UP="no"
EOF

cat > etc/rc.d/rc.netdevice << EOF
/sbin/modprobe 3c59x
EOF
chmod +x etc/rc.d/rc.netdevice

## setup.setconsolefont
mv etc/rc.d/rc.font{.sample,}
chmod -x etc/rc.d/rc.font

## setup.services
chmod +x etc/rc.d/rc.bind
chmod +x etc/rc.d/rc.hotplug
chmod +x etc/rc.d/rc.inetd
```

```
chmod +x etc/rc.d/rc.portmap
chmod +x etc/rc.d/rc.sendmail
#
chmod -x etc/rc.d/rc.atalk
chmod -x etc/rc.d/rc.cups
chmod -x etc/rc.d/rc.httpd
chmod -x etc/rc.d/rc.ip_forward
chmod -x etc/rc.d/rc.lprng
chmod -x etc/rc.d/rc.mysqld
chmod -x etc/rc.d/rc.pcmcia
chmod -x etc/rc.d/rc.samba
chmod -x etc/rc.d/rc.sshd
```

Bagian II. Dasar-Dasar Slackware Linux

Daftar Isi

Bab 7. Shell

7.1. Perkenalan

Pada bab ini kita akan melihat lingkungan kerja tradisional dari sistem UNIX: shell. Shell adalah sebuah interpreter yang dapat digunakan secara interaktif dan non-interaktif. Ketika shell digunakan pada mode non-interaktif, fungsinya sebagai bahasa pemrograman yang sederhana, namun bagus.

Prosedur untuk menjalankan shell tergantung dari apakah Anda menjalankan login berbasis grafis atau teks. Jika Anda login pada mode teks, maka shell sudah langsung dijalankan setelah memasukkan kata sandi yang benar. Jika Anda menjalankan manajer login grafis seperti KDM, lakukan proses login secara normal, dan lihat pada manajer jendela atau menu desktop untuk sebuah isian bernama “XTerm”, “Terminal” atau “Konsole”. XTerm adalah emulator terminal, setelah emulator terminal dijalankan, shell akan muncul.

Sebelum kita pergi lebih jauh, kami harus harus memperingatkan kepada Anda bahwa Slackware Linux menyediakan lebih dari sekedar satu shell. Terdapat dua rasa shell yang menjadi populer, Bourne shell dan C shell. Pada bab ini kita akan membahas Bourne shells yang sesuai dengan standar IEEE 1003.1. Shell Bash (Bourne Again Shell) dan ksh (Korn Shell) sesuai dengan standar ini juga. Jadi merupakan ide bagus untuk menggunakan salah satu dari dua shell ini. Anda bisa melihat shell yang digunakan pada sistem dengan menjalankan **echo $SHELL**. Ini yang akan dilaporkan pada shell Bash:

```
$ echo $SHELL
/bin/bash
```

Jika Anda menggunakan shell yang berbeda, Anda bisa mengganti shell default Anda. Sebelum mengganti shell yang berbeda, Anda harus menyediakan path lengkap dari shell. Anda bisa melakukannya dengan perintah **which**. Misalnya:

```
$ which bash
/bin/bash
$ which ksh
/bin/ksh
```

Pada sistem Slackware, path lengkap terhadap shell bash adalah `/bin/bash`, dan shell ksh `/bin/ksh`. Dengan informasi ini, dan perintah **chsh** Anda bisa mengganti shell default. Contoh berikut mengganti shell default menjadi bash:

```
$ chsh -s /bin/bash
Changing shell for daniel.
Password:
Shell changed.
```

Shell baru akan diaktifkan setelah keluar dari shell yang saat ini sedang dipakai (dengan **logout** atau **exit**), atau dengan membuka jendela terminal X baru jika Anda menjalankan X11.

7.2. Mengeksekusi perintah-perintah

Shell interaktif digunakan untuk menjalankan program dengan mengeksekusi perintah-perintah. Terdapat dua jenis perintah yang dapat dijalankan shell:

- *Perintah bawaan (built-in):* perintah built-in terintegrasi pada shell. Perintah yang paling sering digunakan adalah: **cd**, **fg**, **bg**, dan **jobs**.
- *Perintah eksternal:* perintah eksternal adalah program yang bukan merupakan bagian dari program shell, dan disimpan secara terpisah pada sistem berkas. Perintah yang paling sering digunakan : **ls**, **cat**, **rm**, dan **mkdir**.

Semua perintah shell dieksekusi dengan sintaks yang sama:

```
perintah [argumen1 argumen2 ... argumenN]
```

Jumlah parameter bebas dan selalu dikirimkan ke perintah. Perintah bisa menentukan apa yang harus dilakukan dengan argumen.

Semua perintah built-in selalu dapat dieksekusi, karena merupakan bagian dari shell. Perintah eksternal bisa dieksekusi ketika program masuk didalam pencarian path oleh shell. Jika tidak, Anda harus menentukan path ke program. Pencarian path dari shell disimpan pada variabel bernama *PATH*. Sebuah varibel adalah bagian dari memory, dimana nilainya bisa diganti. Kita bisa melihat isi dari variabel PATH dengan cara berikut:

```
$ echo $PATH
/usr/kerberos/bin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/home/daniel/bin
```

Path direktori pada variabel *PATH* dipisahkan dengan karakter titik dua (*:*). Anda bisa menggunakan perintah **which** untuk memeriksa apakah perintah yang diberikan terdapat pada path shell. Anda bisa melakukannya dengan memberikan perintah sebagai argumen bagi **which**. Sebagai contoh:

```
$ which pwd
/bin/pwd
$ which sysstat
/usr/bin/which: no sysstat in (/usr/kerberos/bin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:
```

Jika sebuah program bukan bagian dari path, Anda masih bisa menjalankannya dengan memasukkan path absolut atau relatifnya.

7.3. Berpindah-pindah

Seringkali kita perlu melompat pada berbagai bagian dari sebuah baris dan mengubahnya, ketika Anda mengedit perintah yang besar. Baik **bash** dan **ksh** memiliki jalan pintas (shortcut) untuk melakukan beberapa operasi umum. Terdapat dua mode shell, yang memiliki dua kunci shortcut yang berbeda. Mode ini bersesuaian dengan dua editor UNIX yang populer. Editor ini adalah **vi** dan **emacs**. Pada buku ini kita hanya akan membahas kunci EMACS-like. Anda bisa melihat pada mode apa yang digunakan oleh shell yang sedang digunakan dengan mencetak variabel *SHELLOPTS*. Pada contoh pertama shell digunakan pada mode *emacs*, pada contoh kedua digunakan mode *vi*. Anda bisa mengidentifikasi mode dengan melihat pada string *emacs* atau *vi* pada isi dari variabel.

```
$ echo $SHELLOPTS
braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor
```

```
$ echo $SHELLOPTS
braceexpand:hashall:histexpand:history:interactive-comments:monitor:vi
```

Jika shell Anda menggunakan mode *vi*, Anda bisa berpindah ke mode *emacs* dengan menentukan opsi *emacs*:

```
$ set -o emacs
```

Dengan mode pengeditan *emacs* diaktifkan, Anda bisa menjalankan dengan shortcut. Kita akan melihat tiga jenis shortcut: pengeditan karakter, pengeditan kata, dan pengeditan baris. Pada bagian lain dari bab ini, kita juga akan emlihat pada beberapa shortcut yang digunakan untuk mengambil isian dari sejarah perintah.

Pengeditan karakter

Grup pertama dari shortcut menggunakan karakter sebagai unit logisnya, berarti mereka mengijinkan operasi pengeditan perintah baris pada karakter. Tabel 7.1, “Berpindah per karakter” menyediakan gambaran dari shortcut yang digunakan untuk berpindah melalui baris pada karakter.

Tabel 7.1. Berpindah per karakter

Kunci	Deskripsi
Ctrl-b	Pindah satu karakter kebelakang.
Ctrl-f	Pindah satu karakter kedepan.

Shortcut ini sangat sederhana, dan tidak melakukan hal-hal yang tidak diinginkan. Misalkan Anda sudah menulis baris seperti ini:

```
find ~/music -name '*.ogg' - -print
```

Kursor akan berada di akhir. Anda bisa berpindah ke awal baris dengan menekan *Ctrl-b*:

```
find ~/music - -name '*.ogg' -print
```

Anda juga bisa kembali ke akhir baris dengan menekan *Ctrl-f*. Terdapat kesalahan pada baris ini, karena terdapat satu karakter yang salah (-). Untuk menghilangkan karakter ini, Anda bisa menggunakan salah satu shortcut untuk penghapusan karakter.

Tabel 7.2. Menghapus karakter

Kunci	Deskripsi
Ctrl-h	Menghapus sebuah karakter sebelum kursor. Hal ini memiliki efek yang sama dengan menggunakan kunci Backspace pada sebagian besar komputer pribadi.
Ctrl-d	Menghapus karakter pada posisi kursor berada.

Anda bisa menghapus karakter minus (-) dengan dua cara. Cara pertama adalah dengan memindahkan kursor ke karakter yang akan dihapus:

```
find ~/music - -name '*.ogg' -print
```

lalu tekan *Ctrl-d* dua kali. Hal ini akan menghapus karakter minus, dan spasi yang mengikuti karakter minus:

```
find ~/music -name '*.ogg' -print
```

Dengan melihat pada potongan aslinya, pendekatan lain adalah dengan meletakkan posisi kursor pada spasi setelah tanda minus:

```
find ~/music - -name '*.ogg' -print
```

lalu menekan *Ctrl-h* dua kali untuk menghapus dua karakter sebelumnya, yaitu karakter minus dan spasi sebelum karakter minus. Hasilnya akan sama, kecuali kursornya akan berpindah:

```
find ~/music -name '*.ogg' -print
```

Salah satu fitur menarik dari shell modern adalah Anda bisa melakukan transposisi (menukar) karakter. Hal ini cukup berguna jika Anda melakukan kesalahan penulisan dimana dua karakter tertukar. Tabel 7.3, “Menukar karakter” mendaftar shortcut untuk pertukaran karakter.

Tabel 7.3. Menukar karakter

Kunci	Deskripsi
Ctrl-t	Menukar (transposisi) karakter dimana kursor berada dan karakter sebelum kursor. Hal ini berguna untuk memperbaiki kesalahan pengetikan dengan cepat.

Misalkan Anda sudah menuliskan perintah berikut:

```
cat myreport.ttx
```

Ekstensi memiliki kesalahan kesalahan pengetikan karena Anda hendak menuliskan **cat** `myreport.txt`. Hal ini bisa diperbaiki dengan shortcut transposisi karakter. Pertama-tama pindahkan kursor ke karakter kedua dari pasangan karakter yang berada pada urutan yang salah:

```
cat myreport.ttx
```

Anda bisa menekan *Ctrl-t*. Karakter akan ditukar dan kursor akan diletakkan diantara karakter yang dipertukarkan:

```
cat myreport.txt
```

Pengeditan kata

Seringkali diperlukan perpindahan pada level karakter. Untungnya shell Korn dan Bash juga bisa berpindah baris pada level kata. Kata adalah urutan dari karakter yang dipisahkan oleh karakter khusus, misalnya spasi. Tabel 7.4, “Berpindah per kata” mendaftar shortcut yang bisa digunakan untuk melakukan navigasi baris per kata.

Tabel 7.4. Berpindah per kata

Kunci	Deskripsi
Esc b	Pindah ke awal dari kata aktual atau kata sebelumnya.
Esc f	Pindah maju ke karakter terakhir dari kata aktual atau kata selanjutnya.

Seperti yang Anda lihat bahwa huruf pada shortcut sama dengan ketika berpindah per karakter. Logika dari perubahan ini cukup menarik. Berpindah kedepan meletakkan kursor di akhir dari kata aktual, bukan pada karakter pertama dari kata selanjutnya seperti yang Anda prediksikan. Mari kita lihat pada sebuah contoh. Pada awalnya kursor berada pada karakter pertama dari sebuah baris.

```
find ~/music -name '*.ogg' -print
```

Menekan *Esc f* akan memindahkan kursor diakhir karakter pertama, yang berupa *find* pada kasus ini:

```
find ~/music -name '*.ogg' -print
```

Maju kedepan sekali akan meletakkan kursor dibelakang *~/music*:

```
find ~/music -name '*.ogg' -print
```

Perpindahan kebelakang meletakkan kursor di karakter pertama dari kata aktual, atau karakter pertama dari kata sebelumnya jika kursor berada di karakter pertama dari sebuah kata. Jadi berpindah kebelakang satu kata pada contoh sebelumnya akan meletakkan kursor pada kata pertama dari “music”:

```
find ~/music -name '*.ogg' -print
```

Menghapus kata memiliki cara kerja yang sama dengan perpindahan kata, tetapi karakter yang ditemukan akan dihapus. Tabel 7.5, “Menghapus kata” mendaftar shortcut yang digunakan untuk menghapus kata.

Tabel 7.5. Menghapus kata

Kunci	Deskripsi
Alt-d	Menghapus kata, dimulai dari posisi kursor aktual.

Kunci	Deskripsi
Alt-Backspace	Menghapus setiap karakter dari posisi kursor aktual hingga karakter pertama dari kata yang ditemukan.

Akhirnya, terdapat beberapa shortcut yang berguna untuk memanipulasi kata. Shortcut ini dapat dilihat pada Tabel 7.6, “Memodifikasi kata”.

Tabel 7.6. Memodifikasi kata

Kunci	Deskripsi
Alt-t	Menukar (transposisi) kata aktual dengan kata sebelumnya.
Alt-u	Membuat kata menjadi huruf kapital, dimulai dari kursor aktual.
Alt-l	Membuat kata menjadi huruf kecil, dimulai dari kursor aktual.
Alt-c	Membuat karakter aktual menjadi huruf kapital atau karakter pertama dari kata selanjutnya yang ditemukan.

Transposisi menukar antar kata. Jika kata biasa digunakan, maka perilakunya bisa diprediksi. Misalnya jika kita memiliki baris berikut dengan kursor berada pada “two”

```
one two three
```

Transposisi kata akan menukar “two” dan “one”:

```
two one three
```

Tetapi jika terdapat karakter bukan kata, shell akan menukar kata dengan kata sebelumnya sambil mempertahankan urutan dari karakter bukan kata. Hal ini cukup berguna untuk mengedit argumen dari sebuah perintah. Misalkan Anda membuat kesalahan, dan mencampurkan ekstensi berkas yang hendak Anda cari dari parameter *print*:

```
find ~/music -name '*.print' -ogg
```

Anda bisa memperbaiki hal ini dengan meletakkan kursor pada kata kedua yang salah, dalam kasus ini “ogg”, dan menukar dua kata tersebut. Hal ini akan memberikan hasil yang kita inginkan:

```
find ~/music -name '*.ogg' -print
```

Akhirnya, terdapat beberapa shortcut yang mengubah kapitalisasi dari kata. Shortcut Alt-u membuat semua karakter menjadi huruf kapital, dimulai dari posisi kursor aktual hingga akhir kata. Jadi, jika kita memiliki nama dengan

huruf kecil “alice”, membuatnya menjadi huruf besar dengan kursor berada pada “i” memberikan hasil “alICE”. Alt-l memiliki perilaku yang sama, tetapi mengubah huruf menjadi huruf kecil. Jadi menggunakan Alt-l pada “alICE” dengan kursor pada “I” akan mengubahnya menjadi “alice”. Alt-c hanya mengubah karakter dimana kursor berada, atau karakter kata selenjutnya yang ditemukan, menjadi huruf kapital. Sebagai contoh, menekan Alt-c dengan posisi kursor pada “a” dalam kata “alice” akan menghasilkan “Alice”.

Pengeditan baris

Level tertinggi yang bisa kita edit adalah baris itu sendiri. Tabel 7.7, “Berpindah per baris” mendaftar dua shortcut.

Tabel 7.7. Berpindah per baris

Kunci	Deskripsi
Ctrl-a	Berpindah ke awal baris.
Ctrl-e	Berpindah ke akhir baris.

Misalkan kursor berada di tengah-tengah sebuah baris:

```
find ~/music -name '*.ogg' -print
```

Menekan Ctrl-e sekali akan memindahkan kursor ke akhir baris:

```
find ~/music -name '*.ogg' -print
```

Menekan Ctrl-a akan mmemindahkan kursor ke awal baris:

```
find ~/music -name '*.ogg' -print
```

Anda bisa menghapus karakter per baris. Shortcutnya dapat dilihat pada Tabel 7.8, “Menghapus baris”. Shortcut ini bekerja seperti pergerakan, tetapi menghapus karakter yang ditemukan. Ctrl-k akan menghapus karakter dimana kursor itu berada, tetapi Ctrl-x Backspace tidak akan menghapusnya. Berpindah ke awal baris dengan Ctrl-a, diikuti dengan Ctrl-k, adalah untuk menghapus sebuah baris sepenuhnya.

Tabel 7.8. Menghapus baris

Kunci	Deskripsi
Ctrl-k	Menghapus semua karakter pada baris, dimulai dari posisi kursor.
Ctrl-x Backspace	Menghapus semua karakter pada baris hingga posisi aktual kursor.

7.4. Catatan sejarah perintah

Seringkali Anda harus menjalankan perintah yang pernah Anda jalankan sebelumnya. Untungnya, Anda tidak harus mengetikkan semuanya kembali. Anda bisa melihat catatan sejarah dari perintah-perintah yang pernah ditulis dengan kunci panah atas dan bawah. Selain itu, juga dimungkinkan untuk mencari perintah. Menekan Control-r dan mulailah

menuliskan perintah yang hendak Anda tulis. Anda akan melihat bahwa bash akan menampilkan perintah yang cocok yang bisa ditemukan. Jika yang ditampilkan bukan yang Anda cari, Anda bisa bisa melanjutkan menuliskan perintah (sampai unik dan ditemukan sebuah kecocokan), atau menekan Control-r sekali lagi untuk mencari kembali. Ketika Anda sudah menemukan perintah yang dicari, Anda bisa mengeksekusinya dengan menekan <Enter>.

7.5. Pelengkap

Pelengkap adalah satu fungsi yang paling berguna dalam shell UNIX-like. Misalkan Anda memiliki direktori dengan dua berkas bernama `websites` dan `recipe`. Dan misalkan Anda hendak melakukan **cat** pada berkas `websites` (**cat** menampilkan isi sebuah berkas), dengan menentukan `websites` sebagai parameter ke perintah cat. Biasanya Anda cukup mengetikkan “cat websites”, dan mengeksekusi perintah. Cobalah untuk mengetikkan “cat w”, dan tekan kunci <Tab>. Bash akan secara otomatis melengkapi apa yang Anda tulis menjadi “cat websites”.

Tetapi apa yang terjadi jika Anda memiliki berkas yang namanya dimulai dengan karakter yang sama? Misalkan Anda memiliki berkas `recipe1.txt` dan `recipe2.txt`. Mengetikkan “cat r” dan menekan <Tab>, Bash akan melengkapi nama berkas sejauh yang bisa dilakukan. Hal ini akan memberikan “cat recipe”. Coba untuk menekan <Tab> lagi, dan Bash akan memberikan daftar nama berkas yang dimulai dengan “recipe”, dalam hal ini kedua berkas recipe. Pada posisi ini Anda harus membantu Bash mengetikkan karakter selanjutnya dari berkas yang Anda perlukan. Misalkan Anda hendak melakukan **cat** `recipe2.txt`, Anda bisa menekan kunci <2>. Setelah itu tidak ada masalah dalam melengkapi nama berkas, dan menekan <Tab> akan melengkapi perintah menjadi “cat recipe2.txt”.

Pelengkap ini bekerja pada perintah. Sebagian besar perintah GNU/Linux cukup pendek, sehingga tidak akan banyak berguna.

Merupakan ide yang bagus untuk berlatih dengan pelengkap, karena bisa menghemat banyak penekanan kunci jika Anda bisa memanfaatkannya. Anda bisa membuat beberapa berkas kosong dengan menggunakan perintah **touch**. Misalkan untuk membuat berkas dengan nama `recipe3.txt`, jalankan **touch recipe3.txt**.

7.6. Wildcard

Sebagian besar shell, termasuk Bash dan ksh, mendukung wildcard. Wildcard adalah karakter spesial yang bisa digunakan untuk melakukan pencocokan pola. Tabel berikut menampilkan beberapa wildcard yang umum dipakai. Kita akan melihat pada beberapa contoh untuk memberikan ide tentang bagaimana wildcard bekerja.

Tabel 7.9. Wildcard Bash

Wildcard	Cocok dengan
*	Sebuah string dari karakter
?	Karakter tunggal
[]	Sebuah karakter dalam sebuah array karakter

Pencocokan string dari karakter

Seperti yang Anda lihat pada tabel diatas, karakter “*” cocok dengan string dari karakter. Sebagai contoh **.html* cocok dengan semua yang diakhiri dengan *.html*, *d*.html* cocok dengan semua yang diawali dengan *d* dan diakhiri dengan *.html*.

Misalkan Anda hendak menampilkan semua berkas pada direktori aktual dengan ekstensi *.html*, perintah berikut akan melakukan pekerjaan tersebut:

```
$ ls *.html
book.html        installation.html     pkgmgmt.html  usermgmt.html
filesystem.html  internet.html         printer.html  xfree86.html
gfdl.html        introduction.html     proc.html
help.html        slackware-basics.html shell.html
```

Kita juga bisa menghapus semua berkas diawali dengan *in*:

```
$ rm in*
```

Pencocokan satu karakter

Wildcard "?" bekerja seperti "*", tetapi hanya cocok untuk satu karakter saja. Misalkan kita memiliki tiga berkas, `file1.txt`, `file2.txt` dan `file3.txt`. String *file?.txt* cocok dengan semua berkas ini, tetapi tidak cocok dengan `file10.txt` ("10" adalah dua karakter).

Pencocokan karakter dari sekumpulan

Wildcard "[]" cocok dengan semua karakter diantara kurung siku. Misalkan kita memiliki berkas dari contoh sebelumnya, `file1.txt`, `file2.txt` dan `file3.txt`. String *file[23].txt* cocok dengan `file2.txt` dan `file3.txt`, tetapi tidak untuk `file1.txt`.

7.7. Pengalihan dan pipe

Salah satu fitur utama dari shell UNIX-like adalah pengalihan dan pipe. Sebelum kita mulai melihat kedua teknik, kita harus melihat bagaimana perintah UNIX-like bekerja. Ketika sebuah perintah tidak mendapatkan data dari sebuah berkas, maka akan membuka sebuah berkas-pseudo khusus bernama *stdin*, dan menunggu data yang muncul. Prinsip yang sama bisa diterapkan pada hasil keluaran sebuah perintah, ketika tidak ada alasan khusus untuk menyimpan hasil keluaran ke sebuah berkas, berkas-pseudo *stdout* akan dibuka untuk keluaran data. Prinsip ini ditampilkan secara sistematis pada Gambar 7.1, "Masukan dan keluaran standar"

Gambar 7.1. Masukan dan keluaran standar

Anda bisa melihat *stdin* dan *stdout* bekerja dengan perintah **cat**. Jika cat dimulai tanpa sembarang parameter, maka dia akan menunggu masukkan dari *stdin* dan menampilkan data yang sama pada *stdout*. Jika tidak ada pengalihan yang digunakan, masukan dari keyboard akan digunakan untuk *stdin*, dan keluaran *stdout* akan dicetak ke terminal:

```
$ cat
Hello world!
Hello world!
```

Seperti yang Anda lihat, cat akan mencetak data ke *stdout* setelah memasukkan data ke *stdin* menggunakan keyboard.

Pengalihan

Shell mengijinkan Anda untuk menggunakan *stdin* dan *stdout* menggunakan "<" dan ">". Data dialihkan ke arah anak panah. Pada contoh berikut kita akan mengalihkan hasil md5 yang dihitung untuk sekumpulan berkas pada sebuah berkas bernama `md5sums`:

```
$ md5sum * > md5sums
$ cat md5sums
6be249ef5cacb10014740f61793734a8  test1
220d2cc4d5d5fed2aa52f0f48da38ebe  test2
631172a1cfca3c7cf9e8d0a16e6e8cfe  test3
```

Seperti yang bisa kita lihat pada keluaran perintah **cat**, hasil keluaran dari perintah **md5sum *** dialihkan ke berkas `md5sums`. Kita juga bisa menggunakan pengalihan untuk menyediakan masukkan ke sebuah perintah:

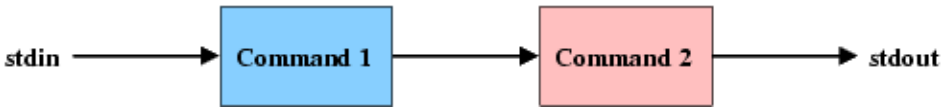
```
$ md5sum < test1
6be249ef5cacb10014740f61793734a8  -
```

Hal ini memberikan isi dari `test1` ke **md5sum**.

Pipe

Anda juga bisa menghubungkan masukan dan keluaran dari perintah menggunakan *pipe*. Sebuah pipe dapat dibuat dengan karakter "|". Dua atau lebih perintah yang digabungkan disebut *pipeline*. Gambar 7.2, "Sebuah pipeline" menampilkan sistematis bagaimana sebuah pipeline yang terdiri dari dua perintah.

Gambar 7.2. Sebuah pipeline



"Sintaks" dari pipeline adalah: **perintah1 | perintah2 ... | perintahN**. Jika Anda tahu bagaimana sebagian besar perintah UNIX bekerja, Anda bisa menggabungkannya. Mari kita lihat dalam sebuah contoh:

```
$ cat /usr/share/dict/american-english | grep "aba" | wc -l
123
```

Perintah pertama, **cat**, membaca berkas kamus `/usr/share/dict/american-english`. Hasil keluaran dari perintah **cat** dikirimkan melalui pipe ke **grep**, yang akan mencetak semua berkas yang mengandung frase "aba". Berikutnya, keluaran dari "grep" akan dikirimkan melalui pipe ke **wc -l**, yang akan menghitung jumlah baris yang diterima dari *stdin*. Akhirnya, ketika aliran sudah selesai, **wc** mencetak jumlah baris yang dihitung. Jadi ketiga perintah yang digabung akan menghitung jumlah baris yang mengandung frase "aba" pada kamus ini.

Terdapat banyak sekali utilitas kecil yang menangani tugas-tugas yang spesifik. Seperti yang Anda bayangkan, perintah-perintah ini bisa menyediakan sebuah kotak kerja yang hebat dengan membuat kombinasi dengan pipe.

Bab 8. Berkas dan direktori

8.1. Beberapa teori

Sebelum kita melihat operasi sistem berkas secara praktis, kita akan melihat secara teoritis bagaimana gambaran sistem berkas pada sistem UNIX-like bekerja. Slackware Linux mendukung banyak sistem berkas yang berbeda, tetapi semua sistem berkas ini menggunakan semantik yang sama secara virtual. Semantik ini disediakan melalui lapisan (layer) *Virtual Filesystem* (VFS), menyediakan layer geneik untuk sistem berkas disk dan jaringan.

inode, direktori dan data

Sistem berkas terdiri dari dua jenis elemen: data dan metadata. Metadata menjelaskan blok data yang sesungguhnya ada pada disk. Sebagian besar sistem berkas menggunakan node informasi (inode) untuk untuk menyimpan metadata. Sebagian besar sistem berkas menyimpan data berikut pada inodenya:

Tabel 8.1. Kolom inode umum

Kolom	Deskripsi
mode	Hak akses berkas.
uid	ID pengguna dari pemilik berkas.
gid	ID grup dari grup pemilik berkas.
size	Ukuran berkas dalam byte.
ctime	Waktu pembuatan berkas.
mtime	Waktu modifikasi terakhir berkas.
links_count	Jumlah link yang mengarah ke inode ini.
i_block	Penunjuk ke blok data

Jika Anda bukan pengguna UNIX atau Linux, nama-nama ini mungkin tampak asing bagi Anda, tetapi akan membuatnya jelas pada bagian berikutnya. Pada titik ini, Anda bisa menebak relasi antara inode dan data dari tabel ini, dan terutama kolom *i_block*: setiap inode memiliki penunjuk (pointer) ke blok data yang diberikan oleh inode. Bersama-sama, inode dan blok data adalah berkas aktual pada sistem berkas.

Anda mungkin terheran-heran dinama nama berkas (dan direktori) berada, karena tidak ada kolom nama berkas pada inode. Sebenarnya, nama berkas terpisah dari inode dan blok data, yang mengijinkan Anda melakukan hal-hal aneh seperti memberikan nama lebih dari satu pada berkas yang sama. Nama berkas disimpan dalam apa yang disebut isian direktori. Isian (entry) ini menentukan nama berkas dan inode dari berkas. Karena direktori juga direpresentasikan oleh inode, sebuah struktur direktori juga bisa dibuat dengan cara ini.

Kita bisa melihat bagaimana semua ini bekerja dengan mengilustrasikan apa yang dilakukan kernel ketika kita menjalankan perintah **cat /home/daniel/note.txt**

1. Sistem membaca inode dari direktori `/`, memeriksa apakah pengguna diijinkan untuk mengakses inode ini, dan membaca blok data untuk menemukan nomor inode dari direktori `home`.

2. Sistem membaca inode dari direktori `home`, memeriksa apakah pengguna diijinkan untuk mengakses inode ini, dan membaca blok data untuk menemukan nomor inode dari direktori `daniel`.

3. Sistem membaca inode dari direktori `daniel`, memeriksa apakah pengguna diijinkan untuk mengakses inode ini, dan membaca blok data untuk menemukan nomor inode dari berkas `note.txt`.

4. Sistem membaca inode dari berkas `note.txt`, memeriksa apakah pengguna diijinkan untuk mengakses inode ini, dan mengembalikan blok data pada **cat** melalui pemanggilan sistem (system call)*read()*.

Hak akses berkas

Seperti yang dijelaskan sebelumnya, Linux adalah sistem multi-user. Hal ini berarti bahwa setiap pengguna memiliki berkasnya sendiri (yang biasanya terdapat pada direktori home). Selain itu, pengguna juga bisa merupakan anggota dari sebuah grup, yang mungkin memberikan hak akses tambahan pada pengguna.

Seperti yang Anda lihat pada tabel kolom inode, setiap berkas memiliki pemilik dan grup. Kontrol hak akses UNIX tradisional memberikan hak akses read (baca), write (tulis), atau executable (eksekusi) pada pemilik berkas, group berkas, dan pengguna lain. Hak akses ini disimpan pada kolom *mode* dari inode. Kolom mode merepresentasikan hak akses berkas sebagai nomor oktal empat digit. Digit pertama merepresentasikan opsi khusus, digit kedua menyimpan hak akses pemilik, digit ketiga menyimpan hak akses grup, dan digit keempat untuk pengguna lain. Hak akses dibuat dengan nomor dengan menggunakan atau menambahkan salah satu angka pada Tabel 8.2, “Arti angka pada oktet model”

Tabel 8.2. Arti angka pada oktet model

Angka	Arti
1	Execute (x)
2	Write (w)
4	Read (r)

Sekarang, misalkan sebuah berkas memiliki mode *0644*, hal ini berarti berkas dapat dibaca dan ditulis oleh pemilik (*6*), dan dapat dibaca oleh group berkas (*4*) dan yang lain (*4*).

Sebagian besar pengguna tidak mau berhubungan dengan nomor oktal, sehingga banyak utilitas dapat bekerja dengan representasi alfabetik dari hak akses berkas. Huruf-huruf tersebut dapat dilihat pada Tabel 8.2, “Arti angka pada oktet model” didalam kurung digunakan pada notasi ini. Pada contoh berikut, informasi tentang berkas dengan hak akses *0644* akan dicetak. Nomor digantikan dengan kombinasi tiga rwx (karakter pertama bisa mendata opsi mode khusus).

```
$ ls -l note.txt
-rw-r--r--  1 daniel daniel 5 Aug 28 19:39 note.txt
```

Selama bertahun-tahun, hak akses UNIX tradisional sudah terbukti tidaklah cukup untuk beberapa kasus. Spesifikasi POSIX 1003.1e bertujuan untuk meningkatkan model kontrol akses UNIX dengan *Access Control Lists* (ACL). Sayangnya usaha ini terhenti, meskipun beberapa sistem (seperti GNU/Linux) sudah mengimplementasikan ACL[1]. Access control lists menggunakan semantik yang sama dengan hak akses berkas biasa, tetapi memberikan Anda kesempatan untuk menambahkan tambahan kombinasi *rwx* untuk pengguna dan grup tambahan.

Contoh berikut menggambarkan access control list dari sebuah berkas. Seperti yang Anda lihat, hak akses seperti hak akses UNIX normal (hak akses untuk pengguna, grup, dan lainnya sudah ditentukan). Tetapi terdapat sebuah tambahan untuk pengguna *joe*.

```
user::rwx
user:joe:r--
```

[1]Pada saat penulisan ini, ACL sudah didukung pada sistem berkas ext2, ext3, dan XFS

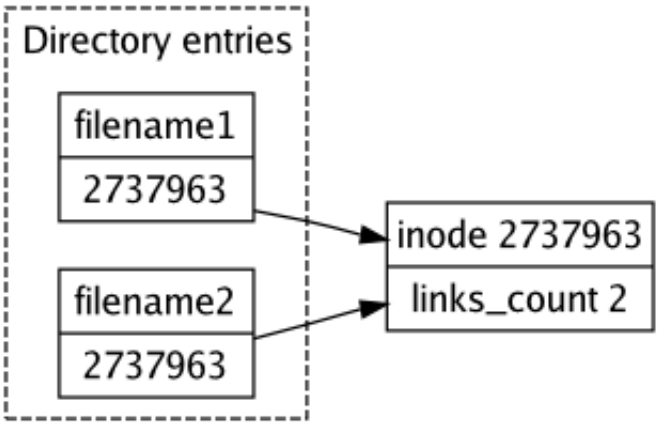
```
group::---
mask::r--
other::---
```

Untuk membuatnya menjadi lebih kompleks (dan canggih), beberapa sistem GNU/Linux menambahkan kontrol akses yang lebih ketat dengan Mandatory Access Control Frameworks (MAC) seperti SELinux dan AppArmor. Tetapi kerangka kontrol akses ini diluar batasan dari buku ini.

Hubungan

Sebuah isian direktori yang menunjuk pada sebuah inode disebut *hard link*. Sebagian besar berkas hanya dihubungkan satu kali, tetapi tidak ada yang membatasi Anda untuk membuat hubungan lebih dari satu kali. Hal ini akan meningkatkan kolom *links_count* dari inode. Ini merupakan cara yang baik bagi sistem untuk melihat inode dan blok data mana yang bebas untuk digunakan. Jika links_count diberi nilai kosong, maka inode tidak dirujuk oleh siapapun, dan bisa dipakai.

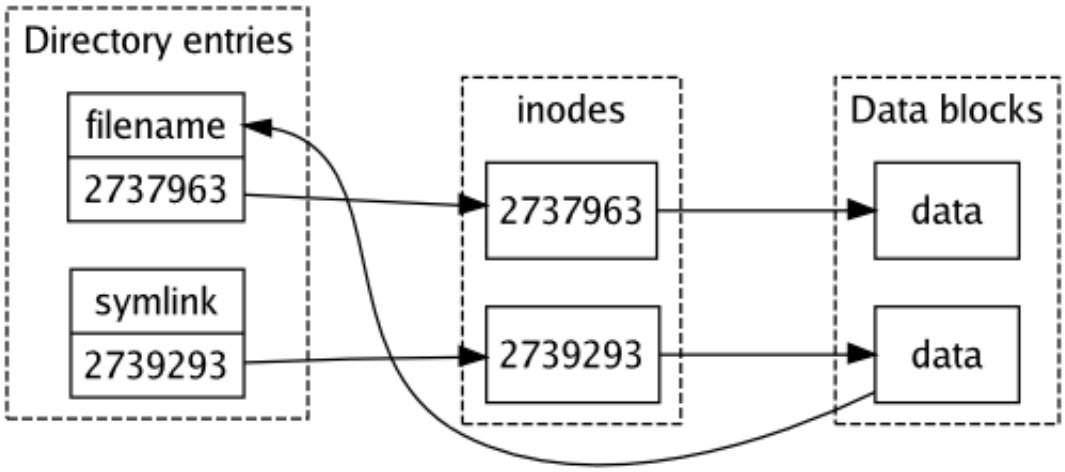
Gambar 8.1. Struktur dari hard link



Hard link memiliki dua keterbatasan. Pertama, hard link tidak bisa dihubungkan antar sistem berkas, karena mereka menunjuk pada inode. Setiap sistem berkas memiliki inode dan berhubungan dengan nomor inodenya masing-masing. Selain itu, sebagian besar sistem berkas tidak mengijinkan Anda untuk membuat hard link pada direktori. Mengijinkan pembuatan hard link pada direktori bisa menciptakan perulangan direktori, yang berpeluang menimbulkan deadlock dan masalah konsistensi pada sistem berkas. Selain itu, sebagian besar implementasi dari **rm** dan **rmdir** tidak tahu bagaimana menyelesaikan direktori tambahan hard link tersebut.

Symbolic link tidak memiliki keterbatasan ini, karena mereka menunjuk ke nama berkas, bukan ke inode. Ketika symbolic link digunakan, sistem operasi akan mengikuti path ke link tersebut. Symbolic links juga bisa menunjuk ke berkas yang tidak lagi ada, karena hanya berisi nama. Link semacam itu disebut dengan dangling link (link teruntai).

Gambar 8.2. Struktur dari symbolic link



Catatan

Jika Anda sampai ke administrasi sistem, sangatlah baik untuk sadar terhadap implikasi keamanan dari hard link. Jika direktori `/home` berada dalam sistem berkas yang sama dengan sembarang berkas biner sistem, seorang pengguna bisa membuat hard links ke berkas-berkas biner. Dalam kasus sebuah program yang cacat diupgrade, link pada direktori home pengguna akan tetap menunjuk pada program biner yang lama, yang memberikan pengguna akses ke program yang cacat.

Untuk alasan ini, maka sangat bagus untuk meletakkan sembarang direktori yang bisa ditulis oleh pengguna pada sistem berkas yang berbeda. Dalam implementasinya, hal ini berarti paling tidak meletakkan `/home` dan `/tmp` pada sistem berkas yang berbeda.

8.2. Menganalisa Berkas

Sebelum melanjutkan ke topik yang lebih menarik, kita akan mulai dengan beberapa penggunaan berkas dan direktori dasar.

Menampilkan berkas

Salah satu hal yang biasa Anda lakukan adalah menampilkan semua atau beberapa berkas. Perintah **ls** memungkinkan hal ini. Dengan menggunakan **ls** tanpa argumen apapun akan menampilkan isi dari direktori aktual:

```
$ ls
dns.txt  network-hosts.txt  papers
```

Jika Anda menggunakan distribusi GNU/Linux, Anda mungkin akan melihat pewarnaan yang menarik berdasarkan jenis berkas. Hasil keluaran standar sudah cukup untuk melihat secara sekilas tentang isi sebuah direktori, tetapi jika Anda menginginkan informasi lebih detail, Anda bisa menggunakan parameter `-l`. Parameter ini memberikan daftar informasi yang lebih lengkap untuk setiap berkas:

```
$ ls -l
total 36
-rw-rw-r--  1 daniel daniel 12235 Sep  4 15:56 dns.txt
-rw-rw-r--  1 daniel daniel  7295 Sep  4 15:56 network-hosts.txt
drwxrwxr-x  2 daniel daniel  4096 Sep  4 15:55 papers
```

Perintah ini memberikan informasi lebih banyak tentang tiga direktori yang ditemukan dengan **ls**. Kolom pertama menunjukan hak akses baris. Baris yang menunjukan `papers` dimulai dengan “d”, yang berarti sebuah direktori. Kolom kedua menampilkan jumlah hard link yang mengacu ke inode yang dituju oleh isi direktori tersebut. Jika lebih dari satu, maka ada nama berkas yang berbeda untuk satu berkas yang sama. Isi direktori biasanya memiliki paling tidak dua hard link, link pada direktori induk dan link pada direktori itu sendiri (setiap direktori memiliki `.`, yang mereferensikan ke direktori itu sendiri). Kolom ketiga dan keempat menampilkan pemilik dan grup berkas. Kolom kelima berisi ukuran berkas dalam byte. Kolom keenam menampilkan waktu dan jam modifikasi terakhir dari berkas. Dan akhirnya, kolom terakhir menampilkan nama.

Berkas yang diawali dengan tanda titik (.) tidak akan ditampilkan oleh sebagian besar aplikasi, termasuk **ls**. Anda bisa menampilkan berkas-berkas ini, dengan menambahkan opsi *`-a`* kepada **ls**:

```
$ ls -la
total 60
drwxrwxr-x   3 daniel daniel  4096 Sep 11 10:01 .
drwx------  88 daniel daniel  4096 Sep 11 10:01 ..
-rw-rw-r--   1 daniel daniel 12235 Sep  4 15:56 dns.txt
-rw-rw-r--   1 daniel daniel  7295 Sep  4 15:56 network-hosts.txt
drwxrwxr-x   2 daniel daniel  4096 Sep  4 15:55 papers
-rw-rw-r--   1 daniel daniel     5 Sep 11 10:01 .settings
```

Seperti yang Anda lihat, terdapat tambahan tiga data baru. Pertama-tama, berkas `.settings` sekarang ditampilkan. Selain itu, Anda bisa meilhat dua data direktori tambahan, `.` dan `..` Ini merepresentasikan direktori aktual dan direktori induk.

Pada awal bab ini (bagian bernama “inode, direktori dan data”) kita membicarakan tentang inode. Nomor inode yang ditunjuk bisa ditampilkan dengan parameter *`-i`*. Misalkan saya telah membuat sebuah hard link ke inode yang menunjuk ke inode yang sama dengan `dns.txt`, maka mereka seharusnya memiliki nomor inode yang sama. Hasil keluaran dari perintah **ls** berikut membuktikan bahwa hal ini benar:

```
$ ls -i dns*
3162388 dns-newhardlink.txt
3162388 dns.txt
```

Menentukan jenis berkas

Seringkali Anda membutuhkan bantuan untuk menentukan jenis sebuah berkas. Ini adalah kondisi dimana utilitas **file** menjadi penting. Misalkan saya menemukan sebuah berkas bernama `HelloWorld.class` pada disk saya. Saya berpendapat bahwa berkas ini berisi Java bytecode, tetapi kita bisa menggunakan **file** untuk memeriksanya:

```
$ file HelloWorld.class
HelloWorld.class: compiled Java class data, version 49.0
```

Berkas ini jelas Java bytecode. **file** cukup cerdas, dan menangani sebagian besar apapun yang Anda kirimkan. Sebagai contoh, Anda bisa memintanya memberikan informasi tentang sebuah node device:

```
$ file /dev/zero
/dev/zero: character special (1/5)
```

Atau symbolic link:

```
$ file /usr/X11R6/bin/X
/usr/X11R6/bin/X: symbolic link to `Xorg'
```

Jika Anda tertarik dengan berkas `/usr/X11R6/bin/X`, Anda bisa menggunakan opsi `-L` dari **file**:

```
$ file -L /usr/X11R6/bin/X
/usr/X11R6/bin/X: setuid writable, executable, regular file, no read permission
```

Anda mungkin heran kenapa **file** bisa menentukan jenis berkas dengan mudah. Sebagian besar berkas dimulai dengan apa yang disebut dengan *magic number*, yaitu angka unik yang memberitahukan kepada program yang bisa membaca berkas apa jenis berkas tersebut. Program **file** menggunakan sebuah berkas yang menjelaskan banyak jenis berkas dan magic numbernya. Sebagai contoh, magic file pada sistem saya berisi baris-baris berikut untuk berkas class Java yang sudah dicompile:

```
# Java ByteCode
# From Larry Schwimmer (schwim@cs.stanford.edu)
0       belong          0xcafebabe      compiled Java class data,
>6      beshort x       version %d.
>4      beshort x       \b%d
```

Data ini menunjukan bahwa jika berkas diawali dengan magic number heksadesimal dengan tipe data long (32-bit) *0xcafebabe*[2], adalah berkas yang berisi “data class Java yang terkompilasi”. Teks sederhana yang mengikutinya menentukan versi format berkas class.

Integritas berkas

Sebelum kita melihat pada pengujian integritas berkas yang lebih canggih, kita akan melihat pada utilitas **cksum**. **cksum** bisa mengkalkulasi cyclic redundancy check (CRC) dari sebuah berkas input. Ini merupakan metode matematis untuk mengkalkulasi nomor unik untuk sebuah berkas. Anda bisa menggunakan nomor ini untuk menguji apakah sebuah berkas sudah berubah atau belum (misalnya setelah mendownload berkas dari server). Anda bisa menentukan berkas yang akan dihitung sebagai parameter kepada **cksum**, dan **cksum** akan menghitung CRC, ukuran berkas dalam byte, dan nama berkas:

```
$ cksum myfile
1817811752 22638 myfile
```

[2]Ya, Anda bisa kreatif dengan magic number tersebut!

Slackware Linux juga menyediakan utilitas untuk mengkalkulasi checksum berdasarkan hash satu arah (misalnya MD5 atau SHA-1).

Melihat berkas

Karena sebagian besar berkas pada sistem UNIX adalah berkas teks, mereka sangatlah mudah untuk dilihat dari terminal berbasis karakter atau emulasi terminal. Cara primitif untuk melihat isi sebuah berkas dengan menggunakan **cat**. **cat** memberkas berkas yang ditentukan sebagai parameter per baris, dan menampilkan baris pada standard output. Sehingga, Anda bisa menulis isi dari berkas `note.txt` ke terminal dengan **cat note.txt**. Meskipun beberapa sistem dan sebagian besar emulasi terminal menyediakan dukungan untuk penggeseran (scrolling), ini bukanlah cara terbaik untuk melihat berkas yang besar. Anda bisa mengirimkan keluaran perintah **cat** ke **less** menggunakan pipe:

```
$ cat note.txt | less
```

atau biarkan **less** membaca berkas secara langsung:

```
$ less note.txt
```

Perintah **less** membuat Anda mampu menggeser maju dan mundur dari sebuah berkas. Tabel 8.3, “kunci perintah less” menyediakan gambaran tentang kunci penting yang biasa digunakan untuk mengendalikan **less**

Tabel 8.3. kunci perintah less

Kunci	Deskripsi
j	Geser maju satu baris.
k	Geser mundur satu baris.
f	Geser maju satu layar.
b	Geser mundur satu layar.
q	Keluar dari less.
g	Lompat ke awal berkas.
G	Lompat ke akhir berkas.
/*pola*	Mencari *pola* regular expression [#chap-textproc-regexps].
n	Mencari kecocokan berikutnya dari regular expression sebelumnya.
m*huruf*	Menandai posisi aktual pada berkas dengan *huruf*.
'*huruf*	Lompat ke tanda *huruf*

Kunci perintah yang bisa diakuantifikasi bisa diawali dengan angka. Sebagai contoh *11j* akan menggeser sebanyak sebelas baris, dan *3n* mencari kecocokan ketiga dari regular expression yang sudah ditentukan sebelumnya.

Slackware Linux juga menyediakan alternatif bagi **less**, perintah “more”. Kita tidak akan membahas *more* disini, **less** lebih nyaman, dan juga lebih populer.

Ukuran berkas dan direktori

Hasil keluaran **ls -l** yang kita lihat sebelumnya menghasilkan informasi tentang ukuran berkas. Meskipun menyediakan informasi yang cukup tentang ukuran sebuah berkas, Anda mungkin hendak mengumpulkan informasi tentang sekumpulkan berkas atau direktori. Ini adalah kondisi dimana perintah **du** dipakai. Secara default, **du** mencetak ukuran berkas per direktori. Misalnya:

```
$ du ~/qconcord
72      /home/daniel/qconcord/src
24      /home/daniel/qconcord/ui
132     /home/daniel/qconcord
```

Secara default, **du** merepresentasikan ukuran dalam unit 1024 byte. Anda bisa secara eksplisit menentukan bahwa **du** harus menggunakan unit 1024 byte dengan menambahkan *-k*. Hal ini berguna saat menuliskan script, karena sistem lain memiliki nilai default untuk menggunakan blok 512-byte. Sebagai contoh:

```
$ du -k ~/qconcord
72 /home/daniel/qconcord/src
24 /home/daniel/qconcord/ui
132 /home/daniel/qconcord
```

Jika Anda hendak melihat penggunaan disk per berkas, Anda bisa menambahkan *-a*:

```
$ du -k -a ~/qconcord
8       /home/daniel/qconcord/ChangeLog
8       /home/daniel/qconcord/src/concordanceform.h
8       /home/daniel/qconcord/src/textfile.cpp
12      /home/daniel/qconcord/src/concordancemainwindow.cpp
12      /home/daniel/qconcord/src/concordanceform.cpp
8       /home/daniel/qconcord/src/concordancemainwindow.h
8       /home/daniel/qconcord/src/main.cpp
8       /home/daniel/qconcord/src/textfile.h
72      /home/daniel/qconcord/src
12      /home/daniel/qconcord/Makefile
16      /home/daniel/qconcord/ui/concordanceformbase.ui
24      /home/daniel/qconcord/ui
8       /home/daniel/qconcord/qconcord.pro
132     /home/daniel/qconcord
```

Anda juga bisa menggunakan nama berkas atau wildcard sebagai parameter. Tetapi hal ini tidak akan mencetak ukuran berkas dalam sub direktori, kecuali *-a* dipakai:

```
$ du -k -a ~/qconcord/*
8       /home/daniel/qconcord/ChangeLog
12      /home/daniel/qconcord/Makefile
8       /home/daniel/qconcord/qconcord.pro
8       /home/daniel/qconcord/src/concordanceform.h
8       /home/daniel/qconcord/src/textfile.cpp
```

```
12      /home/daniel/qconcord/src/concordancemainwindow.cpp
12      /home/daniel/qconcord/src/concordanceform.cpp
8       /home/daniel/qconcord/src/concordancemainwindow.h
8       /home/daniel/qconcord/src/main.cpp
8       /home/daniel/qconcord/src/textfile.h
72      /home/daniel/qconcord/src
16      /home/daniel/qconcord/ui/concordanceformbase.ui
24      /home/daniel/qconcord/ui
```

Jika Anda hendak melihat total ukuran penggunaan disk untuk berkas dan sub direktori yang ada didalam sebuah direktori, gunakan `-s`:

```
$ du -k -s ~/qconcord
132     /home/daniel/qconcord
```

8.3. Bekerja dengan direktori

Setelah melihat gambaran tentang direktori pada bagian bernama “inode, direktori dan data”, kita akan melihat pada beberapa perintah yang berhubungan dengan direktori.

Menampilkan direktori

Perintah **ls** yang kita lihat pada bagian bernama “Menampilkan berkas” juga bisa digunakan untuk menampilkan direktori dalam berbagai cara. Seperti yang sudah kita lihat, hasil keluaran dari **ls** meliputi direktori, dan direktori bisa diidentifikasi dengan kolom pertama pada daftar yang lengkap:

```
$ ls -l
total 36
-rw-rw-r--  1 daniel daniel 12235 Sep  4 15:56 dns.txt
-rw-rw-r--  1 daniel daniel  7295 Sep  4 15:56 network-hosts.txt
drwxrwxr-x  2 daniel daniel  4096 Sep  4 15:55 papers
```

Jika nama direktori, atau jika wildcard ditentukan **ls** akan menampilkan isi dari direktori atau direktori yang sesuai dengan wildcard. Sebagai contoh, jika terdapat direktori `papers`, **ls paper*** akan menampilkan isi dari direktori `paper`. Hal ini cukup menganggu jika Anda hanya ingin lihat yang sesuai, dan bukan isi dari direktori. Parameter `-d` menghindari rekursi ini:

```
$ ls -ld paper*
drwxrwxr-x  2 daniel daniel  4096 Sep  4 15:55 papers
```

Anda juga bisa melihat isi dari sebuah direktori dan sub direktori secara rekursif dengan parameter `-R`:

```
$ ls -R
.:
dns.txt  network-hosts.txt  papers
```

```
./papers:
cs  phil

./papers/cs:
entr.pdf

./papers/phil:
logics.pdf
```

Membuat dan menghapus direktori

UNIX menyediakan perintah **mkdir** untuk membuat direktori. Jika path relatif diberikan, direktori akan dibuat pada direktori aktif. Sintaks dasarnya sangatlah sederhana: *mkdir <nama>*, sebagai contoh:

```
$ mkdir mydir
```

Secara default, **mkdir** hanya membuat satu level direktori. Jadi, jika Anda menggunakan **mkdir** untuk membuat `mydir/mysubdir`, **mkdir** akan gagal jika `mydir` belum ada sebelumnya. Jika Anda ingin membuat kedua direktori sekaligus, gunakan parameter *-p* :

```
$ mkdir -p mydir/mysubdir
```

rmdir menghapu sebuah direktori. Perilakunya sama dengan **mkdir**. **rmdir mydir/mysubdir** menghapus `mydir/subdir`, sementara **rmdir -p mydir/mysubdir** menghapus `mydir/mysubdir` lalu `mydir`.

Jika sebuah sub direktori yang akan kita hapus berisi isi direktori lain, **rmdir** akan gagal. Jika Anda ingin menghapus sebuah direktori, termasuk semua isinya, gunakan perintah **rm**.

8.4. Mengelola berkas dan direktori

Menyalin

Berkas dan direktori bisa disalin dengan perintah **cp**. Pada sintaks dasarnya, berkas sumber dan target akan ditentukan. Contoh berikut akan membuat salinan dari berkas `file1` dengan nama `file2`:

```
$ cp file1 file2
```

Tidaklah mengherankan jika path relatif dan absolut juga bekerja:

```
$ cp file1 somedir/file2
$ cp file1 /home/joe/design_documents/file2
```

Anda juga bisa menentukan direktori sebagai parameter kedua. Jika terdapat kasus ini, **cp** akan membuat salinan berkas tersebut pada direktori tersebut, memberikan nama berkas yang sama dengan berkas asli. Jika terdapat lebih dari satu parameter, parameter akan digunakan sebagai direktori target. Sebagai contoh:

```
$ cp file1 file2 somedir
```

akan menyalin `file1` dan `file2` ke direktori `somedir`. Anda tidak bisa menyalin banyak berkas ke satu berkas. Anda harus menggunakan **cat**:

```
$ cat file1 file2 > combined_file
```

Anda juga bisa menggunakan **cp** untuk menyalin direktori, dengan menambahkan *-R*. Hal ini akan menyalin sebuah direktori dan semua sub direktorinya secara rekursif. Jika direktori target sudah ada, sumber direktori atau direktori akan diletakkan dibawah direktori target. Jika direktori target belum ada, maka akan diciptakan jika hanya terdapat satu direktori sumber.

```
$ cp -r mytree tree_copy
$ mkdir trees
$ cp -r mytree trees
```

Setelah mengeksekusi perintah diatas, akan terdapat dua salinan dari direktori `mytree`, `tree_copy` dan `trees/mytree`. Percobaan untuk menyalin dua direktori pada target yang tidak ada akan gagal:

```
$ cp -R mytree mytree2 newdir
usage: cp [-R [-H | -L | -P]] [-f | -i] [-pv] src target
       cp [-R [-H | -L | -P]] [-f | -i] [-pv] src1 ... srcN directory
```

Catatan

Parameter *-r* sudah ada pada banyak sistem UNIX untuk menyalin direktori secara rekursif. Namun, perilaku dari parameter ini bisa jadi tergantung dari implementasinya, dan Spesifikasi UNIX versi 3 menyatakan bahwa parameter ini bisa dihapus dari versi yang akan datang.

Ketika Anda menyalin berkas secara rekursif, merupakan ide bagus untuk menentukan perilaku dari **cp** ketika symbolic link ditemukan, jika Anda ingin menggunakan **cp** pada script yang bersifat portable. Spesifikasi UNIX versi 3 tidak menentukan bagaimana seharusnya.Jika *-P* digunakan, symbolic links akan tidak diikuti, sehingga hanya menyalin link itu sendiri. Jika *-H* digunakan, symbolic links yang ditentukan sebagai parameter bagi **cp** bisa diikuti, tergantung dari jenis dan isi dari berkas. Jika *-L* digunakan, symbolic links yang ditentukan sebagai parameter pada **cp** dan symbolic links yang ditemukan akan selama penyalinan akan diikuti, tergantung dari isi berkas.

Jika Anda ingin mempertahankan kepemilikan, bit SGID/SUID, dan waktu modifikasi dan akses sebuah berkas, Anda bisa menggunakan parameter *-p*. Parameter ini akan mencoba mempertahankan properti pada saat penyalinan berkas atau direktori. Implementasi yang bagus dari **cp** menyediakan beberapa proteksi tambahan - jika berkas target sudah ada, maka tidak akan ditimpa jika metadata yang berhubungan tidak bisa dipertahankan.

Memindahkan berkas dan direktori

Perintah UNIX untuk memindahkan berkas, **mv**, bisa memindahkan atau mengganti nama berkas atau direktori. Apa yang sebenarnya terjadi tergantung dari lokasi berkas atau direktori. Jika berkas sumber dan tujuan berada dalam sistem berkas yang sama, **mv** hanya membuat hard link baru, yang akan mengganti nama berkas atau direktori. Jika keduanya

berada dalam sistem berkas yang berbeda, berkas akan disalin, dan berkas atau direktori sumber akan diputuskan linknya.

Sintaks dari **mv** hampir serupa dengan **cp**. Sintaks dasar untuk mengganti nama `file1` menjadi `file2`:

```
$ mv file1 file2
```

Sintaks yang sama bisa digunakan untuk dua direktori., yang akan mengganti nama direktori yang diberikan pada parameter pertama menjadi nama yang diberikan pada parameter kedua.

Ketika parameter terakhir adalah direktori yang sudah ada, berkas atau direktori yang ditentukan pada parameter pertama, akan disalin pada direktori tersebut. Pada kasus ini, Anda bisa menentukan banyak berkas atau direktori. Sebagai contoh:

```
$ mkdir targetdir
$ mv file1 directory1 targetdir
```

Hal ini menciptakan direktori `targetdir`, dan memindahkan `file1` dan `directory1` ke direktori ini.

Menghapus berkas dan direktori

Berkas dan direktori bisa dihapus dengan perintah rm (1) . Perintah ini akan menghapus link berkas dan direktori. Jika tidak ada lagi link pada sebuah berkas, inode dan blok disk bisa dipakai untuk berkas yang lain. Berkas bisa dihapus dengan menyediakan berkas yang akan dihapus sebagai parameter pada rm (1) . Jika berkas tidak dapat ditulisi, rm (1) akan bertanya sebagai konfirmasi. Sebagai contoh, untuk menghapus `file1` dan `file2`, Anda bisa menjalankan:

```
$ rm file1 file2
```

Jika Anda harus menghapus banyak berkas yang membutuhkan konfirmasi sebelum bisa dihapus, atau jika Anda ingin menggunakan rm (1) untuk menghapus berkas dari script yang tidak akan dijalankan pada terminal, tambahkan parameter `-f` untuk menimpa penggunaan prompt. Berkas yang tidak bisa ditulis, akan dihapus dengan `-f` jika kepemilikan berkas mengijinkan hal ini. Parameter ini juga akan mengurangi hasil keluaran pesan kesalahan pada *stderr* jika misalnya berkas yang akan dihapus tidak ditemukan.

Direktori juga bisa dihapus secara rekursif dengan parameter `-r`. rm (1) akan menghapus struktur direktori, menghapus link dan direktori yang ditemukan. Semantik yang sama juga digunakan ketika berkas biasa dihapus, selama `-f` dipakai. Untuk memberikan contoh sederhana, Anda bisa menghapus semua berkas dan direktori pada direktori `notes` secara rekursif dengan :

```
$ rm -r notes
```

Karena perintah rm (1) menggunakan fungsi unlink (2) , blok data tidak ditulis ulang pada status yang tidak terinisialisasi. Informasi pada data blok hanya ditulis ulang jika dialokasikan dan digunakan pada waktu lain. Untuk menghapus berkas termasuk blok data secara aman, beberapa sistem menyediakan perintah shred (1) yang menimpa blok data dengan data acak. Tetapi hal ini tidak efektif pada kebanyakan sistem berkas modern (journaling), karena tidak menulis data pada tempatnya.

Perintah unlink (1) menyediakan implementasi dari fungsi unlink (2) . Sangat jarang digunakan, karena tidak bisa menghapus direktori.

8.5. Hak akses

Kita melihat hak akses berkas dan direktori pada bagian bernama “Hak akses berkas”. Pada bagian ini, kita akan melihat pada perintah chown (1) dan chmod (1) , yang digunakan untuk menentukan kepemilikan dan hak akses berkas. Setelah itu, kita akan melihat perluasan dari hak akses yang disebut Access Control Lists (ACL).

Mengganti kepemilikan berkas

Seperti yang sudah kita lihat sebelumnya, setiap berkas memiliki ID pemilik (user) dan ID grup yang disimpan pada inode. Perintah chown (1) bisa digunakan untuk menentukan kolom ini. Hal ini bisa dilakukan dengan ID numerik, atau namanya. Sebagai contoh, untuk mengganti kepemilikan dari berkas `note.txt` menjadi *john*, dan grupnya menjadi *staff*, digunakan perintah berikut:

```
$ chown john:staff note.txt
```

Anda bisa menghilangkan salah satu komponen, jika Anda ingin menentukan salah satunya. Jika Anda hanya ingin menentukan nama pengguna, Anda bisa mengabaikan titik dua. Jadi perintah diatas bisa dibagi menjadi dua langkah:

```
$ chown john note.txt
$ chown :staff note.txt
```

Jika Anda ingin mengganti kepemilikan dari direktori, dan semua berkas atau direktori dibawahnya, Anda bisa menambahkan *-R* pada chown (1) :

```
$ chown -R john:staff notes
```

Jika nama pengguna dan grup ditentukan dan bukan ID, nama akan dikonversi oleh chown (1) . Konvensi ini biasanya bergantung dari basis data kata sandi sistem. Jika Anda beroperasi pada sistem berkas yang menggunakan basis data kata sandi yang berbeda (contohnya Anda me-mount sistem berkas root dari sistem lain untuk proses perbaikan), akan sangat berguna untuk mengganti kepemilikan berkas dengan ID pengguna atau grup. Dengan cara ini, Anda bisa mempertahankan pemetaan nama pengguna/grup. Jadi, mengganti kepemilikan dari `note` menjadi UID 1000 dan GUID 1000 dilakukan dengan cara berikut:

```
$ chown 1000:1000 note.txt
```

Mengganti bit hak akses berkas

Setelah membaca perkenalan terhadap hak akses sistem berkas pada bagian bernama “Hak akses berkas”, mengganti bit hak akses yang disimpan pada inode menjadi mudah dengan perintah chmod (1) . chmod (1) menerima representasi numerik dan simbolik dari hak akses. Merepresentasikan hak akses dari berkas secara numerik sangatlah mudah, karena mengijinkan untuk menentukan semua hak akses yang relevan. Sebagai contoh:

```
$ chmod 0644 note.txt
```

Akan membuat `note.txt` dapat dibaca dan ditulis oleh pemilik berkas, dan dapat dibaca oleh grup dan lainnya.

Hak akses simbolik bekerja dengan penambahan atau mengurangan hak akses, dan mengijinkan perubahan relatif dari hak akses berkas. Sintaks untuk hak akses simbolik adalah :

```
[ugo][-+][rwxst]
```

Komponen pertama menentukan kelas pengguna dimana perubahan hak akses akan diterapkan (pemilik, grup atau lainnya). Banyak karakter pada komponen ini bisa dikombinasikan. Komponen kedua menghilangkan (-), atau menambahkan hak akses (+). Komponen ketiga adalah hak akses (read, write, execute, set UID/GID on execution, sticky). Banyak komponen bisa ditentukan pada untuk komponen ini. Mari kita lihat pada beberapa contoh untuk memperjelas:

```
ug+rw        # Memberikan hak akses baca/tulis pada pemilik dan grup berkas
chmod go-x   # Mengambil hak akses dari grup dan lainnya
chmod ugo-wx # Tidak mengijinkan semua kelas pengguna untuk menulis ke berkas dan mengeksekus
```

Perintah ini bisa digunakan seperti berikut:

```
$ chmod ug+rw note.txt
$ chmod go-x script1.sh
$ chmod ugo-x script2.sh
```

Hak akses dari berkas dan direktori bisa diganti secara rekursif dengan *-R*. Perintah berikut akan membuat direktori `notes` world-readable (bisa dibaca oleh semuanya), termasuk isinya:

```
$ chmod -R ugo+r notes
```

Perhatian tambahan harus dilakukan ketika bermain dengan direktori, karena *x* memiliki arti khusus dalam konteks direktori. Pengguna yang memiliki hak akses eksekusi pada direktori bisa mengakses sebuah direktori. Pengguna yang tidak memiliki hak akses eksekusi pada direktori tidak bisa mengakses direktori. Karena perilaku ini, seringkali lebih mudah untuk mengganti hak akses dari struktur direktori dan berkas-berkasnya dengan bantuan perintah find (1) .

Terdapat bit hak akses tambahan yang bisa ditentukan yang memiliki arti khusus. SUID dan SGID adalah yang paling menarik dari bit tambahan tersebut. Bit ini mengganti ID pengguna dan grup yang digunakan ketika berkas dieksekusi. Perintah **su(1)** adalah contoh berkas yang memiliki bit SUID yang aktif:

```
$ ls -l /bin/su
-rwsr-xr-x  1 root root 60772 Aug 13 12:26 /bin/su
```

Hal ini berarti bahwa perintah **su** berjalan sebagai pengguna *root* ketika dieksekusi. Bit SUID bisa ditentukan dengan modifier *s*. Sebagai contoh, jika bit SUID belum ditentukan pada `/bin/su` hal ini bisa dilakukan dengan :

```
$ chmod u+s /bin/su
```

Catatan

Harap waspada bahwa bit SUID dan SGID memiliki dampak terhadap keamanan. Jika sebuah program dengan bit ini memiliki sebuah kesalahan, maka bisa dieksploitasi untuk mendapatkan hak akses dari pemilik atau grup berkas. Karena alasan inilah, disarankan untuk meminimalkan penggunaan bit SUID dan SGID pada berkas.

Bit sticky juga menarik jika berhubungan dengan direktori. Bit ini tidak mengijinkan pengguna untuk mengganti nama berkas atau menghapus berkas yang tidak dimilikinya, pada direktori dimana mereka memiliki hak akses tulis. Hal ini biasanya digunakan pada direktori yang bersifat world-writeable (bisa ditulis oleh semua), seperti direktori temporal (`/tmp`) pada banyak sistem UNIX. Tag sticky bisa ditentukan dengan *t*:

```
$ chmod g+t /tmp
```

Mask pembuatan berkas

Pertanyaan yang tersisa adalah apa hak akses awal yang digunakan ketika sebuah file dibuat. Hal ini tergantung dari dua faktor: mode yang dikirimkan ke system call *open(2)*, yang digunakan untuk membuat sebuah berkas, dan mask pembuatan berkas yang aktif. Mask pembuatan berkas bisa direpresentasikan sebagai bilangan oktal. Hak akses yang efektif untuk membuat berkas ditentukan sebagai *mode & ~mask*. Atau, jika direpresentasikan dalam model oktal, Anda bisa mengurangkan digit dari mask dari mode. Sebagai contoh, jika sebuah berkas dibuat dengan hak akses *0666* (dapat dibaca dan ditulis oleh pengguna, grup, dan lainnya), dan mask pembuatan berkas efektif adalah *0022*, maka hak akses berkas yang efektif adalah *0644*. Mari kita lihat contoh lain. Misalkan berkas masih dibuat dengan hak akses *0666*, dan Anda lebih waspada, dan ingin menghapus semua hak akses baca dan tulis pada grup dan lainnya... Ini berarti Anda harus menentukan mask pembuatan berkas menjadi *0066*, karena mengurangi *0066* dari *0666* menghasilkan *0600*

Mask pembuatan berkas efektif dapat dilihat dan ditentukan dengan perintah **umask**, yang biasanya merupakan perintah bawaan shell. Mask bisa dicetak dengan menjalankan **umask** tanpa parameter:

```
$ umask
0002
```

Mask bisa ditentukan dengan memberikan nomor oktal sebagai parameter. Misalnya:

```
$ umask 0066
```

Kita bisa memverifikasi hasilnya dengan membuat sebuah berkas baru:

```
$ touch test
$ ls -l test
-rw-------  1 daniel daniel 0 Oct 24 00:10 test2
```

Access Control Lists

Access Control lists (ACLs) adalah pengembangan dari hak akses berkas UNIX tradisional, yang mengijinkan kontrol akses yang lebih ketat. Sebagian besar sistem yang mendukung sistem berkas ACL mengimplementasikannya

seperti yang ditentukan dalam draft spesifikasi POSIX.1e dan POSIX.2c. Sistem UNIX dan UNIX-like yang sudah mengimplementasikan ACL sesuai draft ini adalah FreeBSD, Solaris, and Linux.

Seperti yang kita lihat pada bagian bernama “Hak akses berkas” access control lists mengijinkan Anda untuk menggunakan triplet read, write dan execute untuk pengguna atau grup tambahan. Kebalikan dari hak akses berkas tradisional, access control lists tambahan tidak disimpan secara langsung pada node, tetapi pada attribut tambahan yang berhubungan dengan berkas. Dua hal yang perlu diperhatikan ketika Anda menggunakan access control lists adalah tidak semua sistem mendukungnya, dan tidak semua program mendukungnya.

Membaca access control lists

Pada sebagian besar sistem yang mendukung ACL, **ls** menggunakan indikator visual untuk menunjukan bahwa terdapat ACL yang berhubungan dengan berkas. Misalnya:

```
$ ls -l index.html
-rw-r-----+ 1 daniel daniel 3254 2006-10-31 17:11 index.html
```

Seperti yang Anda lihat, kolom hak akses menunjukan adanya karakter plus tambahan (+). Bit hak akses tidak memiliki perilaku seperti yang Anda harapkan. Kita akan lihat hal itu sebentar lagi.

ACL untuk sebuah berkas dapat dilihat dengan perintah **getfacl**:

```
$ getfacl index.html
# file: index.html
# owner: daniel
# group: daniel
user::rw-
group::---
group:www-data:r--
mask::r--
other::---
```

Sebagian besar berkas dapat diinterpretasikan dengan mudah: pengguna berkas memiliki hak akses baca/tulis, grop berkas tidak memiliki hak akses, pengguna dari grup *www-data* memiliki hak akses baca, dan lainnya tidak memiliki hak akses apapun. Tetapi kenapa menampilkan tidak ada hak akses untuk grup berkas, tetapi **ls** menampilkan? Rahasianya adalah jika ada *mask*, **ls** menampilkan nilai dari mask, dan bukan hak akses grup.

mask digunakan untuk membatasi semua catatan dengan pengecualian untuk pengguna berkas dan untuk pengguna lainnya. Hapalkan aturan berikut untuk menginterpretasikan ACL:

- Hak akses *user::* berhubungan dengan hak akses pemilik berkas.
- Hak akses *group::* berhubungan dengan grup berkas, kecuali jika terdapat *mask::*. Jika terdapat *mask::*, hak akses dari grup berhubungan dengan isi dari grup dengan nilai mask sebagai hak akses maksimal (berarti hak akses grup bisa lebih ketat).
- Hak akses untuk pengguna dan grup lain berhubungan dengan *user:* dan *group:*, dengan nilai *mask::* sebagai hak akses maksimalnya.

Aturan kedua dan ketiga bisa dengan jelas diamati jika terdapat sebuah pengguna atau grup yang memiliki hak akses yang lebih dari mask untuk berkas tersebut:

```
$ getfacl links.html
# file: links.html
# owner: daniel
# group: daniel
user::rw-
group::rw-                      #effective:r--
group:www-data:rw-              #effective:r--
mask::r--
other::---
```

Meskipun hak akses baca dan tulis ditentukan untuk berkas dan berkas dan grup *www-data*, kedua grup akan secara efektif hanya memiliki hak akses baca, karena ini merupakan hak akses maksimal yang diijinkan oleh mask.

Aspek lain yang harus diperhatikan adalah penanganan ACL untuk direktori. Access control list bisa ditambahkan ke direktori untuk mengendalikan akses, tetapi direktori juga memiliki *default ACL* yang menentukan ACL awal untuk berkas dan direktori yang dibuat pada direktori tersebut.

Misalkan direktori `reports` memiliki ACL berikut:

```
$ getfacl reports
# file: reports
# owner: daniel
# group: daniel
user::rwx
group::r-x
group:www-data:r-x
mask::r-x
other::---
default:user::rwx
default:group::r-x
default:group:www-data:r-x
default:mask::r-x
default:other::---
```

Berkas baru yang dibuat pada direktori `reports` memiliki ACL berdasarkan catatan yang memiliki awalan *default:*. Sebagai contoh:

```
$ touch reports/test
$ getfacl reports/test
# file: reports/test
# owner: daniel
# group: daniel
user::rw-
group::r-x                      #effective:r--
group:www-data:r-x              #effective:r--
mask::r--
other::---
```

Seperti yang Anda lihat, ACL default disalin. Bit eksekusi dihapus dari mask, karena berkas baru tidak dibuat dengan hak akses eksekusi.

Membuat access control list

ACL untuk sebuah berkas atau direktori bisa diganti dengan program **setfacl**. Sayangnya, penggunaan program ini sangat bergantung dari sistem yang digunakan. Untuk menambah kebingungan, paling tidak satu parameter (`-d`) memiliki arti yang berbeda pada sistem yang berbeda. Anda bisa berharap bahwa perintah ini akan mendapatkan standarisasi.

Tabel 8.4. Parameter setfacl spesifik

Operasi	Linux
Menentukan isi, menghapus semua isi lama	`--set`
Memodifikasi isi	`-m`
Memodifikasi isi ACL default	`-d`
Menghapus isi	`-x`
Menghapus semua isi ACL (kecuali tiga isi yang diperlukan).	`-b`
Menghitung ulang mask	Selalu dikalkulasi ulang, kecuali `-n` digunakan, atau isi mask ditentukan secara eksplisit.
Menggunakan spesifikasi ACL dari sebuah berkas	`-M` (memodifikasi), `-X` (menghapus), atau `--restore`
Modifikasi rekursif ACL	`-R`

Seperti yang sudah kita lihat pada bagian sebelumnya, isi bisa ditentukan untuk pengguna dan grup, dengan mengikuti sintaks: *user/group:name:permissions*. Hak akses bisa ditentukan sebagai triplet dengan menggunakan huruf *r* (read), *w* (write), atau *x* (execute). Karakter minus (-) digunakan untuk hak akses yang tidak ingin diberikan kepada pengguna atau grup, karena Solaris membutuhkan hal ini. Jika Anda menginginkan agar tidak ada hak akses sepenuhnya, Anda bisa menggunakan ---.

Spesifikasi untuk pengguna lain, dan mask mengikuti format berikut: *other:r-x*. Format yang lebih mudah ditebak juga bisa digunakan: *other::r-x*.

Memodifikasi isi ACL

Operasi paing sederhana adalah untuk memodifikasi sebuah isi ACL. Hal ini akan membuat sebuah isi baru jika isi belum ada sebelumnya. Isi bisa dimodifikasi dengan `-m`. Sebagai contoh, misalkan kita hendak memberikan grup *friend* akses baca dan tulis pada berkas `report.txt`. Hal ini bisa dilakukan dengan:

```
$ setfacl -m group:friends:rw- report.txt
```

Mask dari isian akan dikalkulasi ulang, melakukan setting menjadi gabungan dari semua isi grup, dan isi pengguna tambahan:

```
$ getfacl report.txt
# file: report.txt
# owner: daniel
# group: daniel
user::rw-
```

```
group::r--
group:friends:rw-
mask::rw-
other::r--
```

Anda bisa mengkombinasikan banyak isi ACL dengan memisahkannya dengan karakter koma. Sebagai contoh:

```
$ setfacl -m group:friends:rw-,group:foes:--- report.txt
```

Menghapus Isi ACL

Sebuah isi dapat dihapus dengan opsi *-x*:

```
$ setfacl -x group:friends: report.txt
```

Karakter titik dua di akhir bisa diabaikan.

Membuat ACL Baru

Opsi *--set* digunakan untuk membuat sebuah ACL baru untuk sebuah berkas, menghapus isi ACL yang ada, kecuali isi yang benar-benar diperlukan. Isi untuk file pengguna, grup, dan lainnya juga ditentukan. Sebagai contoh:

```
$ setfacl --set user::rw-,group::r--,other:---,group:friends:rwx report.txt
```

Jika Anda tidak ingin membersihkan hak akses pengguna, grup, dan lainnya, tetapi Anda hendak menghapus isi ACL lain, Anda bisa menggunakan opsi *-b*. Contoh berikut menggunakanna dengan kombinasi dari opsi *-m* untuk menghapus semua isi ACL (kecuali untuk pengguna, grup, dan lainnya), dan menambahkan sebuah isi baru untuk grup *friends*:

```
$ setfacl -b -m group:friends:rw- report.txt
```

Mensetting ACL default

Seperti yang kita lihat pada bagian bernama “Access Control Lists”, direktori bisa memiliki isi ACL default yang menentukan hak akses apa yang harus digunakan untuk berkas dan direktori yang dibuat pada direktori tersebut. Opsi *-d* digunakan untuk beroperasi pada isi default:

```
$ setfacl -d -m group:friends:rwx reports
$ getfacl reports
# file: reports
# owner: daniel
# group: daniel
user::rwx
group::r-x
other::r-x
default:user::rwx
```

```
default:group::r-x
default:group:friends:rwx
default:mask::rwx
default:other::r-x
```

Menggunakan ACL dari berkas referensi

Anda juga bisa menggunakan spesifikasi ACL dari berkas, dan bukan menentukannya dari perintah baris. Sebuah berkas masukan mengikuti sintaks yang sama dengan parameter pada **setfacl**, tetapi dipisahkan dengan baris baru dan bukan dengan koma. Hal ini berguna karena Anda bisa menggunakan ACL untuk berkas yang ada sebagai referensi:

```
$ getfacl report.txt > ref
```

Opsi *-M* disediakan untuk memodifikasi ACL untuk sebuah berkas dengan membaca isi dari berkas lain. Jadi, jika kita memiliki berkas bernama `report2.txt`, kita bisa memodifikasi ACL untuk berkas ini dengan isi dari `ref` dengan:

```
$ setfacl -M ref report2.txt
```

Jika Anda hendak memulai dengan ACL yang bersih, dan menambahkan isi dari `ref`, Anda bisa menambahkan tanda *-b* yang kita bahas sebelumnya:

```
$ setfacl -b -M ref report2.txt
```

Tentu saja, tidak diharuskan untuk menggunakan berkas ini. Kita bisa melakukan pipe dari **getfacl** ke **setfacl**, dengan menggunakan nama simbol dari masukan standar (-), dan bukan menggunakan nama berkas:

```
$ getfacl report.txt | setfacl -b -M - report2.txt
```

Opsi *-X* menghapus isi ACL yang didefinisikan pada berkas. Hal ini menggunakan sintaks yang sama dengan *-x*, dengan koma digantikan dengan baris baru.

8.6. Menemukan berkas

find

Perintah **find** merupakan utilitas yang paling mudah untuk menemukan berkas pada sistem UNIX. Hal ini karena ia bekerja pada cara yang sederhana dan mudah diprediksi: **find** akan menjelajahi pohon direktori yang ditentukan sebagai parameter pada **find**. Selain itu, sebuah pengguna bisa menentukan ekspresi yang akan dievaluasi untuk setiap berkas dan direktori. Nama dari berkas atau direktori akan dicetak jika ekspresi bernilai *true*. Argumen pertama dimulai dengan tanda minus (-), tanda seru (*!*,atau kurung buka (*(*, menandakan awal dari ekspresi. Ekspresi bisa berisi berbagai operan. Untuk jelasnya, sintaks dari **find** adalah: *find paths expression*.

Penggunaan sederhana dari **find** adalah dengan tanpa ekspresi. Karena hal ini akan cocok dengan setiap direktori dan sub direktori, semua berkas dan direktori akan dicetak. Sebagai contoh:

```
$ find .
.
./economic
./economic/report.txt
./economic/report2.txt
./technical
./technical/report2.txt
./technical/report.txt
```

Anda juga bisa menentukan banyak direktori:

```
$ find economic technical
economic
economic/report.txt
economic/report2.txt
technical
technical/report2.txt
technical/report.txt
```

Operan yang membatasi berdasakan nama atau jenis obyek

Satu skenario umum untuk mencari berkas atau direktori adalah dengan mencari berdasarkan nama. Operan *-name* bisa digunakan untuk mencocokan obyek yang memiliki nama tertentu, atau cocok dengan wildcard tertentu. Sebagai contoh, dengan menggunakan operan *-name 'report.txt'* hanya akan bernilai true untuk berkas atau direktori dengan nama `report.txt`. Sebagai contoh:

```
$ find economic technical -name 'report.txt'
economic/report.txt
technical/report.txt
```

Hal yang sama berlaku juga untuk wildcards:

```
$ find economic technical -name '*2.txt'
economic/report2.txt
technical/report2.txt
```

Catatan

Ketika menggunakan **find** Anda sebaiknya mengirimkan wildcard ke[ada **find**, dan jangan mengharapkan shell untuk memperluasnya. Jadi, pastikan pola sudah diberi tanda kutip, atau wildcard sudah di-escape.

Juga dimungkinkan untuk mengevaluasi jenis obyek dengan operan *-type c*, dimana *c* menentukan jenis yang akan dicocokkan. Tabel 8.5, “Parameter untuk operan '-type'” mendaftar berbagai jenis obyek yang bisa digunakan.

Tabel 8.5. Parameter untuk operan '-type'

Parameter	Arti
b	Berkas blok device

Parameter	Arti
c	Berkas karakter device
d	Direktori
f	Berkas biasa
l	Symbolic link
p	FIFO
s	Socket

Jadi, misalnya Anda hendak mencocokkan direktori, Anda bisa menggunakan parameter *d* pada operan *-type*:

```
$ find . -type d
.
./economic
./technical
```

Kita akan lihat ekspresi yang kompleks pada akhir bagian dari **find**, tetapi pada saat ini, cukup penting untuk diketahui bahwa Anda bisa membuat sebuah ekspresi boolean 'and' dengan menentukan berbagai operan. Sebagai contoh, *operan1 operan2* bernilai true jika kedua *operan1* dan *operan2* bernilai true untuk obyek yang sedang dievaluasi. Jadi, Anda bisa mengkombinasikan operan *-name* dan *-type* untuk menemukan semua direktori yang diawali dengan *eco*:

```
$ find . -name 'eco*' -type d
./economic
```

Operan yang membatasi berdasarkan kepemilikan atau hak akses obyek

Selain mencocokan obyek berdasarkan nama atau jenis, Anda juga bisa mencocokkan berdasarkan hak akses aktif atau kepemilikan obyek. Hal ini berguna untuk mencari berkas yang memiliki hak akses atau kepemilikan yang tidak tepat.

Pemilik atau grup dari obyek dapat dicocokkan dengan *-user username* dan *-group groupname*. Nama dari pengguna atau grup akan diinterpretasikan sebagai ID pengguna atau ID grup jika nama adalah desimal dan tidak bisa ditemukan pada sistem dengan getpwnam(3) atau getgrnam(3). Jadi, jika Anda hendak mencocokkan semua obyek yang dimiliki *joe*, Anda bisa menggunakan *-user joe* sebagai operan:

```
$ find . -user joe
./secret/report.txt
```

Atau untuk menemukan semua obyek dengan grup *friend* sebagai grup berkas:

```
$ find . -group friends
./secret/report.txt
```

Operan untuk menguji hak akses berkas *-perm* lebih bersifat trivial. Seperti perintah **chmod** operator ini bisa bekerja dengan notasi oktal maupun simbol. Kita akan mulai dengan melihat notasi oktal. Jika sebuah nomor oktal ditentukan sebagai parameter pada operan *-perm*, maka akan cocok dengan semua obyek yang memiliki hak akses yang sama

persis. Sebagai contoh, *-perm 0600* akan cocok dengan semua obyek yang hanya dapat dibaca dan ditulis oleh pengguna, dan tidak memiliki tambahan lain-lain:

```
$ find . -perm 0600
./secret/report.txt
```

Jika sebuah tanda minus ditambahkan sebagai awalan dari sebuah angka, maka akan cocok dengan semua obyek yang minimal memiliki bit yang ditentukan sebagai nomor oktal. Sebuah contoh yang berguna adalah mencari semua berkas yang paling tidak memiliki bit penulisan aktif untuk pengguna *other* dengan *-perm -0002*. Hal ini bisa membantu Anda node device atau obyek lain dengan hak akses yang tidak aman.

```
$ find /dev -perm -0002
/dev/null
/dev/zero
/dev/ctty
/dev/random
/dev/fd/0
/dev/fd/1
/dev/fd/2
/dev/psm0
/dev/bpsm0
/dev/ptyp0
```

Catatan

Beberapa node device harus bersifat dapat ditulis oleh semua pada beberapa sistem UNIX agar dapat berfungsi dengan benar. Sebagai contoh, device `/dev/null` selalu bersifat dapat ditulis.

Notasi simbol dari parameter *-perm* menggunakan notasi yang sama dengan perintah **chmod**. Hak akses simbolis dibangun dengan mode berkas dimana semua bit akan dihapus, sehingga tidak diperlukan untuk menggunakan tanda minus untuk menghapusnya. Hal ini juga mencegah ambiguitas yang dapat muncul dengan awalan tanda minus. Seperti sintaks oktal, mengawali hak akses dengan tanda minus akan mencocokkan obyek yang paling tidak memiliki hak akses yang ditentukan. Penggunaan nama simbol cukup mudah ditebak - dua perintah berikut mengulang contoh sebelumnya dengan hak akses simbolis:

```
$ find . -perm u+rw
./secret/report.txt
```

```
$ find /dev -perm -o+w
/dev/null
/dev/zero
/dev/ctty
/dev/random
/dev/fd/0
/dev/fd/1
/dev/fd/2
/dev/psm0
/dev/bpsm0
```

```
/dev/ptyp0
```

Operan yang membatasi berdasarkan waktu pembuatan obyek

Terdapat tiga operan yang beroperasi pada interval waktu. Sintaks dari operan ini adalah *operan n*, dimana *n* adalah waktu dalam hari. Semua operator menghitung waktu delta dalam detik yang dibagi dengan jumlah detik dalam satu hari (86400), tanpa memperhatikan sisanya. Jadi, jika delta adalah satu hari, *operand 1* akan cocok dengan obyek yang dicari. Ketiga operan tersebut adalah:

- *-atime n* - operan ini bernilai true jika waktu inisialisasi dari **find** dikurangi waktu akses terakhir dari obyek sama dengan *n*.
- *-ctime n* - operan ini bernilai true jika waktu inisialisasi dari **find** dikurangi waktu perubahan terakhir dari status berkas sama dengan *n*.
- *-mtime n* - operan ini bernilai true jika waktu inisialisasi dari **find** dikurangi waktu perubahan terakhir dari berkas sama dengan *n*.

Jadi, operan-operan ini cocok jika waktu akses terakhir, waktu perubahan terakhir, dan waktu modifikasi adalah *n* hari yang lalu. Untuk memberikan sebuah contoh, perintah berikut menampilkan semua obyek pada `/etc` yang sudah dimodifikasi satu hari yang lalu:

```
$ find /etc -mtime 1
/etc
/etc/group
/etc/master.passwd
/etc/spwd.db
/etc/passwd
/etc/pwd.db
```

Tanda tambah atau kurang bisa digunakan sebagai modifier untuk arti dari *n*. *+n* berarti lebih dari *n* hari, *-n* berarti kurang dari *n* hari. Jadi, untuk mencari semua berkas pada `/etc` yang dimodifikasi kurang dari dua hari yang lalu, Anda bisa menjalankan:

```
$ find /etc -mtime -2
/etc
/etc/network/run
/etc/network/run/ifstate
/etc/resolv.conf
/etc/default
/etc/default/locale
[...]
```

Operan berbasis waktu lain yang cukup penting adalah operan *-newer reffile*. Operan ini cocok dengan semua berkas yang dimodifikasi setelah berkas dengan nama `reffile`. Contoh berikut menggambarkan bagaimana Anda bisa menggunakan untuk menampilkan semua berkas yang memiliki waktu modifikasi setelah `economic/report2.txt`:

```
$ find . -newer economic/report2.txt
.
```

```
./technical
./technical/report2.txt
./technical/report.txt
./secret
./secret/report.txt
```

Operand yang berefek pada penjelajahan pohon

Beberapa operan memiliki efek pada bagaimana perintah **find** menjelajahi pohon. Operan pertama adalah *-xdev*. *-xdev* mencegah **find** mencari hingga direktori yang memiliki ID device yang berbeda, sehingga secara efektif mencegah penjelajahan dari sistem berkas yang lain. Direktori dimana sistem berkas di-mount akan dicetak, karena operan ini akan selalu mengembalikan nilai *true*. Sebuah contoh sederhana adalah sebuah sistem dimana `/usr` di-mount pada sistem berkas yang berbeda dari `/`. Sebagai contoh, jika kita mencari direktori dengan nama *bin*, hal ini akan menghasilkan hasil sebagai berikut:

```
$ find / -name 'bin' -type d
/usr/bin
/bin
```

Tetapi jika kita menambahkan *-xdev* `/usr/bin` tidak akan ditemukan, karena berada pada sistem berkas yang berbeda (dan device):

```
$ find / -name 'bin' -type d -xdev
/bin
```

Operan *-depth* memodifikasi urutan dimana direktori akan dievaluasi. Dengan *-depth* isi dari direktori akan dievaluasi terlebih dahulu, dan kemudian direktorinya. Hal ini bisa dibuktikan dengan contoh berikut:

```
$ find . -depth
./economic/report.txt
./economic/report2.txt
./economic
./technical/report2.txt
./technical/report.txt
./technical
.
```

Seperti yang bisa Anda lihat pada hasil keluaran, berkas pada direktori *./economic* dievaluasi sebelum `.`, dan `./economic/report.txt` sebelum `./economic`. *-depth* selalu bernilai *true*.

Akhirnya, operan *-prune* menyebabkan find untuk tidak menyertakan direktori yang sedang dievaluasi. *-prune* akan dibuang jika operan *-depth* juga digunakan. *-depth* selalu bernilai *true*.

Operand yang menjalankan utilitas eksternal

find menjadi alat yang sangat hebat ketika dikombinasikan dengan utilitas eksternal. Hal ini bisa dilakukan dengan operan *-exec*. Terdapat dua sintaks untuk operan *-exec*. Sintaks pertama adalah *-exec utility argument ;*. Perintah *utility* akan dieksekusi dengan argumen yang ditentukan untuk setiap obyek yang sedang dievaluasi. Jika salah satu dari

argumen adalah *{}*, maka tanda ini akan digantikan dengan berkas yang sedang dievaluasi. Hal ini sangat berguna, terutama ketika kita tidak menggunakan sintaks ekspresi tambahan, operan akan akan dievaluasi dari kiri ke kanan. Mari kita lihat dalam contoh:

```
$ find . -perm 0666 -exec chmod 0644 {} \;
```

Operan pertama mengemablikan nilai true untuk berkas yang memiliki hak akses *0666*. Operan kedua menjalankan *chmod 0644 filename* untuk setiap berkas yang dievaluasi. Jika Anda heran kenapa perintah ini tidak dijalankan untuk setiap berkas, itu merupakan pertanyaan yang bagus. Seperti interpreter ekspresi yang lain, **find** menggunakan "short-circuiting". Karena tidak ada operator lain yang ditentukan, operator logika *and* secara otomatis diasumsikan diantara kedua operan. Jika operan pertama bernilai *false*, maka tidak masuk akal untuk menguji operan lain, karena ekspresi akhir akan selalu bernilai false. Jadi, operan *-exec* hanya akan dievaluasi jika operan pertama bernilai true. Catatan lain adalah tanda titik koma yang menutup *-exec* harus di-escape, untuk mencegah agar jangan sampai diparsing oleh shell.

Satu hal yang menarik tentang operator *-exec* adalah ia bernilai *true* jika perintah berhasil dengan sempurna. Jadi, Anda juga bisa menggunakan perintah *-exec* untuk menambahkan kondisi tambahan yang tidak direpresentasikan oleh operan **find**. Sebagai contoh, perintah berikut mencetak semua obyek yang diakhiri dengan *.txt* yang berisi kata *gross income*:

```
$ find . -name '*.txt' -exec grep -q 'gross income' {} \; -print
./economic/report2.txt
```

Perintah **grep** akan dibahas kemudian. Untuk saat ini, cukup diketahui bahwa perintah tersebut bisa digunakan untuk mencocokkan pola teks. Operan *-print* mencetak path obyek aktual. Perintah ini selalu digunakan secara implisit, kecuali ketika operan *-exec* atau *-ok* digunakan.

Sintaks kedua dari operan *-exec* adalah *-exec utility arguments {} +*. Hal ini menggabungkan semua obyek yang cocok dengan nilai ekspresi bernilai true, dan menyediakan sekumpulan berkas sebagai argumen pada utilitas yang ditentukan. Contoh pertama dari operan *-exec* juga bisa ditulis sebagai:

```
$ find . -perm 0666 -exec chmod 0644 {} +
```

Perintah ini akan menjalankan perintah **chmod** hanya sekali, dengan semua berkas dimana ekspresi bernilai true sebagai argumennya. Operan selalu menghasilkan *true*.

Jika sebuah perintah dieksekusi oleh find menghasilkan nilai bukan nol (artinya eksekusi dari perintah tersebut tidak sukses), **find** juga menghasilkan nilai bukan nol.

Operator untuk membangun ekspresi yang kompleks

find menyediakan beberapa operator yang bisa dikombinasikan untuk membuat ekspresi yang lebih kompleks:

Operator

(expr)	Bernilai *true* jika *expr* bernilai *true*.
expr1 [-a] expr2	Bernilai *true* jika kedua *expr1* dan *expr2* bernilai true. Jika *-a* diabaikan, operator ini diasumsikan secara implisit. **find** akan menggunakan short-circuiting ketika operator ini diuji: *expr2* tidak akan diuji ketika *expr1* bernilai *false*

expr1 -o expr2	Bernilai *true* jika salah satu atau kedua *expr1* dan *expr2* bernilai true. **find** akan menggunakan short-circuiting ketika operator ini diuji: *expr2* tidak akan dievaluasi ketika *expr1* bernilai *true*
! expr	Negasi *expr*. Jadi, jika *expr* bernilai true, ekspresi ini akan bernilai *false* dan sebaliknya.

Karena tanda kurung dan tanda seru diinterpretasikan oleh sebagian besar shell, mereka biasanya harus di-escape.

Contoh berikut menggambarkan beberapa operator dalam penggunaannya. Perintah ini menjalankan **chmod** untuk semua berkas yang memiliki hak akses *0666* atau *0664*.

```
$ find . \( -perm 0666 -o -perm 0664 \) -exec chmod 0644 {} \;
```

which

Perintah **which** bukan bagian dari Single UNIX Specification versi 3, tetapi disediakan oleh banyak sistem. **which** mencari sebuah perintah yang berada pada path pengguna (yang ditentukan oleh variabel PATH) dan mencetak path lengkapnya. Dengan menyediakan nama perintah sebagai parameter akan menampilkan path lengkap:

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

Anda juga bisa melakukan query path dari banyak perintah:

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

which mengembalikan nilai tidak nol jika perintah tidak bisa ditemukan.

whereis

Perintah **whereis** mencari berkas biner, halaman manual, dan sumber dari perintah pada tempat yang sudah ditentukan terlebih dahulu. Sebagai contoh, perintah berikut menampilkan path dari **ls** dan halaman manual dari ls(1):

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

locate

Slackware Linux juga menyediakan perintah **locate** yang mencari melalui berkas basis data yang bisa dihasilkan secara periodik dengan perintah **updatedb**. Karena ia menggunakan basis data yang dibangun pada sistem berkas, maka perintah ini bekerja lebih cepat dari **which**, terutama ketika informasi direktori belum disimpan dalam cache. meski demikian, kombinasi **locate**/**updatedb** memiliki beberapa kekurangan:

- Berkas baru bukan bagian dari basis data sampai pemanggilan **updatedb** berikutnya.
- **locate** tidak memiliki batasan hak akses, sehingga pengguna bisa mencari berkas yang biasanya tersembunyi bagi mereka.
- Implementasi yang lebih baru, bernama *slocate* mengatasi masalah hak akses, tetapi membutuhkan hak yang lebih tinggi. Ini merupakan variasi dari **locate** yang disertakan pada Slackware Linux.

Dengan sistem berkas yang semakin cepat, dan dengan menerapkan akal sehat ketika menjalankan query dengan **find**, **locate** tampaknya menjadi tidak berarti. Tentu saja, keputusan Anda bisa berbeda. Penggunaan dasar dari **locate** adalah *locate namaberkas*. Sebagai contoh:

```
$ locate locate
/usr/bin/locate
/usr/lib/locate
/usr/lib/locate/bigram
/usr/lib/locate/code
/usr/lib/locate/frcode
[...]
```

8.7. Arsip

Perkenalan

Cepat atau lambat, pengguna GNU/Linux akan menjumpai arsip tar, tar adalah format standar untuk membuat arsip berkas pada GNU/Linux. Tar sering digunakan bersamaan dengan **gzip** atau **bzip2**. Kedua perintah tersebut mampu mengkompresi berkas dan arsip. Tabel 8.6, “Ekstensi berkas arsip” menampilkan ekstensi arsip yang paling sering digunakan, dan artinya.

Tabel 8.6. Ekstensi berkas arsip

Ekstensi	Arti
.tar	Arsip tar tidak terkompresi
.tar.gz	Arsip tar dikompresi dengan gzip
.tgz	Arsip tar dikompresi dengan gzip
.tar.bz2	Arsip tar dikompresi dengan bzip2
.tbz	Arsip tar dikompresi dengan bzip2

Perbedaan antara **bzip2** dan **gzip** adalah **bzip2** bisa mencari informasi yang diulang-ulang pada blok yang lebih besar, sehingga menghasilkan kompresi yang lebih baik. Tetapi **bzip2** juga lebih lambat, karena melakukan lebih banyak analisa data.

Mengurai arsip

Karena beberapa perangkat lunak dan data pada dunia GNU/Linux diarsip dengan **tar**, menjadi sangat penting untuk membiasakan diri dengan menguraikan arsip tar. Langkah pertama yang harus Anda lakukan ketika menerima arsip tar adalah menampilkan isinya. Hal ini bisa dilakukan dengan parameter `t`. Namun, jika kita hanya menjalankan **tar**

dengan parameter ini, dan nama arsip, maka tar hanya akan menunggu sampai Anda memasukkan sesuatu ke standard input:

```
$ tar t test.tar
```

Hal ini terjadi karena **tar** membaca data dari standar input. Jika Anda lupa bagaimana pengalihan bekerja, merupakan ide bagus untuk membaca ulang Bagian 7.7, “Pengalihan dan pipe”. Mari kita lihat apa yang terjadi jika kita mengalihkan arsip tar kita ke tar:

```
$ tar t < test.tar
test/
test/test2
test/test1
```

Ini lebih sesuai dengan apa yang Anda harapkan. Arsip ini berisi sebuah direktori `test`, yang berisi berkas `test2` dan `test1`. Juga dimungkinkan untuk menentukan nama arsip sebagai parameter pada **tar**, dengan menggunakan parameter *f*:

```
$ tar tf test.tar
test/
test/test2
test/test1
```

Ini merupakan arsip yang berisi berkas yang berguna ;). Kita bisa melanjutkan dan menguraikan arsip dengan menggunakan parameter *x*:

```
$ tar xf test.tar
```

Sekarang kita bisa melakukan verifikasi bahwa tar menguraikan arsip dengan melihat isi direktori dengan **ls**:

```
$ ls test/
test1  test2
```

Menguraikan atau menampilkan berkas dari arsip yang digabung dengan gzip atau bzip tidaklah lebih susah. Hal ini bisa dilakukan dengan menambahkan parameter *z* atau *b* untuk arsip yang dikompresi dengan **gzip** atau **bzip2**. Sebagai contoh, kita bisa menampilkan isi dari arsip yang dikompresi dengan gzip dengan :

```
$ tar ztf archive2.tar.gz
```

Dan arsip dengan kompresi bzip bisa diuraikan dengan :

```
$ tar bxf archive3.tar.bz2
```

Membuat arsip

Anda bisa membuat arsip dengan parameter *c*. Misalkan kita memiliki direktori `test` yang ditampilkan pada contoh sebelumnya. Kita bisa membuat arsip dengan direktori `test` dan berkas pada direktori ini dengan:

```
$ tar cf important-files.tar test
```

Perintah ini akan membuat arsip `important-files.tar` (yang ditentukan dengan parameter *f*). Sekarang kita bisa memverifikasi arsip:

```
$ tar tf important-files.tar
test/
test/test2
test/test1
```

Membuat arsip dengan kompresi gzip atau bzip bisa dilakukan pada baris yang sama dengan penguraian arsip terkompresi: tambahkan parameter *z* untuk mengkompresi arsip dengan gzip, atau *b* untuk mengkompresi arsip dengan bzip. Misalkan kita hendak membuat arsip terkompresi versi **gzip** dari contoh diatas. Kita melakukannya dengan:

```
tar zcf important-files.tar.gz test
```

8.8. Melakukan mount sistem berkas

Perkenalan

Seperti yang lain, Linux menggunakan satu teknik yang disebut “mounting” untuk mengakses sistem berkas. Mounting berarti sebuah sistem berkas dihubungkan pada sebuah direktori pada sistem berkas root. Seseorang bisa melakukan mount sebuah drive CD-ROM pada direktori `/mnt/cdrom`. Linux mendukung banyak jenis sistem berkas, seperti Ext2, Ext3, ReiserFS, JFS, XFS, ISO9660 (digunakan untuk CD-ROM), UDF (digunakan pada beberapa DVD) dan sistem berkas DOS/Windows, seperti FAT, FAT32 dan NTFS. Sistem berkas ini bisa berada pada banyak media, misalnya hard disk, CD-RM, dan disk Flash. Bagian ini menjelaskan bagaimana sistem berkas bisa di-mount dan di-unmount.

mount

Perintah **mount** digunakan untuk melakukan mount terhadap sistem berkas. Sintaks dasarnya adalah : “mount /dev/ devname /mountpoint”. Nama device bisa berupa blok device, seperti hard disk atau drive CD-ROM. Mount point bisa berupa titik pada sistem berkas root. Mari kita lihat contoh:

```
# mount /dev/cdrom /mnt/cdrom
```

Hal ini akan me-mount `/dev/cdrom` pada mount point `/mnt/cdrom`. Nama device `/dev/cdrom` biasanya merupakan link ke nama device CD-ROM yang sebenarnya (misalnya `/dev/hdc`). Seperti yang Anda lihat, konsepnya sangat sederhana, hanya membutuhkan sedikit waktu untuk mempelajari nama device ;). Seringkali Anda juga perlu menentukan jenis sistem berkas yang akan di-mount. Jenis sistem berkas bisa ditentukan dengan menambahkan parameter `-t`:

```
# mount -t vfat /dev/sda1 /mnt/flash
```

Hal ini akan me-mount sistem berkas vfat pada `/dev/sda1` ke `/mnt/flash`.

umount

Perintah **umount** digunakan untuk melepas mount terhadap sistem berkas. **umount** menerima dua jenis parameter, mount point atau device. Sebagai contoh:

```
# umount /mnt/cdrom
# umount /dev/sda1
```

Perintah pertama akan melepas mount sistem berkas yang di-mount pada `/mnt/cdrom`, perintah kedua melepas mount sistem berkas pada `/dev/sda1`.

Berkas fstab

Sistem GNU/Linux memiliki berkas khusus, `/etc/fstab`, yang menentukan sistem berkas apa yang harus di-mount selama sistem boot. Mari kita lihat contoh:

```
/dev/hda10      swap            swap       defaults         0   0
/dev/hda5       /               xfs        defaults         1   1
/dev/hda6       /var            xfs        defaults         1   2
/dev/hda7       /tmp            xfs        defaults         1   2
/dev/hda8       /home           xfs        defaults         1   2
/dev/hda9       /usr            xfs        defaults         1   2
/dev/cdrom      /mnt/cdrom      iso9660    noauto,owner,ro  0   0
/dev/fd0        /mnt/floppy     auto       noauto,owner     0   0
devpts          /dev/pts        devpts     gid=5,mode=620   0   0
proc            /proc           proc       defaults         0   0
```

Seperti yang Anda lihat, setiap catatan pada berkas `fstab` memiliki lima bagian: fs_spec, fs_file, fs_vfstype, fs_mntops, fs_freq, dan fs_passno. Kita akan melihat pad setiap bagian.

fs_spec

Opsi fs_spec menentukan blok device, atau sistem berkas remote yang harus di-mount. Seperti yang Anda lihat pada contoh, beberapa partisi /dev/hda ditentukan, begitu juga dengan drive CD-ROM dan disket. Ketika volume NFS di-mount, alamat IP dan direktori bisa ditentukan, seperti contoh: `192.168.1.10:/exports/data`.

fs_file

fs_file menentukan titik mount (mount point). Ini bisa berupa direktori pada sistem berkas.

fs_vfstype

Opsi ini menentukan sistem berkas apa yang digunakan. Sebagai contoh, hal ini bisa berupa : ext2, ext3, reiserfs, xfs, nfs, vfat, or ntfs.

fs_mntops

Opsi fs_mntops menentukan parameter yang harus digunakan untuk melakukan mount terhadap sistem berkas. Halaman manual **mount** memiliki deskripsi yang lengkap tentang opsi yang tersedia. Ini merupakan opsi yang paling menarik:

- *noauto*: sistem berkas yang ditampilkan pada `/etc/fstab` pada umumnya akan di-mount secara otomatis. Ketika "noauto" digunakan, sistem berkas tidak akan di-mount selama sistem boot, tetapi hanya jika diberikan perintah **mount**. Ketika melakukan mount terhadap sistem berkas tersebut, hanya mount point atau nama device yang harus ditentukan, misalnya: **mount /mnt/cdrom**

- *user*: menambahkan opsi "user" akan mengijinkan pengguna biasa untuk melakukan mount terhadap sistem berkas (biasanya hanya pengguna root yang diijinkan untuk melakukan mount terhadap sistem berkas).

- *owner*: Opsi "owner" mengijinkan pemilik dari device yang ditentukan untuk melakukan mount terhadap device yang ditentukan. Anda bisa melihat pemilik dari device menggunakan **ls**, Cth. **ls -l /dev/cdrom**.

- *noexec*: dengan opsi ini aktif, pengguna tidak bisa menjalankan berkas dari sistem berkas yang di-mount. Hal ini bisa digunakan untuk menyediakan keamanan yang lebih baik.

- *nosuid*: opsi ini hampir serupa dengan opsi "noexec". Dengan "nosuid" aktif, bit SUID pada berkas pada sistem berkas tidak akan diijinkan. SUID digunakan untuk beberapa biner untuk mengijinkan pengguna melakukan sesuatu yang hanya bisa dilakukan oleh pengguna khuusus (root). Hal ini jelas merupakan ancaman keamanan, sehingga opsi ini hanya seharusnya digunakan pada media removable. Pengguna biasa akan memaksa opsi nosuid, tetapi tidak bagi pengguna root!

- *unhide*: opsi ini hanya relevan untuk CD-ROM dengan sistem berkas ISO9660. Jika "unhide" ditentukan, maka berkas tersembunyi akan ditampilkan.

fs_freq

Jika "fs_freq" ditentukan menjadi 1 atau lebih tinggi, ia menentukan berapa haris sebuah sistem berkas akan dibuat cadangannya (backup). Opsi ini hanya digunakan ketika dump [http://dump.sourceforge.net/] terinstall, dan ditentukan untuk menangani hal ini.

fs_passno

Kolom ini digunakan oleh **fsck** untuk menentukan urutan pengujian sistem berkas selama sistem boot.

8.9. Mengenkripsi dan menandatangani berkas

Perkenalan

Terdapat dua mekanisme keamanan untuk mengamankan berkas: menandatangani berkas dan mengenkripsi berkas. Menandatangani berkas berarti sebuah tanda tangan digital khusus dibuat untuk sebuah berkas. Anda, atau orang lain bisa menggunakan tanda tangan (signature) untuk memastikan integritas dari sebuah berkas. Enkripsi berkas mengubah sebuah berkas sehingga hanya hanya orang yang berhak untuk membaca mampu melihat isi dari berkas tersebut.

Sistem ini bergantung dari dua kunci: kunci pribadi (private) dan kunci publik. Kunci publik digunakan untuk mengenkripsi berkas, dan berkas hanya bisa didekripsi dengan kunci private. Hal ini berarti bahwa orang bisa mengirimkan kunci publiknya ke orang lain. Orang lain bisa menggunakan kunci ini untuk mengirimkan berkas terenkripsi, bahwa hanya orang dengan kunci private yang bisa mendekripsinya. Tentu saja, hal ini berarti keamanan dari sistem ini tergantung dari seberapa bagus kunci private dijaga.

Slackware Linux menyediakan aplikasi untuk menandatangani dan mengenkripsi berkas, bernama GnuPG. GnuPG bisa diinstall dari set disk “n”.

Menghasilkan kunci private dan publik Anda

Menghasilkan kunci publik dan private cukup rumit, karena GnuPG menggunakan kunci DSA sebagai default. DSA adalah algoritma enkripsi, masalahnya adalah pada panjang kunci maksimal dari DSA yaitu 1024 bit, yang dianggap terlalu pendek untuk jangka panjang. Hal ini mengapa ide bagus untuk menggunakan kunci RSA 2048 bit. Bagian ini menjelaskan bagaimana hal ini bisa dilakukan.

Catatan

Kunci 1024-bit diyakini aman untuk waktu yang lama. Tetapi paper dari Bernstein *Circuits for Integer Factorization: a Proposal* berkata lain, intinya adalah cukup mungkin untuk national security agency untuk menghasilkan perangkat keras yang mampu membobol kunci dalam waktu yang relatif singkat. Selain itu, juga dibuktikan bahwa kunci RSA 512-bit bisa dipecahkan dalam waktu yang relatif singkat dengan perangkat keras yang umum. Informasi tentang masalah ini bisa ditemukan pada email berikut: http://tin.le.org/vault/security/encryption/rsa1024.html

Kita bisa menghasilkan kunci dengan menjalankan :

```
$ gpg --gen-key
```

Pertanyaan pertama adalah jenis kunci yang akan dibuat. Kita akan memilih *(4) RSA (sign only)*:

```
Please select what kind of key you want:
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (4) RSA (sign only)
Your selection? 4
```

Anda akan ditanyakan ukuran kunci yang ingin Anda buat. Ketikkan *2048* untuk menghasilkan kunci 2048 bit, dan tekan enter untuk melanjutkan.

```
What keysize do you want? (1024) 2048
```

Pertanyaan berikutnya mudah untuk dijawab, cukup pilih sesuai keinginan Anda. Pada umumnya, bukanlah ide yang jelek untuk memilih kunci yang valid selamanya. Anda bisa menonaktifkan kunci dengan sertifikat pembatalan khusus.

```
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
```

GnuPG akan mencoba mengkonfirmasi. Setelah mengkonfirmasi, GnuPG akan menanyakan nama dan alamat email Anda. GnuPG juga akan menanyakan tentang komentar, Anda bisa membiarkannya kosong, atau Anda bisa mengisi seperti “Work” atau “Private”, untuk mengindikasikan fungsi dari kunci ini. Sebagai contoh:

```
Real name: John Doe
Email address: john@doe.com
Comment: Work
You selected this USER-ID:
    "John Doe (Work) <john@doe.com>"
```

GnuPG akan meminta Anda untuk mengkonfirmasi ID pengguna Anda. Setelah itu, GnuPG akan meminta Anda untuk mengisi kata sandi. Pastikan Anda menggunakan kata sandi yang bagus:

```
You need a Passphrase to protect your secret key.

Enter passphrase:
```

Setelah memasukkan kata sandi sebanyak dua kali, GnuPG akan menghasilkan kunci. Tetapi kita belum selesai. GnuPG hanya menghasilkan kunci untuk menandatangani informasi, bukan untuk mengenkripsi informasi. Untuk melanjutkan, lihat hasil keluaran dan cari ID kunci. Pada informasi kunci, Anda bisa melihat *pub 2048R/*. ID kunci dicetak setelah bagian ini. Pada contoh ini:

```
public and secret key created and signed.
key marked as ultimately trusted.

pub  2048R/8D080768 2004-07-16 John Doe (Work) <john@doe.com>
     Key fingerprint = 625A 269A 16B9 C652 B953  8B64 389A E0C9 8D08 0768
```

ID kunci adalah *8D080768*. Jika Anda kehilangan hasil keluaran, Anda bisa menemukan ID kunci dari hasil perintah **gpg --list-keys**. Gunakan ID kunci ini untuk memberitahukan GnuPG bahwa Anda hendak mengedit kunci Anda:

```
$ gpg --edit-key <Key ID>
```

Dengan kunci diatas, perintah diatas menjadi:

```
$ gpg --edit-key 8D080768
```

GnuPG akan menampilkan prompt perintah. Jalankan perintah **addkey** pada prompt ini:

```
Command> addkey
```

GnuPG akan menanyakan kata sandi yang Anda gunakan untuk kunci Anda:

```
Key is protected.

You need a passphrase to unlock the secret key for
user: "John Doe (Work) <john@doe.com>"
2048-bit RSA key, ID 8D080768, created 2004-07-16

Enter passphrase:
```

Setelah memasukkan kata sandi, GnuPG akan menanyakan jenis kunci yang hendak Anda buat. Pilih *RSA (encrypt only)*, dan isi informasi seperti sebelumnya (pastikan Anda menggunakan kunci 2048 bit). Sebagai contoh:

```
Please select what kind of key you want:
   (2) DSA (sign only)
   (3) ElGamal (encrypt only)
   (4) RSA (sign only)
   (5) RSA (encrypt only)
Your selection? 5
What keysize do you want? (1024) 2048
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
```

Dan konfirmasikan bahwa informasi itu benar. Setelah kunci dibuat, Anda bisa meninggalkan prompt GnuPG, dan simpan kunci baru dengan perintah **save**:

```
Command> save
```

Selamat!, Anda sudah menghasilkan kunci yang diperlukan untuk mengenkripsi dan mendekripsi email dan berkas. Anda sekarang bisa mengkonfigurasikan klien email Anda untuk menggunakan GnuPG. Merupakan ide bagus untuk menyimpan isi dari direktori `.gnupg` pada media yang dapat dipercaya, dan simpan dalam tempat yang amann! Jika kunci private Anda hilang, Anda tidak bisa mendekripsi berkas dan pesan yang dienkripsi dengan kunci publik Anda. Jika kunci private, dan kata sandi Anda dicuri, keamanan dari sistem ini jelas-jelas hilang.

Mengekspor kunci publik Anda

Untuk membuat GnuPG berguna, Anda harus memberikan kunci publik Anda kepada orang-orang yang mengirimkan berkas atau email kepada Anda. Mereka bisa menggunakan kunci publik Anda untuk mengenkripsi berkas, atau menggunakannya untuk memastikan apakah berkas memiliki signature yang benar atau tidak. Kunci bisa diekspor dengan parameter *`--export`*. Merupakan ide yang bagus untuk menentukan parameter *`--output`*, hal ini akan menyimpan berkas pada sebuah berkas. Contoh berikut akan menyimpan kunci publik dari *John Doe*, yang digunakan pada contoh sebelumnya, pada berkas `key.gpg`:

```
$ gpg --output key.gpg --export john@doe.com
```

Hal ini menyimpan kunci dalam format biner. Seringkali, lebih nyaman untuk menggunakan apa yang disebut “ASCII armored”, yang lebih cocok untuk menambahkan kunci pada email, atau halaman web. Anda bisa mengekspor versi ASCII dengan menambahkan parameter `--armor`:

```
$ gpg --armor --output key.gpg --export john@doe.com
```

If you look at the `key.gpg` file you will notice that the ASCII armored key is a much more comfortable format.

Tanda tangan

Dengan GPG Anda bisa membuat tanda tangan (signature) untuk sebuah berkas. Tanda tangan ini bersifat unik, karena tanda tangan Anda hanya bisa dibuat oleh kunci pribadi Anda. Hal ini berarti orang lain bisa memastikan apakah berkas memang dikirim oleh Anda, dan apakah diubah atau tidak selama pengiriman. Berkas bisa ditandai dengan parameter `--detach-sign`. Mari kita lihat pada sebuah contoh. Perintah berikut akan membuat sebuah tanda tangan untuk berkas `memo.txt`. Tanda tangan akan disimpan pada `memo.txt.sig`.

```
$ gpg --output memo.txt.sig --detach-sign memo.txt

You need a passphrase to unlock the secret key for
user: "John Doe (Work) <john@doe.com>"
2048-bit RSA key, ID 8D080768, created 2004-07-16

Enter passphrase:
```

Seperti yang Anda lihat, GnuPG akan meminta Anda untuk memasukkan kata sandi untuk kunci private Anda. Setelah Anda memasukkan kata sandi yang benar, berkas signature (`memo.txt.sig`) akan dibuat.

Anda bisa melakukan verifikasi sebuah berkas dengan signaturenya dengan menggunakan `--verify`. Tentukan berkas signature sebagai parameter kepada `--verify`. Berkas yang perlu diverifikasi bisa ditentukan sebagai parameter final:

```
$ gpg --verify memo.txt.sig memo.txt
gpg: Signature made Tue Jul 20 23:47:45 2004 CEST using RSA key ID 8D080768
gpg: Good signature from "John Doe (Work) <john@doe.com>"
```

Hal ini akan mengkonfirmasikan bahwa berkas memang ditandatangani oleh *John Doe (Work) <john@doe.com>*, dengan kunci *8D080768*, dan berkas tadi belum berubah. Jika misalnya berkas berubah, GnuPG akan memprotesnya:

```
$ gpg --verify memo.txt.sig memo.txt
gpg: Signature made Tue Jul 20 23:47:45 2004 CEST using RSA key ID 8D080768
gpg: BAD signature from "John Doe (Work) <john@doe.com>"
```

Enkripsi

Salah satu fitur utama dari GnuPG adalah enkripsi. Karena penggunaan kriptografi asimetris, orang yang mengenkripsi berkas dan orang yang mendekripsi berkas tidak harus berbagi kunci yang sama. Anda bisa mengenkripsi sebuah berkas dengan kunci publik dari orang lain, dan hanya orang tersebut yang bisa mendekripsinya dengan kunci privatenya.

Anda bisa mengenkripsi berkas dengan `--encrypt`. Jika Anda tidak menyatakan ID pengguna yang akan digunakan sebagai penerima, GnuPG akan menanyakannya. Anda bisa menentukan ID pengguna dengan parameter `-r`. Pada contoh berikut, berkas `secret.txt` akan dienkripsi untuk orang lain bernama *John Doe*:

```
$ gpg --encrypt -r "John Doe" secret.txt
```

ID pengguna diberi tanda kutip ganda untuk memastikan ID diinterpretasikan sebagai satu argumen. Setelah enkripsi selesai, versi terenkripsi akan tersedia sebagai `secret.txt.gpg`.

Pengguna yang menerima berkas bisa mendekripsinya dengan parameter `--decrypt` dari perintah **gpg**:

```
$ gpg --output secret.txt --decrypt secret.txt.gpg

You need a passphrase to unlock the secret key for
user: "John Doe (Work) <john@doe.com>"
2048-bit RSA key, ID 8D080768, created 2004-07-16 (main key ID EC3ED1AB)

Enter passphrase:

gpg: encrypted with 2048-bit RSA key, ID 8D080768, created 2004-07-16
      "John Doe (Work) <john@doe.com>"
```

Pada contoh ini, parameter `--output` digunakan untuk menyimpan isi yang sudah didekripsi pada `secret.txt`.

Bab 9. Pemrosesan Teks

Manipulasi teks adalah salah satu keunggulan UNIX, karena merupakan inti dari filosofi UNIX, seperti yang dijelaskan pada Bagian 2.4, “Filosofi UNIX”. Sebagian besar perintah UNIX merupakan perintah sederhana yang membaca data dari masukan standar, melakukan beberapa operasi pada data, dan mengirimkan hasil ke keluaran standar program. Program ini biasanya bertindak sebagai filter, yang bisa dihubungkan dengan bantuan pipe. Hal ini mengijinkan pengguna untuk meletakkan aplikasi UNIX untuk kebutuhan yang lain. Pada bagian selanjutnya Anda akan melihat bagaimana Anda bisa membuat filter Anda sendiri.

Bagian ini menjelaskan beberapa perintah UNIX sederhana namun penting yang bisa digunakan untuk memanipulasi teks. Setelah itu, kita akan menjelajah lebih lanjt ke ekspresi teratur (regular expressions) sebuah sub bahasa yang dapat digunakan untuk mencocokkan pola teks.

9.1. Manipulasi teks sederhana

Mengulang apa yang dikatakan

Filter teks yang paling sederhana adalah **cat**, Ia hanya mengirimkan data dari stdin ke stdout:

```
$ echo "hello world" | cat
hello world
```

Fitur lain adalah Anda bisa mengirimkan isi sebuah berkas ke standard output:

```
$ cat file.txt
Hello, this is the content of file.txt
```

cat akan mengerjakan sesuai namanya jika beberapa berkas ditambahkan sebagai argumen. Hal ini akan menggabungkan berkas, dimana program akan mengirimkan isi dari semua berkas ke standard output, dalam urutan yang sama sesuai dengan urutan pada argumen. Contoh berikut menggambarkan proses tersebut:

```
$ cat file.txt file1.txt file2.txt
Hello, this is the content of file.txt
Hello, this is the content of file1.txt
Hello, this is the content of file2.txt
```

Statistik teks

Perintah **wc** menyediakan statistik tentang berkas teks atau aliran teks. Tanpa parameter apapun, perintah ini akan menghitung jumlah baris, jumlah kata, dan jumlah byte. Sebuah kata dibatasi oleh satu karakter white space, atau sekumpulan karakter whitespace.

Contoh berikut menampilkan jumlah baris, kata, dan byte dari contoh “Hello world!”:

```
$ echo "Hello world!" | wc
       1       2      13
```

Jika Anda hendak mencetak hanya satu dari ketiga komponen ini, Anda bisa menggunakan salah satu dari parameter `-l` (baris), `-w` (kata), atau `-c` (byte). Sebagai contoh, menambahkan parameter `-l` akan menampilkan jumlah baris dalam sebuah berkas:

```
$ wc -l /usr/share/dict/words
  235882 /usr/share/dict/words
```

Atau, Anda bisa mencetak kolom tambahan dengan menambahkan parametrer:

```
$ wc -lc /usr/share/dict/words
 235882 2493082 /usr/share/dict/words
```

Harap diperhatikan bahwa urutan keluaran akan selalu sama (baris, kata, byte) tanpa tergantung urutan dari opsi.

Karena `-c` mencetak jumlah byte, parameter ini mungkin tidak merepresentasikan jumlah karakter yang ada pada teks, karena set karakter yang digunakan mungkin bisa lebih besar dari satu byte. Untuk mengatasi ini, parameter `-m` telah ditambahkan yang akan mencetak jumlah karakter pada teks, dimana parameter ini bebas dari set karakter yang digunakan. `-c` and `-m` hanyalah pengganti, dan tidak bisa digunakan secara bersamaan

Statistik yang diberikan **wc** lebih berguna dari apa yang seringkali tampak. Misalnya, parameter `-l` digunakan sebagai counter untuk keluaran dari sebuah perintah. Hal ini berguna, karena banyak perintah membedakan unit logis dengan baris baru. Misalkan Anda hendak menghitung jumlah baris pada direktori home Anda yang memiliki nama berkas diakhiri dengan `.txt`. Anda bisa melakukan hal ini dengan mengkombinasikan **find** untuk mencari berkas yang relevan dan **wc** untuk menghitung jumlahnya:

```
$ find ~ -name '*.txt' -type f | wc -l
```

Memanipulasi karakter

Perintah **tr** dapat digunakan untuk melakukan berbagai operasi yang umum, seperti mengganti karakter, menghapus karakter, dan meringkas urutan karakter. Satu atau dua karakter harus ditentukan pada sebagian operasi. Selain karakter biasa, terdapat beberapa karakter khusus yang bisa digunakan:

\karakter	Notasi ini digunakan untuk menentukan karakter apa yang memerlukan proses escape, utamanya \n (baris baru), \t (tab horizontal), dan \\ (backslash).
karakter1-karakter2	Secara implisit menyisipkan semua karakter dari *karakter1* hingga *karakterr2*. Notasi ini harus digunakan dengan hati-hati, karena tidak selalu memberikan hasil yang diharapkan. Sebagai contoh, urutan *a-d* akan menghasilkan *abcd* untuk lokal POSIX (pengaturan bahasa), tetapi belum tentu benar untuk lokal yang lain.
[:class:]	Cocok dengan kelas karakter yang sudah didefinisikan terlebih dahulu. Semua kemungkinan ditampilkan pada Tabel 9.1, “Kelas karakter tr”.
[karakter*]	Mengulang *karakter* sampai set kedua sama panjangnya dengan set karakter pertama. Hal notasi ini hanya bisa digunakan pada set kedua.

[karakter*n] Mengulang *karakter n* kali.

Tabel 9.1. Kelas karakter tr

Kelas	Arti
[:alnum:]	Semua huruf dan angka.
[:alpha:]	Huruf.
[:blank:]	Whitespace horizontal (cth. spasi dan tab).
[:cntrl:]	Karakter kontrol.
[:digit:]	Semua angka (0-9).
[:graph:]	Semua karakter yang dapat dietak, kecuali whitespace.
[:lower:]	Huruf kecil.
[:print:]	Semua karakter yang dapat dicetak, termasuk whitespace horizontal, tetapi tidak termasuk whitespace vertical.
[:punct:]	Karakter tanda baca.
[:space:]	Semua whitespace.
[:upper:]	Huruf besar.
[:xdigit:]	Digit heksadesimal (0-9, a-f).

Menukar karakter

Operasi default dari **tr** adalah untuk menukar (menterjemahkan) karakter. Hal ini berarti karakter ke*n* pada set pertama akan diganti dengan karakter ke-*n* pada set kedua. Sebagai contoh, Anda bisa mengganti semua *e* dengan *i* dan *o* dengan *a* dengan satu operasi **tr**:

```
$ echo 'Hello world!' | tr 'eo' 'ia'
Hilla warld!
```

Ketika set kedua tidak sebesar set pertama, karakter terakhir dari set kedua akan diulang. Hal ini tidak berlaku pada sistem UNIX lain. Jadi jika Anda hendak menggunakan **tr** pada mode yang bebas dari ketergantungan sistem, tentukan karakter yang harus diulang secara eksplisit. Sebagai contoh:

```
$ echo 'Hello world!' | tr 'eaiou' '[@*]'
H@ll@ w@rld!
```

Contoh lain adalah penggunaan sintaks repetisi ditengah. Misalkan set 1 adalah *abcdef*, dan set 2 *@[-*]!*. **tr** akan mengganti *a* dengan *@*, *b*, *c*, *d*, dan *e* dengan -, dan *f* dengan *!*. Beberapa sistem UNIX lain akan mengganti *a* dengan *@*, dan sisanya dengan -. Jadi, notasi yang lebih benar adalah dengan lebih eksplisit *@[-*4]!*, yang akan memberikan hasil yang sama pada semua sistem UNIX:

```
$ echo 'abcdef' | tr 'abcdef' '@[-*4]!'
```

```
@----!
```

Meringkas urutan karakter

Ketika parameter *-s* digunakan, **tr** akan meringkas semua karakter set yang berada pada set kedua. Hal ini berarti sebuah urutan yang sama akan direduksi menjadi satu karakter. Mari kita meringkas karakter "*e*":

```
$ echo "Let's squeeze this." | tr -s 'e'
Let's squeze this.
```

Kita bisa mengkombinasikan hal ini dengan translasi untuk menunjukkan contoh dari **tr**. Misalkan kita hendak menandai semua huruf hidup dengan tanda *at* (@), dengan huruf hidup berikutnya ditandai dengan satu tand a *at* sign. Hal ini bisa dilakukan dengan mudah dengan menggabungkan dua perintah **tr** dengan pipe:

```
$ echo "eenie meenie minie moe" | tr 'aeiou' '[@*]' | tr -s '@'
@n@ m@n@ m@n@ m@
```

Menghapus karakter

Akhirnya, **tr** bisa digunakan untuk menghapus karakter. Jika parameter *-d* digunakan, semua karakter dari set pertama akan dihapus:

```
$ echo 'Hello world!' | tr -d 'lr'
Heo wod!
```

Cut dan paste kolom teks

Perintah **cut** disediakan oleh sistem UNIX untuk “memotong” satu atau lebih kolom dari sebuah berkas atau aliran data, dan mencetaknya ke standard output. Seringkali perintah ini berguna untuk mengambil beberapa informasi dari teks. **cut** menyediakan tiga pendekatan untuk mengambil informasi dari berkas:

1. Dengan byte.
2. Dengan karakter, yang tidak sama dengan mengambil dengan byte pada sistem yang menggunakan set karakter yang lebih besar dari delapan bit.
3. Dengan kolom, yang ditandai dengan sebuah karakter.

Pada semua pendekatan, Anda bisa menentukan elemen yang dipilih dengan angkanya dimulai dari *1*. Anda bisa menentukan interval dengan menggunakan tanda dash (-). Jadi, *M-N* berarti elemen ke-M hingga ke N. Menghilangkan M (*-N*) akan memilih semua elemen dari elemen pertama hingga elemen ke-N. Menghilangkan N (*M-*) akan memilih elemen ke-M hingga elemen terakhir. Banyak elemen atau interval bisa dikombinasikan dengan memisahkannya dengan koma (,). Jadi, sebagai contoh, *1,3-* memilih elemen pertama dan elemen ketiga hingga elemen terakhir.

Data bisa diambil per kolom dengan parameter *-f kolom*. Secara default, tab horizontal digunakan sebagai pemisah. Mari kita lihat bagaimana **cut** digunakan dengan kamus Belanda ke Inggris:

```
$ cat dictionary
appel   apple
banaan  banana
peer    pear
```

Kita bisa mendapatkan semua kata-kata bahasa Inggris dengan memilih kolom pertama:

```
$ cut -f 2 dictionary
apple
banana
pear
```

Itu cukup mudah. Mari kita lakukan hal yang sama dengan berkas yang memiliki tanda titik dua sebagai pemisah kolom. Kita bisa melakukan hal ini dengan mudah dengan mengkonversi kamus dengan perintah **tr** yang sudah kita lihat sebelumnya, dengan mengganti semua tab dengan titik dua:

```
$ tr '\t' ':' < dictionary > dictionary-new
$ cat dictionary-new
appel:apple
banaan:banana
peer:pear
```

Jika kita menggunakan perintah yang sama pada contoh sebelumnya, kita tidak akan mendapatkan hasil yang benar:

```
$ cut -f 2 dictionary-new
appel:apple
banaan:banana
peer:pear
```

Apa yang terjadi disini adalah karena pembatas tidak dapat ditemukan. Jika sebuah baris tidak mengandung pembatas yang digunakan, maka **cut** akan mencetak baris tersebut. Anda bisa mencetak hal ini dengan parameter `-s`.

Untuk menggunakan pembatas yang berbeda dengan tab horizontal, tambahkan parameter `-d karakter_pembatas` untuk menentukan karakter pembatas. Jadi, pada kasus `dictionary-new`, kita akan meminta **cut** untuk menggunakan tanda titik dua sebagai pembatas:

```
$ cut -d ':' -f 2 dictionary-new
apple
banana
pear
```

Jika sebuah kolom yang ditentukan tidak ada dalam baris, maka kolom tersebut tidak akan dicetak.

Parameter `-b byte` dan `-c karakter` memilih byte dan karakter dari teks. Pada sistem lama, sebuah karakter besarnya sama dengan satu byte. Tetapi sistem yang baru bisa menyediakan set karakter yang lebih besar dari satu byte. Jadi, jika Anda hendak memastikan untuk mengambil seluruh karakter, gunakan parameter `-c`. Contoh yang memperagakan parameter `-c` adalah mencari sepuluh set yang paling umum dari tiga karakter pertama dari sebuah

kata. Sebagian besar sistem UNIX menyediakan daftar kata yang dipisahkan dengan baris baru. Paket bsd-games pada Slackware Linux mengandung /usr/share/dict/words with dengan sebuah daftar kata. Kita bisa menggunakan **cut** untuk mendapatkan empat karakter pertama dari daftar kata, menambahkan *uniq* untuk menghitung urutan tiga karakter yang identik, dan menggunakan **sort** untuk mengurutkan secara terbalik (**sort** dijelaskan pada bagian bernama "Mengurutkan teks"). Akhirnya, kita akan menggunakan **head** untuk mendapatkan sepuluh urutan yang paling sering muncul:

```
$ cut -c 1-4 /usr/share/dict/words | uniq -c | sort -nr | head
    254 inte
    206 comp
    169 cons
    161 cont
    150 over
    125 tran
    111 comm
    100 disc
     99 conf
     96 reco
```

Setelah selesai dengan perintah UNIX ini, kita akan melanjutkan dengan perintah **paste** yang mengkombinasikan berkas dalam kolom pada satu aliran teks.

Penggunaan dari **paste** sangatlah sederhana. Perintah ini akan menggabungkan semua berkas yang diberikan sebagai argumen, dipisahkan dengan tab. Dengan daftar kata-kata Inggris dan Belanda, kita bisa menghasilkan sebuah kamus kecil:

```
$ paste dictionary-en dictionary-nl
apple   appel
banana  banaan
pear    peer
```

Anda juga bisa mengkombinasikan lebih dari dua berkas:

```
$ paste dictionary-en dictionary-nl dictionary-de
apple   appel   Apfel
banana  banaan  Banane
pear    peer    Birne
```

Jika salah satu berkas lebih besar, urutan kolom akan dipertahankan dan isi yang kosong akan digunakan untuk mengisi isi dari berkas yang lebih kecil.

Anda bisa menggunakan pembatas lain dengan menambahkan parameter *-d pembatas*. Sebagai contoh, kita bisa membuat kamus yang dipisahkan dengan titik dua:

```
$ paste -d ':' dictionary-en dictionary-nl
apple:appel
banana:banaan
pear:peer
```

Normalnya, **paste** mengkombinasikan berkas sebagai kolom yang berbeda. Anda bisa membuat **paste** menggunakan setiap baris dari setiap berkas sebagai kolom, dan meletakkan kolom dari setiap berkas pada baris yang berbeda. Hal ini dilakukan dengan parameter *-s*:

```
$ paste -s dictionary-en dictionary-nl dictionary-de
apple   banana  pear
appel   banaan  peer
Apfel   Banane  Birne
```

Mengurutkan teks

UNIX menawarkan perintah **sort** untuk mengurutkan teks. **sort** dapat juga menguji apakah sebuah berkas sudah diurutkan atau belum, dan menggabungkan dua berkas yang sudah diurutkan. **sort** bisa mengurutkan dalam urutan kamus atau numerik. Metode pengurutan default adalah berbasis kamus. Hal ini berarti baris teks akan dibandungkan per karakter, dan diurutkan sesuai dengan apa yang sudah ditentukan dalam urutan collation (yang ditentukan melalui variabel lingkungan LC_COLLATE). Hal ini memiliki masalah ketika Anda mengurutkan angka, misalnya jika Anda memiliki angka 1 hingga 10 pada baris yang berbeda, urutannya akan menjadi 1, 10, 2, 3, dst. Hal ini dikarenakan interpretasi per karakter dari pengurutan berdasarkan kamus. Jika Anda hendak mengurutkan baris berdasarkan angka, gunakan metode pengurutan berdasarkan numerik.

Jika tidak ada parameter tambahan yang ditentukan, **sort** mengurutkan baris masukkan dalam urutan seperti pada kamus. Sebagai contoh:

```
$ cat << EOF | sort
orange
apple
banana
EOF
apple
banana
orange
```

Seperti yang Anda lihat, masukkan diurutkan dengan benar. Seringkali terdapat dua baris yang sama. Anda bisa menggabungkan baris yang sama dengan menambahkan parameter *-u*. Dua contoh berikut akan mengilustrasikan hal ini.

```
$ cat << EOF | sort
orange
apple
banana
banana
EOF
apple
banana
banana
orange
$ cat << EOF | sort -u
orange
apple
banana
```

```
banana
EOF
apple
banana
orange
```

Terdapat beberapa parameter tambahan yang bisa berguna untuk memodifikasi hasil:

- Parameter `-f` membuat pengurutan menjadi case-insensitive.
- Jika `-d` ditambahkan, hanya karakter kosong dan alphanumerik yang digunakan untuk menentukan urutan.
- Parameter `-i` membuat **sort** mengabaikan karakter yang tidak bisa dicetak.

Anda bisa mengurutkan berkas secara numerik dengan menambahkan parameter `-n`. Parameter ini berhenti membaca baris masukan ketika karakter bukan angka ditemukan. Tanda minus, nilai desimal, pembatas ribuan, karakter radix (yang membatasi eksponen dari angka biasa), dan karakter kosong bisa digunakan sebagai bagian dari angka. Karakter-karakter ini akan diinterpretasikan jika dimungkinkan.

Contoh berikut menggambarkan pengurutan berbasiskan numerik, dengan melakukan pipe terhadap hasil keluaran dari **du** ke **sort**. Hal ini bisa bekerja karena **du** menentukan ukuran dari setiap berkas sebagai kolom pertama.

```
$ du -a /bin | sort -n
0       /bin/kernelversion
0       /bin/ksh
0       /bin/lsmod.modutils
0       /bin/lspci
0       /bin/mt
0       /bin/netcat
[...]
```

Pada kasus ini, hasil keluaran mungkin tidak berguna jika Anda hendak membaca keluaran dalam paginator, karena berkas terkecil akan ditampilkan terlebih dahulu. Kondisi ini dimana parameter `-r` akan berguna. Parameter ini akan memutar urutan pengurutan.

```
$ du -a /bin | sort -nr
4692    /bin
1036    /bin/ksh93
668     /bin/bash
416     /bin/busybox
236     /bin/tar
156     /bin/ip
[...]
```

Parameter `-r` juga bisa bekerja untuk pengurutan berdasarkan kamus.

Seringkali, berkas menggunakan layout dengan banyak kolom, dan Anda mungkin hendak mengurutkan berkas dengan kolom yang bukan kolom pertama. Sebagai contoh, misalkan ada berkas nilai bernama `score.txt`:

```
John:US:4
Herman:NL:3
```

```
Klaus:DE:5
Heinz:DE:3
```

Misalkan kita hendak mengurutkan berkas ini berdasarkan dua huruf nama negara. **sort** mengijinkan kita untuk mengurutkan berkas dengan sebuah kolom dengan parameter `-k col1[,col2]`. Dimana *col1* hingga *col2* digunakan sebagai kolom untuk mengurutkan masukan. Jika *col2* tidak ditentukan, semua kolom hingga akhir baris akan digunakan. Jadi jika Anda hendak menggunakan hanya satu kolom, gunakan `-k col1,col1`. Anda juga bisa menentkan karakter awal dengan kolom dengan menambahkan sebuah tanda titik (.) dan indeks karakter. Sebagai contoh, `-k 2.3,4.2` berarti kolom kedua dimulai dari karakter ketiga, kolom ketiga, dan kolom keempat hingga karakter kedua.

Terdapat hal lain yang berguna jika berhubungan dengan pengurutan kolom: secara default, **sort** menggunakan karakter kosong sebagai pembatas kolom. Jika Anda menggunakan karakter pemisah yang lain, Anda harus menggunakan parameter `-t char`, yang digunakan untuk menentukan pemisah kolom.

Dengan kombinasi parameter `-t` dan `-k`, kita bisa mengurutkan berkas nilai berdasarkan kode negara:

```
$ sort -t ':' -k 2,2 scores.txt
Heinz:DE:3
Klaus:DE:5
Herman:NL:3
John:US:4
```

Jadi, bagaimana kita bisa mengurutkan berkas berdasarkan skor? Jelas, kita harus meminta sort untuk menggunakan kolom ketiga. Tetapi sort menggunakan metode pengurutan berdasarkan kamus secara default [1]. Anda bisa menggunakan `-n`, tetapi **sort** juga mengijinkan pendekatan yang lebih canggih. Anda bisa menambahkan satu atau lebih *n*, *r*, *f*, *d*, *i*, atau *b* pada penanda kolom. Huruf-huruf ini merepresentasikan parameter **sort** dengan nama yang sama. Jika Anda menambahkan hanya kolom awal, menambahkan pada kolom tersebut, selain itu, tambahkan pada akhir kolom.

Perintah berikut mengurutkan berkas berdasarkan skor:

```
$ sort -t ':' -k 3n /home/daniel/scores.txt
Heinz:DE:3
Herman:NL:3
John:US:4
Klaus:DE:5
```

Disarankan untuk mengikuti pendekatan ini, daripada menggunakan varian parameter, karena **sort** mengijinkan Anda menggunakan lebih dari satu parameter `-k`. Dan menambahkan parameter ini pada spesifikasi kolom akan memungkinkan Anda untuk mengurutkan kolom yang berbeda dalam berbagai cara yang berbeda. Sebagai contoh, menggunakan **sort** dengan parameter `-k 3,3n -k 2,2` akan mengurutkan semua barus secara numerik oleh kolom ketiga. Jika beberapa baris memiliki angka yang sama pada kolom ketiga, baris ini bisa diurutkan dengan metode pengurutan berbasiskan kamus pada kolom kedua.

Jika Anda hendak menguji apakah sebuah berkas sudah diurutkan atau belum, Anda bisa menggunakan parameter `-c`. Jika berkas diurutkan sesuai urutan, sort akan mengembalikan nilai *0*, jika tidak maka *1*. Kita bisa menguji ini dengan menuliskan isi dari variabel *?* yang berisi nilai kembalian dari perintah terakhir yang dieksekusi. command.

[1] Tentu saja, hal ini bukan masalah, karena kita tidak menggunakan angka yang lebih besar dari 9, dan semua set karakter memiliki angka dalam urutan angka).

```
$ sort -c scores.txt ; echo $?
1
$ sort scores.txt | sort -c ; echo $?
0
```

Perintah kedua menunjukkan bahwa hal ini bekerja, dengan melakukan pipe hasil keluaran dari soft terhadap `scores.txt` ke **sort**.

Akhirnya, Anda bisa menggabungkan dua berkas yang sudah diurutkan dengan parameter *-m*, dengan tetap menjaga urutan pengurutan. Hal ini lebih cepat daripada menggabungkan kedua berkas, dan mengurutkannya kembali.

```
# sort -m scores-sorted.txt scores-sorted2.txt
```

Perbedaan antar berkas

Karena aliran teks, dan berkas teks sangatlah penting pada UNIX, seringkali sangat penting untuk menampikan perbedaan antara dua berkas. Utilitas utama untuk bekerja dengan perbedaan antar berkas adalah **diff** dan **patch**. **diff** menunjukkan perbedaan antar berkas. Hasil keluaran dari **diff** bisa diproses oleh **patch** untuk menerapkan perbedaan diantara kedua berkas pada sebuah berkas. “diffs” juga merupakan dasar dari version/source management systems. Bagian berikut juga menjelaskan **diff** dan **patch**. Untuk mendapatkan contoh yang dapat digunakan, dua berkas C berikut akan digunakan untuk mendemonstrasikan perintah ini. Berkas ini bernama `hello.c` dan `hello2.c`.

```
#include <stdio.h>

void usage(char *programName);

int main(int argc, char *argv[]) {
  if (argc == 1) {
    usage(argv[0]);
    return 1;
  }

  printf("Hello %s!\n", argv[1]);

  return 0;
}

void usage(char *programName) {
  printf("Usage: %s name\n", programName);
}
```

```
#include <stdio.h>
#include <time.h>

void usage(char *programName);
```

```
int main(int argc, char *argv[]) {
  if (argc == 1) {
    usage(argv[0]);
    return 1;
  }

  printf("Hello %s!\n", argv[1]);

  time_t curTime = time(NULL);
  printf("The date is %s\n", asctime(localtime(&curTime)));


  return 0;
}

void usage(char *programName) {
  printf("Usage: %s name\n", programName);
}
```

Menampilkan perbedaan antar berkas

Misalkan Anda menerima program `hello.c` dari seorang teman, dan Anda memodifikasi berkas tersebut untuk memberikan waktu dan jam aktual dari pengguna. Anda bisa saja mengirimkan program yang sudah diupdate ke teman Anda. Tetapi jika sebuah berkas sudah bertambah besar, hal ini menjadi tidak menyenangkan, karena perubahan menjadi lebih susah untuk dilacak. Selain itu, teman Anda juga mungkin menerima perubahan dari orang lain. Ini merupakan situasi yang sering terjadi dimana **diff** menjadi berguna.**diff** menampilkan perbedaan diantara kedua berkas. Sintaks dasarnya adalah **diff berkas berkas2**, yang akan menampilkan perbedaan antara `berkas` dan `berkas2`. Mari kita coba dengan berkas kita:

```
$ diff hello.c hello2.c
1a2 ❶
> #include <time.h> ❷
12a14,17
>   time_t curTime = time(NULL);
>   printf("The date is %s\n", asctime(localtime(&curTime)));
>
```

Tambahan pada `hello2.c` tampak pada hasil keluaran, tetapi formatnya sedikit aneh. sebenarnya, ini adalah perintah yang bisa diinterpretasikan oleh **ed**. Kita akan melihat hasil keluaran yang lebih nyaman setelah melihat format keluaran default.

Dua elemen yang berbeda bisa kita lihat dari keluaran ini:

❶ Ini adalah perintah ed yang menentukan teks yang harus ditambahkan (`a`) setelah baris `2`.

❷ Ini adalah teks aktual yang akan ditambahkan setelah baris kedua. Tanda “>” digunakan untuk menandai baris yang akan ditambahkan

Elemen yang sama digunakan untuk menambahkan blok kedua dari teks. Bagaimana dengan baris yang dihapus? Kita bisa dengan mudah melihat bagaimana mereka direpresentasikan dengan menukar dua parameter pada **diff**, menampilkan perbedaan antara `hello2.c` dan `hello.c`:

```
$ diff hello2.c hello.c
2d1 ❶
< #include <time.h> ❷
14,16d12
<   time_t curTime = time(NULL);
<   printf("The date is %s\n", asctime(localtime(&curTime)));
<
```

Elemen berikut dapat dibedakan:

❶ Ini adalah perintah penghapusan pada ed (`d`), menyatakan bahwa baris 2 harus dihapus. Perintah penghapusan kedua menggunakan interval (baris 14 hingga 17).
❷ Teks yang akan dihapus diawali dengan tanda “<”.

Tampaknya cukup untuk gaya keluaran-ed. Program GNU diff yang disertakan pada Slackware Linux mendukung apa yang disebut unified diff. Unified diff sangat mudah dibaca, dan menyediakan konteks secara default. **diff** bisa menyediakan hasil keluaran yang bersifat seragam dengan parameter *-u*:

```
$ diff -u hello.c hello2.c
--- hello.c     2006-11-26 20:28:55.000000000 +0100 ❶
+++ hello2.c    2006-11-26 21:27:52.000000000 +0100 ❷
@@ -1,4 +1,5 @@ ❸
 #include <stdio.h> ❹
+#include <time.h> ❺

 void usage(char *programName);

@@ -10,6 +11,9 @@

   printf("Hello %s!\n", argv[1]);

+  time_t curTime = time(NULL);
+  printf("The date is %s\n", asctime(localtime(&curTime)));
+
   return 0;
 }
```

Elemen-elemen berikut dapat ditemukan pada hasil keluaran

❶ Nama dari berkas asli, dan timestamp dari waktu modifikasi terakhir.
❷ Nama berkas yang berubah, dan timestamp dari waktu modifikasi terakhir.
❸ Dua angka ini menunjukkan lokasi dari ukuran dari teks yang mempengaruhi berkas asli dan berkas setelah modifikasi. Jadi, pada kasus ini, angka-angka ini berarti pada bagian yang berhubungan pada berkas asli dimulai pada baris 1, dan panjangnya empat baris. Pada berkas yang dimodifikasi, bagian yang berhubungan dimulai pada baris 1, dan panjangnya lima barus. Bagian yang berbeda pada hasil keluaran diff dimulai oleh header ini.
❹ Baris yang tidak diawali oleh tanda minus (-) atau plus (+) berarti tidak berubah. Baris yang tidak berubah disertakan karena mereka memberikan informasi kontekstual, dan untuk mencegah terlalu banyak bagian yang dibuat. Jika terdapat hanya beberapa baris yang tidak dimodifikasi pada setiap perubahan, **diff** akan memilih untuk membuat satu bagian saja.

❺ Baris yang diawali dengan tanda plus (+) adalah tambahan pada berkas yang dimodifiksi, dibandingkan dengan berkas asli.

Seperti dengan format **diff** gaya berbasis-ed, kita bisa melihat penghapusan dengan menukar nama berkas:

```
$ diff -u hello2.c hello.c

--- hello2.c    2006-11-26 21:27:52.000000000 +0100
+++ hello.c     2006-11-26 20:28:55.000000000 +0100
@@ -1,5 +1,4 @@
 #include <stdio.h>
-#include <time.h>

 void usage(char *programName);

@@ -11,9 +10,6 @@

   printf("Hello %s!\n", argv[1]);

-  time_t curTime = time(NULL);
-  printf("The date is %s\n", asctime(localtime(&curTime)));
-
   return 0;
 }
```

Seperti yang Anda lihat dari hasil keluaran ini, baris yang dihapus dari berkas yang dimodifikasi akan diawali dengan tanda minus (-).

Ketika Anda bekerja dengan sekumpulan berkas yang besar, akan sangat berguna untuk membandingkan seluruh direktori. Sebagai contoh, jika Anda memiliki versi asli dari kode program dalam sebuah direktori bernama `hello.orig`, dan versi modifikasi pada direktori bernama `hello`, Anda bisa menggunakan parameter *`-r`* untuk membandingkan kedua direktori secara rekursif. Sebagai contoh:

```
$ diff -ru hello.orig hello
diff -ru hello.orig/hello.c hello/hello.c

--- hello.orig/hello.c  2006-12-04 17:37:14.000000000 +0100
+++ hello/hello.c       2006-12-04 17:37:48.000000000 +0100
@@ -1,4 +1,5 @@
 #include <stdio.h>
+#include <time.h>

 void usage(char *programName);

@@ -10,6 +11,9 @@

   printf("Hello %s!\n", argv[1]);

+  time_t curTime = time(NULL);
+  printf("The date is %s\n", asctime(localtime(&curTime)));
```

```
+
   return 0;
 }
```

Harus diperhatikan bahwa hal ini hanya akan membandingkan berkas yang tersedia pada kedua direktori. Versi GNU dari diff yang digunakan oleh Slackware Linux menyediakan parameter *-N*. Parameter ini mengangap berkas yang ada pada salah satu dari kedua direktori seperti berkas kosong. Jadi sebagai contoh, kita telah menambahkan berkas bernama `Makefile` pada direktori `hello`, menggunakan parameter *-N* akan menghasilkan keluaran sebagai berikut:

```
$ diff -ruN hello.orig hello

diff -ruN hello.orig/hello.c hello/hello.c
--- hello.orig/hello.c  2006-12-04 17:37:14.000000000 +0100
+++ hello/hello.c       2006-12-04 17:37:48.000000000 +0100
@@ -1,4 +1,5 @@
 #include <stdio.h>
+#include <time.h>

 void usage(char *programName);

@@ -10,6 +11,9 @@

   printf("Hello %s!\n", argv[1]);

+  time_t curTime = time(NULL);
+  printf("The date is %s\n", asctime(localtime(&curTime)));
+
   return 0;
 }

diff -ruN hello.orig/Makefile hello/Makefile
--- hello.orig/Makefile 1970-01-01 01:00:00.000000000 +0100
+++ hello/Makefile      2006-12-04 17:39:44.000000000 +0100
@@ -0,0 +1,2 @@
+hello: hello.c
+       gcc -Wall -o $@ $<
```

Seperti yang Anda lihat, indikator menunjukkan bahwa bagian pada berkas asli dimulai pada baris 0 dan sepanjang 0 baris.

Pengguna UNIX seringkali bertukar hasil keluaran dari **diff**, yang sering disebut “diffs” atau “patches”. Bagian selanjutkan akan menunjukkan kepada Anda bagaimana Anda bisa menangani diffs. Tetapi sekarang Anda sudah mampu membuatnya sendiri, dengan mengarahkan hasil keluaran dari diff pada sebuah berkas. Sebagai contoh:

```
$ diff -u hello.c hello2.c > hello_add_date.diff
```

Jika Anda memiliki beberapa diffs, Anda bisa dengan mudah mengkombinasikannya menjadi satu diff:

```
$ cat diff1 diff2 diff3 > combined_diff
```

Tetapi pastikan bahwa mereka dibuat dari direktori yang sama jika Anda hendak menggunakan utilitas **patch** yang dibahas pada bagian selanjutnya.

Memodifikasi berkas dengan hasil keluaran diff

Misalkan seseorang mengirimkan hasil keluaran dari **diff** untuk berkas yang telah Anda buat. Akan sangat memakan waktu untuk menggabungkan semua perubahan yang dibuat. Untungnya, **patch** bisa melakukannya untuk Anda. **patch** menerima diff pada standard input, dan mencoba mengubah berkas asli, sesuai dengan perubahan yang didaftarkan pada diff. Jadi, misalnya kita memiliki berkas `hello.c`, dan patch yang dibuat sebelumnya berdasarkan perubahan antara `hello.c` dan `hello2.c`, kita bisa melakukan patch `hello.c` agar sama dengan berkas yang lain:

```
$ patch < hello_add_date.diff
patching file hello.c
```

Jika Anda memiliki `hello2.c`, Anda bisa menguji apakah berkas sudah sama:

```
$ diff -u hello.c hello2.c
```

Tidak ada hasil keluaran, jadi ini hasilnya. Salah satu fitur menarik dari **patch** adalah ia dapat mengembalikan perubahan yang dilakukan oleh diff, dengan menggunakan parameter *-R*:

```
$ patch -R < hello_add_date.diff
```

Pada contoh ini, berkas asli sudah di-patch. Seringkali, Anda hendak menerapkan patch pada berkas dengan nama yang berbeda. Anda bisa melakukan hal ini dengan menyediakan nama dari berkas sebagai argumen terakhir:

```
$ patch helloworld.c < hello_add_date.diff
patching file helloworld.c
```

Anda juga bisa menggunakan **patch** dengan diffs yang dibuat dengan parameter *-r*, tetapi Anda harus hati-hati. Misalkan header dari berkas diff seperti berikut:

```
---------------------------
|diff -ruN hello.orig/hello.c hello/hello.c
|--- hello.orig/hello.c 2006-12-04 17:37:14.000000000 +0100
|+++ hello/hello.c      2006-12-04 17:37:48.000000000 +0100
---------------------------
```

Jika Anda memproses diff ini dengan **patch**, maka program akan mencoba mengubah `hello.c`. Jadi, direktori yang berisi berkas ini harus berada dalam direktori aktif. Anda bisa menggunakan nama path lengkap dengan *-p n*, dimana *n* adalah angka path dimana komponen harus dihilangkan. Nilai *0* akan menggunakan path seperti yang ditentukan

pada patch, *1* akan menghapus nama path pertama dari komponen, dan seterusnya. Pada contoh ini, menghapus komponen pertama akan melakukan patch terhadap `hello.c`. Sesuai dengan standar Single UNIX Specification versi 3, path yang diawali dengan --- harus digunakan untuk membangun berkas yang harus di-patch. Patch versi GNU tidak mengikuti standar disini. Jadi, sangat disarankan untuk menghapus pada titik dimana baik nama direktori sudah sama (hal ini biasanya direktori utama dari pohon yang sedang diganti). Pada sebagian besar kasus dimana path relatif digunakan, hal ini bisa diselesaikan menggunakan *-p 1*. Sebagai contoh:

```
$ cd hello.orig
$ patch -p 1 < ../hello.diff
```

Atau, Anda bisa menggunakan parameter *-d* untuk menentukan direktori dimana perubahan harus diterapkan:

```
$ patch -p 1 -d hello.orig < hello.diff
patching file hello.c
patching file Makefile
```

Jika Anda hendak mempertahankan backup ketika Anda mengubah berkas, Anda bisa menggunakan paramater *-b* dari **patch**. Parameter ini akan membuat salinan dari setiap berkas yang diganti dengan nama `filename.orig`, sebelum mengubah berkas:

```
$ patch -b < hello_add_date.diff
$ ls -l hello.c*
-rw-r--r-- 1 daniel daniel 382 2006-12-04 21:41 hello.c
-rw-r--r-- 1 daniel daniel 272 2006-12-04 21:12 hello.c.orig
```

Seringkali sebuah berkas tidak bisa di-patch. Misalnya, jika sudah pernah di-patch, terlalu banyak perubahan sehingga tidak bisa dipatch dengan sempurna, atau jika berkas tidak ada. Pada kasus ini, bagian yang tidak bisa disimpan akan disimpan pada berkas dengan nama `namaberkas.rej`, dimana *namaberkas* adalah berkas yang coba dimodifikasi oleh **patch**.

9.2. Regular expressions

Perkenalan

Pada kehidupan sehari-hari, Anda seringkali ingin agar beberapa teks cocok pada beberapa pola. Banyak utilitas UNIX yang mengimplementasikan sebuah bahasa untuk pencocokan pola teks *regular expressions* (regexps). Seiring dengan waktu, bahasa regular expression telah berkembang, sehingga saat ini sudah terdapat tiga sintaks untuk regular expression:

- Traditional UNIX regular expressions.
- POSIX extended regular expressions.
- Perl-compatible regular expressions (PCRE).

POSIX regexps adalah superset dari traditional UNIX regexps, dan PCRE adalah superset dari POSIX regexps. Sintaks yang didukung oleh setiap aplikasi bisa berbeda-beda, tetapi hampir semua aplikasi mendukung paling tidak POSIX regexps.

Setiap unit sintatis pada regexp merepresentasikan salah satu dari hal berikut:

- **Sebuah karakter**: ini adalah dasar dari setiap regular expression, sebuah karakter atau sekumpulan karakter yang harus dicocokkan. Sebagai contoh, kata *p* atau tanda ,.
- **Kuantifikasi**: Sebuah quantifier menentukan berapa banyak karakter atau set karakter yang harus dicocokkan.
- **Alternatif**: Alternatif digunakan untuk mencocokkan “a atau b” dimana *a* dan *b* bisa berupa karakter atau regexp.
- **Pengelompokkan**: hal ini digunakan untuk mengelompokkan sub ekspresi, sehingga kuantifikasi atau alternatif bisa diterapkan pada grup.

Traditional UNIX regexps

Bagian ini membahas traditional UNIX regexps. Karena kurangnya standarisasi, sintaks yang sebenarnya bisa berbeda untuk setiap utilitas. Biasanya, halaman manual dari sebuah perintah menyediakan lebih banyak informasi tentang regular expressions yang didukung. Merupakan ide bagus untuk mempelajari traditional regexps, tetapi untuk menggunakan POSIX regexps untuk script Anda sendiri.

Pencocokkan karakter

Karakter akan dicocokkan dengan sendirinya. Jika karakter tertentu digunakan sebagai sintaks karakter untuk regexps, Anda bisa mencocokkan karakter tersebut dengan menambahkan sebuah backslash. Sebagai contoh, \+ cocok dengan karakter plus.

Tanda titik (.) cocok dengan sembarang karakter, sebagai contoh, regexp *b.g* cocok dengan *bag*, *big*, dan *blg*, tetapi tidak dengan *bit*.

Karakter titik seringkali memberikan kebebasan yang berlebihan. Anda bisa memberikan tanda kurung siku (*[]*) untuk menentukan karakter yang harus dicocokkan. Sebagai contoh, regexp *b[aei]g* cocok dengan *bag*, *beg*, dan *big*, tetapi tidak yang lain. Anda juga bisa mencocokkan sembarang karakter kecuali karakter pada sebuah set menggunakan kurung siku, dan menggunakan tanda caret (^) sebagai karakter pertama. Sebagai contoh, *b[^aei]g* cocok dengan sembarang tiga karakter string yang dimulai dengan *b* dan diakhiri dengan *g*, dengan pengecualian *bag*, *beg*, dan *big*. Juga dimungkinkan untuk mencocokkan sebuah interval karakter dengan tanda minus (-). Sebagai contoh, *a[0-9]* cocok dengan a diikuti dengan sebuah angka.

Dua karakter khusus, caret (^) dan dollar (*$*), cocok dengan awal dan akhir baris. Karakter ini sangat berguna untuk mem-parsing berkas. Sebagai contoh, Anda bisa mencocokan semua berkas yang dimulai dengan tanda hash (#) dengan regexp ^#.

Kuantifikasi

Tanda kuantifikasi paling sederhana yang didukung oleh traditional regular expressions adalah tanda bintang (*). Tanda ini cocok dengan nol atau lebih dari karakter yang mendahuluinya. Sebagai contoh, *ba** cocok dengan *b*, *babaa*, dst. Anda harus hati-hati bahwa sebuah karakter diikuti dengan tanda bintang tanpa konteks apapun cocok dengan semua string, karena *c** juga cocok dengan string yang memiliki nol karakter *c*.

Repetisi lebih spesifik bisa ditentukan dengan tanda kurung kurawal yang di-escape *\{x,y\}* cocok dengan karakter yang mengawli minimal *x* kali, tetapi tidak lebih dari *y* kali. Jadi, *ba\{1,3\}* cocok dengan *ba*, *baa*, dan *baaa*.

Pengelompokkan

Tanda kurung yang di-escape akan mengelompokkan sekumpulan karakter bersama-sama, sehingga Anda bisa menerapkan kuantifikasi atau alternatif pada sekumpulan karakter. Sebagai contoh, *\(ab\)\{1,3\}* cocok dengan *ab*, *abab*, dan *ababab*.

Alternatif

Sebuah pipe yang di-escape (\|) mengijinkan Anda untuk mencocokkan salah satu ekspresi. Hal ini tidak berguna untuk satu karakter, karena *a\|b* ekuivalen dengan *[ab]*, tetapi sangat berguna pada saat pengelompokkan. Misalkan Anda hendak sebuah ekspresi yang cocok dengan *apple* dan *pear*, tetapi bukan yang lain. Hal ini bisa dilakukan dengan mudah dengan : *(apple)|(pear)*.

POSIX extended regular expressions

POSIX regular expressions dibangun diatas traditional regular expressions, dengan menambahkan beberapa primitif yang berguna. Perbedaan lain adalah tanda kurung pengelompokkan, tanda kuantifikasi, dan tanda alternatif (|) tidak perlu di escape dengan backslash. Jika mereka di-escape, maka akan cocok dengan karakter literal biasa, sehingga menghasilkan tingkah laku yang kebalikan dari traditional regular expressions. Sebagian besar orang menganggap POSIX extended regular expressions lebih nyaman, sehingga lebih banyak digunakan.

Pencocokkan karakter

Pencocokkan karakter biasa tidak berubah dibandingkan dengan traditional regular expressions yang dibahas pada bagian bernama “Pencocokkan karakter”

Kuantifikasi

Selain tanda star (*), yang cocok dengan nol atau lebih karakter yang mendahuluinya, POSIX extended regular expressions menambahkan dua primitif kuantifikasi. Tanda plus (+) cocok dengan satu atau lebih karakter atau grup sebelumnya. Sebagai contoh, *a+*, cocok dengan *a* (atau sembarang string dengan lebih dari satu *a*), tetapi tidak cocok dengan nol karakter *a*. Karakter tanda tanya (*?*) cocok dengan nol atau satu karakter yang mendahuluinya. Jadi, *ba?d* cocok dengan *bd* dan *bad*, tetapi tidak untuk *baad* atau *bed*.

Kurung kurawal juga digunakan untuk repetisi, seperti pada traditional regular expressions, meskipun backslask bisa diabaikan. Untuk mencocokkan *ba* dan *baa*, Anda harus menggunakan *ba{1,2}* dan bukan *ba\{1,2\}*.

Pengelompokkan

Pengelompokkan dilakukan dengan cara yang sama dengan traditional regular expressions, kecuali Anda tidak perlu melakukan escape sebelum tanda kurung. Sebagai contoh, *(ab){1,3}* cocok dengan *ab*, *abab*, dan *abababab*.

Alternatif

Alternatif dilakukan dengan cara yang sama dengan traditional regular expressions, kecuali Anda tidak perlu melakukan escape sebelum tanda pipe. Jadi, *(apple)|(pear)* cocok dengan *apple* and *pear*.

9.3. grep

Penggunaan grep dasar

Kita telah tiba pada salah satu utilitas paling penting pada Sistem UNIX, dan salah satu yang mencoba untuk menggunakan regular expressions. Perintah **grep** digunakan untuk mencari aliran teks atau berkas untuk sebuah pola. Pola disini adalah regular expression, dan bisa berupa regular expression dasar atau POSIX extended regular expression (ketika parameter *-E* digunakan). Secara default, **grep** akan mencoba menulis baris yang cocok pada standard output. Pada sintaks dasarnya, Anda bisa menentukan regular expression sebagai sebuah argumen, dan **grep** akan mencari kecocokkan pada teks dari standard input. Hal ini merupakan cara yang baik untuk berlatih dengan regular expressions.

```
$ grep '^\(ab\)\{2,3\}$'
ab
abab
abab
ababab
abababab
```

Contoh diatas menunjukkan basic regular expression, yang cocok dengan sebuah baris tunggal yang berisi dua atau tiga string *ab*. Anda juga bisa melakukan hal yang sama dengan POSIX extended regular expressions, dengan menambahkan parameter `-E` (untuk extended):

```
$ grep -E '^(ab){2,3}$'
ab
abab
abab
ababab
abababab
```

Karena tingkah laku default dari **grep** adalah membaca dari standard input, Anda bisa menambahkan ke pipe untuk mendapatkan bagian yang menarik dari hasil keluaran dari perintah sebelumnya. Sebagai contoh, jika Anda hendak mencari string *2006* pada kolom ketiga dari berkas, Anda bisa mengkombinasikan perintah **cut** dan **grep**:

```
$ cut -f 3 | grep '2006'
```

Melakukan grep pada berkas

Secara umum, **grep** juga bisa membaca dari berkas, dan bukan dari standard input. Seperti biasa, hal ini bisa dilakukan dengan menambahkan berkas yang hendak dibaca sebagai argumen terakhir. Contoh berikut akan mencetak semua baris dari berkas `/etc/passwd` dari awal hingga string *daniel:*.

```
$ grep "^daniel" /etc/passwd
daniel:*:1001:1001:Daniel de Kok:/home/daniel:/bin/sh
```

Dengan opsi `-r`, **grep** akan menjelajahi sebuah struktur direktori secara rekursif, mencoba mencari pencocokkan pada setiap berkas yang ditemukan selama pencarian. Meski demikian, akan lebih baik untuk mengkombinasikan **grep** dengan **find** dan operan *-exec* pada script agar lebih portabel.

```
$ grep -r 'somepattern' somedir
```

adalah fungsi yang ekuivalen namun tidak portabel dari

```
$ find /somedir -type f -exec grep 'somepattern' {} \; -print
```

Perilaku pola

grep juga bisa mencetak semua barus yang tidak cocok dengan pola yang digunakan. Hal ini bisa dilakukan dengan menambahkan parameter `-v`:

```
$ grep -Ev '^(ab){2,3}$'
ab
ab
abab
ababab
abababab
abababab
```

Jika Anda hendak menggunakan pola dalam pola case-insensitive, Anda bisa menambahkan parameter `-i`. Sebagai contoh:

```
$ grep -i "a"
a
a
A
A
```

Anda juga mencocokkan string secara literal dengan parameter `-F`:

```
$ grep -F 'aa*'
a
aa*
aa*
```

Menggunakan banyak pola

Seperti yang telah kita lihat, Anda bisa menggunakan karakter alternatif (|) untuk mencocokkan dua atau lebih sub pola. Jika dua pola yang hendak Anda cocokkan berbeda jauh, akan lebih enak untuk dua pola yang berbeda. **grep** mengijinkan Anda untuk menggunakan lebih dari satu pola dengan memisahkannya dengan baris baru. Sehingga, misalnya Anda hendak mencetak baris yang cocok dengan pola *a* atau *b*, hal ini bisa dilakukan dengan mudah dengan menambahkan baris baru:

```
$ grep 'a
b'
a
a
b
b
c
```

Hal ini bisa bekerja, karena tanda kutip akan digunakan, dan shell mengirimkan parameter yang dikuti secara literal. Meskipun, harus diakui bahwa hal ini tidaklah bagus. **grep** menerima satu atau lebih parameter *-e pattern*, memberikan kesempatan lebih besar untuk menentukan satu atau lebih parameter pada satu barus. Pemanggilan **grep** pada contoh sebelumnya bisa ditulis sebagai:

```
$ grep -e 'a' -e 'b'
```

Bab 10. Manajemen Proses

10.1. Teori

Proses

Sebuah instansiasi dari program yang berjalan disebut *proses*. Setiap proses memiliki memori terproteksinya masing-masing, yang disebut *alamat ruang proses*. Alamat ini berisi dua area: *area teks* dan *area data*. Area teks berisi kode program sebenarnya, dan memberitahukan kepada sistem apa yang harus dilakukan. Area data menyimpan konstanta dan data runtime dari sebuah proses. Karena terdapat banyak proses pada sebuah sistem, dan hanya satu atau beberapa prosessor, kernel sistem operasi membagi waktu prosesor diantara proses. Proses ini disebut dengan *time-sharing*.

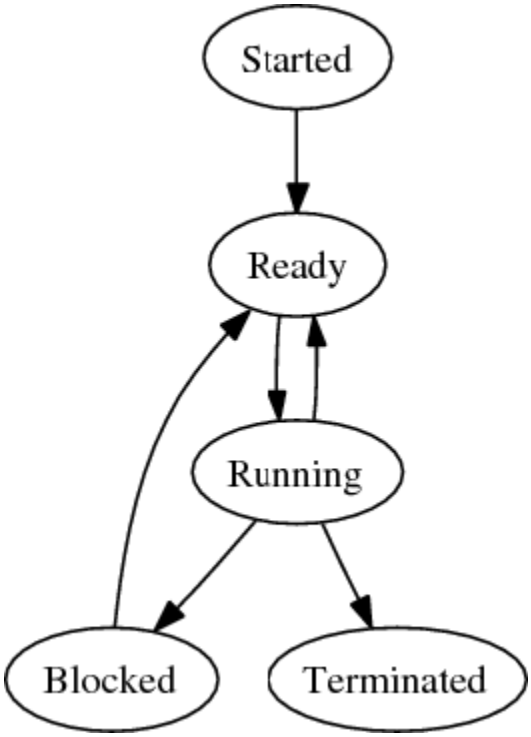
Tabel 10.1. Struktur sebuah proses

Kolom	Deskripsi
pid	Penanda numerik proses
ppid	Penanda proses untuk proses induk
euid	ID pengguna efektif dari proses
ruid	ID pengguna sebenarnya dari proses
egid	ID grup dari proses
rgid	ID grup sebenarnya dari proses
fd	Penunjuk ke daftar file descriptor yang dipakai
vmspace	Penunjuk ke alamat ruang proses

Tabel 10.1, “Struktur sebuah proses” menampilkan kolom informasi yang paling penting yang disimpan oleh kernel tentang sebuah proses. Setiap proses dapat diidentifikasi secara unik dengan *PID* (identifier proses), yang merupakan sebuah angka tidak bertanda. Seperti yang akan kita lihat nanti, seorang pengguna dapat dengan mudah mendapatkan PID dari sebuah proses. Setiap proses dihubungkan dengan sebuah *UID* (ID pengguna) dan *GID* (ID grup) pada sistem. Setiap proses memiliki *real UID*, yang merupakan UID yang digunakan ketika proses dijalankan, dan *effective UID*, yang merupakan UID yang digunakan untuk menjalankan proses. Biasanya, effective UID sama dengan UID, tetapi beberapa program meminta sistem untuk mengganti effective UID. Effective UID menentukan kontrol akses. Hal ini berarti bahwa jika seorang pengguna bernama joe menjalankan sebuah perintah, misalkan less, less hanya bisa membuka berkas dimana joe memiliki hak untuk membaca. Secara paralel, sebuah proses juga memiliki *real GID* dan *effective GID*.

Banyak proses akan membuka berkas, handle yang digunakan untuk beroperasi pada sebuah berkas disebut *file descriptor*. Kernel mengelola daftar file descriptor yang dibuka untuk setiap proses. Kolom *fd* berisi penunjuk pada daftar berkas yang dibuka. Kolom *vmspace* menunjuk pada ruang alamat proses dari proses.

Gambar 10.1. Status proses



Tidak setiap proses membutuhkan waktu CPU pada waktu tertentu. Sebagai contoh, beberapa proses mungkin menunggu untuk beberapa operasi *I/O* (Input/Output) untuk bisa selesai atau mungkin dihentikan. Status proses biasanya *started*, *running*, *ready* (untuk mulai), *blocked* (menunggu untuk I/O), atau *terminated*. Gambar 10.1, “Status proses” menampilkan daur hidup dari sebuah proses. Sebuah proses yang berada dalam status terminated, tetapi isi tabel proses tidak diambil kembali sering disebut dengan *proses zombie*. Proses zombie seringkali berguna untuk mengijinkan proses induk membaca status keluar dari proses, atau menyiapkan isi tabel proses secara sementara.

Membuat proses baru

Proses baru dibuat dengan system call *fork()*. System call ini menyalin ruang alamat proses dan informasi proses dari pemanggil, dan memberikan proses baru, yang disebut proses anak yang memiliki ID yang berbeda. Proses anak akan melanjutkan eksekusi dari titik yang sama dengan proses induk, tetapi mendapatkan nilai kembalian yang berbeda dari system call *fork()*. Berdasarkan dari nilai kembalian ini, kode dari induk dan anak bisa menentukan bagaimana melanjutkan proses eksekusi. Potongan kode C berikut menunjukan pemanggilan *fork()* :

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main() {
  pid_t pid = fork();
  if (pid == 0)
    printf("Hi, I am the child!\n");
  else
    printf("Hi, I am the parent, the child PID is %d!\n", pid);

  return 0;
```

```
}
```

. Program sederhana ini memanggil *fork()*, menyimpan nilai kembalian dari *fork()* pada variabel *pid*. *fork()* mengembalikan nilai kembalian *0* pada anak, dan PID dari anak kepada induk. Pada kasus ini, kita bisa menggunakan struktur kondisional sederhana untuk menguji nilai dari variabel *pid*, dan mencetak pesan yang benar.

Anda mungkin akan heran bagaimana mungkin memulai proses baru, karena pemanggilan *fork()* menduplikasi proses yang ada. Itu merupakan pertanyaan yang bagus, karena dengan *fork()* saja, tidak mungkin untuk menjalankan program baru. Kernel UNIX juga menyediakan sekumpulan system call, dimulai dari *exec*, yang memuat citra program pada proses aktual. Kita sudah lihat pada awal bab ini bahwa sebuah proses adalah program yang berjalan -- sebuah proses dibuat pada memori dari citra program yang disimpan pada media penyimpanan. Jadi, keluarga system call *exec* memberikan proses yang berjalan sebuah fasilitas untuk mengganti kontennya dengan program yang disimpan pada media tertentu. Dengan sendirinya, hal ini tidak terlalu berguna, karena setiap kali pemanggilan *exec* selesai dilakukan, kode pemanggil sebelumnya (atau program) akan dihapus dari proses. Hal itu bisa dibuktikan dari program C berikut:

```
#include <stdio.h>
#include <unistd.h>

int main() {
  execve("/bin/ls", NULL, NULL);

  /* This will never be printed, unless execve() fails. */
  printf("Hello world!\n");

  return 0;
}
```

Program ini mengeksekusi **ls** dengan pemanggilan *execve()*. Pesan yang dicetak dengan *printf()* tidak akan ditampilkan, karena citra program yang berjalan digantikan dengan **ls**. Meski demikian, kombinasi dari fungsi *fork()* dan *exec* sangatlah hebat. Sebuah proses bisa melakukan fork terhadap dirinya sendiri, dan membiarkan proses anak "mengorbankan" dirinya untuk menjalankan program lain. Program berikut mendemonstrasikan pola ini:

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main() {
  pid_t pid = fork();
  if (pid == 0)
    execve("/bin/ls", NULL, NULL);

  printf("Hello world!");

  return 0;
}
```

Program ini melakukan form terhadap dirinya sendiri. Citra program dari proses anak akan digantikan dengan **ls**, Sementara proses induk mencetak pesan "Hello world!" ke layar dan selesai.

Prosedur ini diikuti dengan banyak program, termasuk shell, ketika sebuah perintah dieksekusi dari prompt shell. Pada kenyataannya, semua proses pada sistem UNIX secara langsung atau tidak langsung diturunkan dari proses *init*, yang merupakan program pertama yang dijalankan pada sistem UNIX.

Thread

Meskipun fork sangat berguna untuk membangun paralelisme[1], hal ini bisa menjadi mahal untuk beberapa tujuan. Menyalin seluruh proses membutuhkan waktu, dan terdapat biaya yang dilibatkan jika proses hendak berbagi data. Hal ini diselesaikan dengan menawarkan alternatif yang lebih ringan, atau dengan kata lain mengijinkan lebih dari satu thread eksekusi. Setiap thread eksekusi dieksekusi secara terpisah, tetapi data proses dibagi antar thread.

Menulis program *multithreaded* yang bagus membutuhkan pengetahuan yang baik akan konsep berbagi data dan proses penguncian. Karena semua data dipakai bersama-sama, pemrograman yang tidak benar bisa menimbulkan kesalahan seperti race conditions.

10.2. Menganalisa proses yang berjalan

Mendata proses yang berjalan

Sistem UNIX menyediakan perintah **ps** untuk menampilkan daftar proses yang berjalan. Sayangnya, perintah ini adalah contoh dari kurangnya standarisasi. Varian BSD dan System V dari **ps** memiliki opsinya masing-masing. Untungnya, GNU/Linux mengimplementasikan parameter-parameter pada System V dan BSD-style, dan juga beberapa opsi (gaya-GNU). Opsi yang diawali dengan tanda dash akan diinterpretasikan sebagai opsi milik System V dan opsi tanpa dash adalah opsi milik BSD. Kami akan menjelaskan opsi System V pada bagian ini.

Jika **ps** digunakan tanpa parameter apapun, perintah ini akan menampilkan semua proses yang dimiliki oleh pengguna yang memanggil **ps** dan semua yang berhubungan pada terminal yang dipanggil. Sebagai contoh:

```
$ ps
  PID TTY          TIME CMD
 8844 pts/5    00:00:00 bash
 8862 pts/5    00:00:00 ps
```

Banyak informasi berguna yang bisa didapatkan dari hasil ini. Seperti yang Anda lihat, dua proses akan ditampilkan: shell yang digunakan untuk memanggil **ps** (*bash*), dan perintah **ps** itu sendiri. Pada kasus ini, terdapat empat kolom informasi. *PID* adalah ID proses dari proses, *TTY* terminal yang mengontrol, *TIME* jumlah waktu CPU yang telah digunakan oleh proses, dan **CMD** perintah atau program dari salinan yang berjalan. Kolom yang ditampilkan secara default bisa berbeda-beda untuk setiap sistem, tetapi pailng tidak kolom-kolom berikut akan ditampilkan, dengan label kolom yang berbeda.

Seringkali Anda hendak melihat tampilan yang lebih luas dari proses yang sedang berjalan. Menambahkan opsi `-a` akan menampilkan semua proses yang dihubungkan dengan terminal. Sebagai contoh:

```
$ ps -a
  PID TTY          TIME CMD
 7487 pts/1    00:00:00 less
 8556 pts/4    00:00:10 emacs-x
```

[1]Misalnya, web server bisa melakukan form banyak proses anak untuk menangani permintaan

```
11324 pts/3    00:00:00 ps
```

Seperti yang Anda lihat, proses-proses dengan terminal pengontrol yang berbeda akan ditampilkan. Meskipun, dibandingkan dengan hasil keluaran dari **ps**, hanya proses-proses yang mengontrol terminal yang akan ditampilkan. Sebagai contoh, shell yang digunakan untuk memanggil **ps** tidak ditampilkan.

Anda juga bisa mencetak semua proses yang sedang berjalan, termasuk proses yang tidak berhubungan dengan terminal, dengan menggunakan opsi `-A` :

```
$ ps -A | head -n 10
  PID TTY          TIME CMD
    1 ?        00:00:01 init
    2 ?        00:00:00 migration/0
    3 ?        00:00:00 ksoftirqd/0
    4 ?        00:00:00 watchdog/0
    5 ?        00:00:00 migration/1
    6 ?        00:00:00 ksoftirqd/1
    7 ?        00:00:00 watchdog/1
    8 ?        00:00:00 events/0
    9 ?        00:00:00 events/1
```

Anda bisa mencetak semua proses dengan ID pengguna tertentu dengan opsi *-U*. Opsi ini menerima nama pengguna sebagai parameter, atau banyak nama pengguna yang dipisahkan dengan koma. Perintah berikut akan menampilkan semua proses yang memiliki ID pengguna *xfs* atau *rpc* :

```
$ ps -U xfs,rpc
  PID TTY          TIME CMD
 2409 ?        00:00:00 portmap
 2784 ?        00:00:00 xfs
```

Anda juga bisa mencetak proses-proses dengan ID grup tertentu, dengan opsi *-G* :

```
$ ps -G messagebus,haldaemon
  PID TTY          TIME CMD
 8233 ?        00:00:00 dbus-daemon
11312 ?        00:00:00 hald
11320 ?        00:00:00 hald-addon-keyb
11323 ?        00:00:00 hald-addon-acpi
```

Jika Anda hendak mendapatkan terminal fisik atau pseudo, Anda bisa menggunakan opsi `-t` :

```
$ ps -t tty2
  PID TTY          TIME CMD
 2655 tty2     00:00:00 getty
```

10.3. Mengelola proses

Mengirimkan sinyal ke proses

Sinyal adalah bentuk kasar, tetapi efektif dari komunikasi antar proses. Sebuah sinyal pada dasarnya adalah sebuah angka yang dikirimkan pada sebuah proses yang memiliki arti khusus. Untuk semua sinyal, terdapat handle untuk sinyal default. Proses dapat menginstall handle untuk penanganan sinyalnya, atau memilih untuk mengabaikan sinyal. Beberapa sinyal (biasanya SIGKILL dan SIGSTOP) tidak dapat diabaikan. Semua sinyal memiliki nama simbol yang lebih nyaman untuk digunakan.

Hanya beberapa sinyal yang biasanya menarik untuk penggunaan interaktif pada sistem berbasis UNIX. Mereka adalah (diikuti oleh nomornya):

- *SIGKILL (9):* memaksa menghentikan sebuah proses.
- *SIGTERM (15):* meminta proses untuk terminate. Karena ini merupakan sebuah permintaan, sebuah program bisa mengabaikan sinyal ini, kebalikan dengan *SIGKILL*.
- *SIGHUP (1):* Secara tradisional, sinyal ini akan memaksa sebuah terminal untuk selesai. Tetapi beberapa daemon (cth. *inetd*) membaca ulang konfigurasinya ketika sinyal ini dikirimkan.

Perintah **kill** digunakan untuk mengirimkan sinyal pada proses. Secara default, **kill** mengirimkan sinyal *SIGTERM*. Untuk mengirimkan sinyal ini, ID proses dari proses yang hendak Anda kirimi sinyal ini harus ditambahkan sebagai parameter. Sebagai contoh :

```
$ kill 15631
```

Untk mengirimkan sinyal lain, Anda bisa menggunakan salah satu dari dua opsi : *-nomorsinyal* atau *-namasinyal*. Jadi, perintah berikut akan mengirimkan sinyal *SIGKILL* pada proses dengan ID proses *15631*:

```
$ kill -9 15631
```

```
$ kill -SIGKILL 15631
```

Berlaku baik buat yang lain

Anda bisa berlaku baik bagi pengguna lain dalam hal sumber daya komputer. Jika Anda berencana untuk membangun proses yang memakan waktu CPU yang intensif, tetapi tidak ingin menganggu pekerjaan pengguna lain dari sistem (atau proses lain), Anda bisa memberikan beberapa tingkat 'kebaikan' pada sebuah proses. Secara praktis, hal ini berarti Anda bisa mempengaruhi prioritas penjadwalan dari proses. Proses yang lebih baik mendapatkan prioritas yang lebih rendah. Kebaikan dari proses secara normal adalah *0*, dan bisa diganti dengan menjalankan program dengan perintah **nice**. Opsi *-n [kebaikan]* bisa digunakan untk menentukan tingkat kebaikan:

```
$ nice -n 20 cputimewaster
```

Nilai maksimal untuk kebaikan sangat tergantung dari implementasi. Jika program dijalankan dengan **nice**, tetapi tidak ada nilai yang diberikan, maka akan diset sebagai *10*. Jika Anda heran: ya, Anda bisa saja bertindak semena-mena,

tetapi hak ini hanya terbatas untuk pengguna *root*. Anda bisa meningkatkan prioritas dari proses dengan menentukan nilai negatif.

Anda juga bisa memodifikasi proses yang berjalan dengan perintah **renice**. Hal ini bisa dilakukan untuk ID proses tertentu (`-p PIDs`), pengguna (`-u user/uid`), dan grup efektif (`-g group/gid`). Nilai baru ditentukan sebagai parameter pertama.

Tingkat kebaikan dari proses hanya bisa dinaikkan. Dan, tentu saja tidak ada pengguna lain selain *root* yang dapat mempengaruhi tingkat kebaikan proses dari pengguna lain.

Mari kita lihat pada contoh, untuk menentukan tingkat kebaikan dari sebuah proses dengan PID *3108* menjadi *14*, Anda bisa menggunakan perintah berikut :

```
$ renice 14 -p 3108
```

10.4. Kontrol pekerjaan

Seringkali penting untuk mengelompokkan proses-proses untuk mengijinkan operasi pada sekumpulan proses, misalnya mendistribusikan sebuah sinyal untuk semua proses pada sebuah grup dan bukan pada satu proses saja. Tidak heran, sekumpulan proses ini disebut *program groups* pada UNIX. Setelah sebuah fork, sebuah proses anak secara otomatis merupakan anggota dari process group dari induk. Meski demikian, process groups baru bisa dibuat dengan membuat satu proses menjadi pemimpin, dan menambahkan proses lain pada grup. ID process group adalah PID dari pemimpin process group.

Secara virtual, semua shell UNIX modern memberikan process group pada proses yang dibuat melalui pemanggilan sebuah perintah. Semua proses pada pipeline biasanya ditambahkan pada satu process group. Contoh berikut yang membuat sebuah pipeline akan dieksekusi.

```
cat | tr -s ' ' | egrep 'foob.r'
```

shell akan menjalankan langkah-langkah berikut:

1. Tiga proses akan di-fork.
2. Proses pertama pada pipeline diletakkan pada process group dengan PIDnya sendiri sebagai ID process group, membuatnya menjadi pemimpin. Proses lain dari pipeline akan ditambahkan pada process group.
3. File descriptors dari proses pada pipeline akan dikonfigurasi ulang untuk membentuk sebuah pipeline.
4. Program pada pipeline akan dieksekusi.

Shell menggunakan process groups untuk mengimplementasikan *kontrol pekerjaan*. Sebuah shell dapat menjalankan banyak pekerjaan dibelakang layar, bisa terdapat banyak pekerjaan yang dihentikan, dan satu pekerjaan dapat berjalan di depan (foreground). Sebuah job yang berjalan dihubungkan ke terminal untuk standard input (berarti merupakan pekerjaan yang mendapatkan masukan dari pengguna).

Menghentikan dan melanjutkan pekerjaan

Sebuah pekerjaan yang berada di foreground (pekerjaan yang menerima masukan pengguna dari terminal) dapat dihentikan dengan menekan *Ctrl-z* (menekan tombol kunci 'Ctrl' dan 'z' secara bersamaan). Hal ini akan mengentikan

pekerjaan, dan memberikan kontrol atas terminal kepada shell. Mari kita coba dengan perintah **sleep**, yang menunggu sejumlah waktu dalam detik sesuai dengan yang disediakan sebagai argumen:

```
$ sleep 3600
Ctrl-z
[1]+  Stopped                 sleep 3600
```

Process group, yang kita rujuk sebagai sebuah pekerjaan telah dihentikan, yang berarti **sleep** sudah berhenti - eksekusinya benar-benar berhenti. Anda bisa mendapatkan daftar pekerjaan dengan perintah **jobs** :

```
$ jobs
[1]+  Stopped                 sleep 3600
```

Hal ini menampilkan angka pekerjaan (*1*), statusnya, dan perintah yang digunakan untuk memulai pekerjaan ini. Mari kita jalankan program lain, hentikan program tersebut, dan lihat pada daftar pekerjaan.

```
$ cat
Ctrl-z
[2]+  Stopped                 cat
$ jobs
[1]-  Stopped                 sleep 3600
[2]+  Stopped                 cat
```

Seperti yang diharapkan, pekerjaan kedua juga dihentikan, dan diberikan angka pekerjaan *2*. Tanda plus (+) mengikuti pekerjaan pertama telah berubah menjadi tanda minus (-), sedangkan pekerjaan kedua sekarang ditandai dengan tanda plus. Tanda plus digunakan untuk mengindikasikan *pekerjaan aktual*. Perintah **bg** dan **fg** yang akan kita lihat sebentar lagi, akan beroperasi pada pekerjaan aktual jika tidak ada pekerjaan yang diberikan sebagai parameter.

Biasanya, ketika Anda bekerja dengan pekerjaan, Anda hendak memindahkan pekerjaan pada foreground kembali. Hal ini dilakukan dengan perintah **fg**. Menjalankan **fg** tanpa parameter akan memindahkan pekerjaan aktual pada foreground. Sebagian besar shell akan mencetak perintah yang dipindahkan ke foreground untuk memberikan indikator proses apa yang dipindahkan :

```
$ fg
cat
```

Tentu saja, tidak selalu berguna untuk meletakkan pekerjaan aktual pada foreground. Anda bisa meletakkan pekerjaan lain pada foreground dengan menambahkan nomor pekerjaan diawali dengan tanda persen (%) sebagai argumen dari **fg**:

```
$ fg %1
sleep 3600
```

Berpindah pekerjaan akan menghentikan pekerjaan aktual dan meletakkannya pada foreground seringkali berguna ketika shell digunakan secara interaktif. Sebagai contoh, misalkan Anda sedang mengedit sebuah berkas dengan editor

teks, dan hendak menjalankan perintah lain lalu kembali mengedit. Anda bisa menghentikan editor dengan *Ctrl-z*, menjalankan sebuah perintah, dan meletakkan editor kembali dengan **fg**.

Pekerjaan dibelakang

Selain menjalankan program pada foreground, pekerjaan juga bisa dijalankan dibelakang (background). Hal ini berarti mereka tetap berjalan, tetapi masukkan dari terminal tidak dialihkan ke proses background. Sebagian besar shell mengkonfigurasikan dirinya untuk mengeluarkan hasil keluaran pada shell dimana mereka pertama kali dijalankan.

Sebuah proses yang dihentikan bisa dilanjutkan di background dengan perintah **bg** :

```
$ sleep 3600

[1]+  Stopped                 sleep 3600
$ bg
[1]+ sleep 3600 &
$
```

Anda bisa melihat bahwa pekerjaan memang benar-benar berjalan dengan **jobs**:

```
$ jobs
[1]+  Running                 sleep 3600 &
```

Seperti **fg**, Anda bisa memindahkan pekerjaan lain selain pekerjaan aktual ke background dengan menentukan angka pekerjaannya :

```
$ bg %1
[1]+ sleep 3600 &
```

Anda juga bisa meletakkan sebuah pekerjaan langsung ke background ketika dijalankan, dengan menambahkan tanda ampersand (&) pada perintah atau pipeline. Sebagai contoh :

```
$ sleep 3600 &
[1] 5078
```

Bagian III. Pengeditan dan typesetting

Daftar Isi

Bab 11. LaTeX

11.1. Perkenalan

LaTeX adalah sistem typesetting yang dapat digunakan untuk membuat artikel, buku, surat, dan publikasi lain berkualitas tinggi. LaTeX berbasiskan pada TeX, bahasa typesetting aras bawah yang didesain oleh Donald E. Knuth. LaTeX tidak bekerja seperti pengolah kata WYSIWYG (what you see is what you get), jenis persiapan dokumen yang sudah banyak dipakai oleh banyak orang. Dengan LaTeX, Anda tidak harus perduli dengan pemformatan dokumen, hanya tentang penulisan dokumen.

Berkas LaTeX adalah berkas plain-text yang berisi makro LaTeX macros. LaTeX memformat dokumen berdasarkan makro yang digunakan. Pada awalnya, menggunakan LaTeX mungkin akan membingungkan bagi pengguna baru. Tetapi setelah beberapa waktu, Anda akan menemukan bahwa menggunakan LaText memiliki beberapa keuntungan. Sebagian diantaranya:

- Dokumen yang diformat dengan LaTeX tampak profesional.
- Anda tidak perlu perduli dengan layout dari dokumen Anda. Anda cukup menambahkan struktur pada dokumen Anda, dan LaTeX mengurus bagian pemformatan.
- Berkas LaTeX cuma plain text, dan bisa diganti menggunakan utilitas standar UNIX, seperti **vi**, **sed** atau **awk**
- LaTeX menyediakan dukungan yang sangat bagus untuk hal-hal yang berhubungan dengan typesetting seperti formula matematika, referensi, dan citra Postscript.

LaTeX sangatlah luas, sehingga bab ini hanya membahas permukaan dari LaTeX. Tetapi sudah cukup untuk memulai membuat dokumen sederhana.

11.2. Mempersiapkan dokumen LaTeX sederhana

Struktur dokumen minimal

Setiap dokumen LaTeX memiliki struktur dasar minimal yang menjelaskan dokumen. Mari kita lihat sebuah contoh:

```
\documentclass[10pt,a4paper]{article} ❶

\title{The history of symmetric ciphers} ❷
\author{John Doe} ❸

\begin{document} ❹

This is a basic document. ❺

\end{document} ❻
```

Anda sudah melihat struktur sintaks dasar dari perintah LaTeX. Sebuah perintah dimulai dengan sebuah backslash, diikuti dengan nama perintah. Setiap makro memiliki argumen wajib yang diletakkan dalam kurung kurawal, dan argumen opsional yang diletakkan pada kurang siku.

❶ Perintah pertama dari setiap dokumen adalah *documentclass*. Perintah ini menentukan jenis dokumen apa yang sedang dihadapi oleh LaTeX. Jenis dokumen ditentukan sebagai parameter wajib. Anda juga bisa menentukan parameter opsional, seperti ukuran font dan kertas. Pada kasus ini, ukuran font diganti dari 12pt menjadi 10pt, dan A4 digunakan sebagai ukuran kertas. Kelas dokumen yang ada pada LaTeX ditunjukkan pada Tabel 11.1, “Kelas dokumen LaTeX”.

❷ Setelah documentclass Anda bisa menambahkan beberapa informasi meta pada dokumen, misalnya judul dokumen. Pada kasus ini, judulnya adalah *The history of symmetric ciphers*.

❸ Perintah author menentukan penulis buku.

❹ Perintah *\begin* memulai awal dari lingkungan penulisan. Terdapat banyak lingkungan penulisan, tetapi mereka memiliki konvensi yang berbeda untuk setiap teks yang berada pada lingkungan tersebut. Pada kasus ini, kita memulai lingkungan *document*. Ini merupakan lingkungan dasar, LaTeX menginterpretasikan semuanya pada lingkungan ini sebagai isi dari teks.

❺ Isi dokumen dapat diletakkan didalam *document*, pada kasus ini sebuah pesan peringatan tentang isi dari dokumen.

❻ Semua lingkungan harus ditutup. Lingkungan *document* adalah lingkungan terakhir yang harus ditutup, karena menandai akhir dari isi teks.

Tabel 11.1. Kelas dokumen LaTeX

Kelas
article
book
letter
report

Menghasilkan format siap cetak

Setelah Anda memiliki berkas LaTeX, Anda bisa menggunakan perintah **latex** untuk menghasilkan berkas DVI (Device Independent format) :

```
$ latex crypto.tex
This is pdfeTeX, Version 3.141592-1.21a-2.2 (Web2C 7.5.4)
entering extended mode
(./crypto.tex
LaTeX2e <2003/12/01>
Babel <v3.8d> and hyphenation patterns for american, french, german, ngerman, b
ahasa, basque, bulgarian, catalan, croatian, czech, danish, dutch, esperanto, e
stonian, finnish, greek, icelandic, irish, italian, latin, magyar, norsk, polis
h, portuges, romanian, russian, serbian, slovak, slovene, spanish, swedish, tur
kish, ukrainian, nohyphenation, loaded.
(/usr/share/texmf/tex/latex/base/article.cls
Document Class: article 2004/02/16 v1.4f Standard LaTeX document class
(/usr/share/texmf/tex/latex/base/size10.clo)) (./crypto.aux) [1] (./crypto.aux)
 )
 Output written on crypto.dvi (1 page, 248 bytes).
 Transcript written on crypto.log.
```

As the LaTeX command reports a DVI file is created after running the **latex** command. You can view this file with an X viewer for DVI files, **xdvi**:

```
$ xdvi crypto.dvi
```

Berkas ini tidak langsung dapat dicetak (meskipun berkas DVI dapat dicetak dengan **xdvi**). Sebuah format yang ideal untuk berkas yang siap cetak adalah Postscript. Anda bisa menghasilkan berkas Postscript dari berkas DVI dengan satu perintah sederhana:

```
$ dvips -o crypto.ps crypto.dvi
```

Parameter *-o* menentukan hasil keluaran (berkas) untuk dokumen Postscript. Jika parameter tidak ditentukan, hasil keluaran akan di-pipe melalui **lpr**, yang akan menjadwalkan dokumen untuk dicetak.

PDF (Portable Document Format) adalah format lain yang populer untuk dokumen elektronik. PDF dapat dengan mudah dibuat dari Postscript:

```
$ ps2pdf crypto.ps
```

Hasil keluaran akan sama dengan berkas masukkan, dengan ekstensi `.ps` digantikan dengan **.pdf**.

Seksi

Sekarang Anda sudah tahu bagaimana membuat dokumen LaTeX sederhana, merupakan ide bagus untuk menambahkan beberapa struktur tambahan. Struktur ditambahkan dengan menggunakan seksi, sub seksi, dan sub sub seksi. Elemen struktural ini dibuat dengan perintah *section*, *subsection* dan *subsubsection*. Parameter wajib untuk sebuah seksi adalah judul untuk seksi. Seksi normal, sub seksi dan sub sub seksi secara otomatis diberi nomor, dan ditambahkan pada daftar isi. Dengan menambahkan tanda bintang setelah perintah section, misalnya *\section*{title}* penomoran seksi akan berkurang, dan seksi tersebut tidak akan ditambahkan pada daftar isi. Contoh berikut mendemonstrasikan bagaimana Anda bisa menggunakan seksi:

```
\documentclass[10pt,a4paper]{article}

\title{The history of symmetric ciphers}
\author{John Doe}

\begin{document}

\section{Pre-war ciphers}

To be done.

\section{Modern ciphers}

\subsection*{Rijndael}

Rijndael is a modern block cipher that was designed by Joan Daemen and
Vincent Rijmen.

In the year 2000 the US National Institute of Standards and Technologies
selected Rijndael as the winner in the contest for becoming the Advanced
Encryption Standard, the successor of DES.
```

```
\end{document}
```

Contoh diatas cukup jelas, tetapi merupakan waktu yang tepat untuk melihat bagaimana LaTeX memperlakukan karakter baris dan baris kosong. Baris kosong akan diabaikan oleh LaTeX, membuat teks menjadi aliran yang mengalir. Sebuah baris kosong memulai sebuah paragraf baru. Semua paragraf kecuali paragraf pertama dimulai dengan dengan tambahan ruang di sisi kiri kata pertama.

Gaya font

Biasanya Anda ingin bekerja dengan berbagai gaya font juga. LaTeX memiliki beberapa perintah yang bisa digunakan untuk mengubah tampilan dari font aktual. Perintah font yang paling sering digunakan adalah *emph* untuk teks miring, dan *textbf*. Lihat Tabel 11.2, “Gaya font LaTeX” untuk daftar gaya font yang lebih ekstensif. Teks miring dan tebal didemonstrasikan pada paragraf contoh berikut:

```
Working with font styles is easy. \emp{This text is emphasized} and
\textbf{this text is bold}.
```

Tabel 11.2. Gaya font LaTeX

Perintah	Deskripsi
\emph	Menambahkan emphasis pada font.
\textbf	Mencetak teks dalam huruf tebal.
\textit	Menggunakan font italic.
\textsl	Menggunakan font yang di-slant.
\textsc	Menggunakan huruf kecil.
\texttt	Menggunakan font typewriter.
\textsf	Menggunakan font sans-serif.

Bagian IV. Surat elektronik (E-Mail)

Daftar Isi

Bab 12. Membaca dan menulis e-mail dengan mutt

12.1. Perkenalan

Mutt adalah sebuah mail user agent (MUA) yang bisa digunakan untuk membaca dan menulis e-mail. Mutt adalah installasi berbasis teks, yang berarti aplikasi ini hanya bisa digunakan pada konsol, melalui SSH dan pada terminal X. Karena antarmuka berbasis menu, maka aplikasi ini sangat mudah untuk membaca banyak e-mail dalam waktu yang singkat, dan mutt bisa dikonfigurasi untuk menggunakan editor teks favorit Anda.

Bab ini akan mendiskusikan bagaimana Anda bisa mengkustomisasi mutt untuk kebutuhan Anda, bagaimana menggunakannya, dan bagaimana dukungan PGP/GnuPG digunakan.

12.2. Penggunaan

Mutt sangatlah mudah digunakan, meskipun mungkin membutuhkan sedikit waktu untuk membiasakan diri dengan kunci yang digunakan untuk melakukan navigasi, membaca dan menulis e-mail. Beberapa bagian berikutnya menjelaskan sebagian dari banyak kunci yang penting. Mutt menyediakan gambaran tentang kunci yang tersedia yang lebih lengkap setelah menekan kunci <h>.

Melihat daftar e-mail

Setelah memanggil perintah **mutt**, sebuah gambaran dari semua e-mail akan muncul. Anda bisa melihat daftar e-mail dengan kunci panah atas dan bawah, atau kunci <k> dan <j>.

Membaca e-mails

Untuk membaca sebuah e-mail, gunakan kunci <Enter>, setelah memilih sebuah e-mail pada daftar. Ketika membaca sebuah e-mail, Anda bisa menggunakan <Page Up> dan <Page Down> untuk menjelajahi semua e-mail. Anda masih bisa menggunakan kunci navigasi yang digunakan untuk menjelajahi semua e-mail untuk melihat e-mail lain.

Jika sebuah e-mail memiliki attachment, Anda bisa melihatnya dengan menekan kunci <v>. Anda bisa melihat masing-masing attachment dengan memilihnya dan menekan kunci <Enter>. Untuk menyimpan ke sebuah berkas, tekan kunci <s>.

Mengirimkan e-mail

Anda bisa membuat sebuah e-mail baru dengan kunci <m>, atau membalas e-mail dengan <r>. Mutt akan meminta Anda menentukan penerima (*To:*), dan subyek (*Subject:*). Setelah memasukkan informasi ini, sebuah editor akan dijalankan (**vi** digunakan secara default), yang bisa Anda gunakan untuk membuat e-mail. Setelah menyimpan e-mail, dan keluar dari editor, mutt akan memberikan kesempatan kepada Anda untuk membuat perubahan lain pada e-mail. Jika Anda memutuskan untuk mengubah e-mail, Anda bisa menjalankan ulang editor dengan kunci <e>. Anda bisa mengganti penerima atau subyek dengan <t> atau <s>. Akhirnya, Anda bisa mengirimkan e-mail dengan menekan <y>. Jika Anda hendak membatalkan e-mail, tekan <q>. Mutt akan menanyakan apakah Anda hendak menunda e-mail. Jika Anda melakukannya, Anda akan diberikan kesempatan untuk melanjutkan pembuatan pesan di lain waktu.

12.3. Pengaturan sederhana

Terdapat beberapa pengaturan mutt yang pasti Anda konfigurasi. Bagian ini menjelaskan beberapa pengaturan ini. Kustomisasi mutt yang spesifik untuk pengguna dapat dibuat pada berkas `.muttrc` pada direktori home pengguna. Anda bisa mengganti pengaturan mutt secara global pada `/etc/mutt/Muttrc`.

Header yang dikustomisasi

Setiap e-mail memiliki header dengan banyak informasi. Sebagai contoh, header berisi informasi tentang jalur yang telah dilalui e-mail setelah dikirimkan. Informasi pengirim (*From:*) dan penerima (*To:*) juga disimpan pada header, begitu juga dengan subyek (*Subject:*) e-mail.

Catatan

Kenyataannya, header *To:* tidak digunakan untuk menentukan tujuan dari e-mail selama proses pengiriman e-mail. MTA menggunakan *envelope address* untuk menentukan tujuan dari e-mail. Meski demikian, sebagian besar MUA menggunakan alamat *To:* yang diisi oleh penulis sebagai alamat surat.

Anda bisa menambahkan header Anda sendiri pada e-mail dengan opsi konfigurasi *my_hdr*. Opsi ini memiliki sintaks sebagai berikut: *my_hdr <nama header>: <isi header>*. Sebagai contoh, Anda bisa menambahkan informasi tentang sistem operasi yang Anda gunakan dengan menambahkan baris berikut pada konfigurasi mutt Anda:

```
my_hdr X-Operating-System: Slackware Linux 10.2
```

Anda juga bisa menimpa beberapa header yang biasa digunakan, seperti alamat pengirim yang ditentukan pada header *From:*:

```
my_hdr From: John Doe <john.doe@example.org>
```

Biner sendmail

Secara default, mutt menggunakan MTA sendmail untuk mengirimkan e-mail yang akan dikirimkan. Anda bisa menggunakan perintah lain untuk mengirimkan e-mail dengan mengubah variabel konfigurasi *sendmail*. Pengganti sendmail harus mampu menangani sintaks parameter yang sama dengan sendmail. Sebagai contoh, jika Anda telah menginstall MSMTP untuk mengirimkan e-mail, Anda bisa mengkonfigurasikan mutt untuk menggunakannya dengan menambahkan baris berikut pada konfigurasi mutt:

```
set sendmail="/usr/bin/msmtp"
```

Ketika Anda sudah sepenuhnya mengganti sendmail dengan MTA lain, misalnya Postfix, pada umumnya tidak lagi diperlukan untuk menentukan parameter ini, karena sebagian besar MTA menyediakan alternatif terhadap berkas biner sendmail.

12.4. Menggunakan IMAP

Biasanya, mutt membaca -email dari kotak surat lokal pengguna. Namun, mutt juga mendukung penggunaan kotak surat IMAP. IMAP (the Internet Message Access Protocol) adalah sebuah protokol yang digunakan untuk mengakses

e-mail dari server remote, dan didukung oleh banyak server e-mail. Mutt menggunakan format URL berikut untuk merepresentasikan server IMAP:

imap://[user@]hostname[:port]/[mailbox]

Atau format berikut untuk IMAP melalui SSL/TLS:

imaps://[user@]hostname[:port]/[mailbox]

Anda bisa langsung menggunakan sintaks ini pada operasi yang berhubungan dengan direktori. Sebagai contoh, jika Anda menekan “c” untuk mengubah dari direktori, Anda bisa memasukkan URL IMAP. Hal ini cukup merepotkan, sehingga lebih mudah untuk menyimpan informasi pada berkas `.muttrc`.

Jika Anda henya menggunakan satu akun IMAP, Anda bisa menentukan direktori INBOX dari akun ini sebagai kotak surat spool, dan akun utama IMAP sebagai direktori e-mail. Sebagai contoh, menambahkan baris berikut pada berkas konfigurasi `.muttrc` akan membuat mutt untuk log in pada server *imap.example.org* sebagai pengguna *me*.

```
set folder=imap://me@imap.example.org/
set spoolfile=imap://me@imap.example.org/INBOX
```

12.5. Menandatangani/mengenkripsi e-mail

Perkenalan

Mutt menyediakan dukungan yang bagus untuk menandatangani atau mengenkripsi e-mail dengan GnuPG. Seseorang mungkin heran kenapa dia harus menggunakan salah satu teknik ini. Meskipun sebagian orang tidak merasakan perlunya mengenkripsi e-mail mereka, merupakan ide yang bagus untuk menandatangani e-mail Anda. Terdapat banyak virus yang menggunakan alamat e-mail orang lainpada kolom From:. Jika orang yang Anda ajak berkomunikasi tahu bahwa Anda menandatangani e-mail Anda, mereka tidak akan membuka e-mail palsu dari virus. Selain itu, akan tampak lebih profesional jika orang lain bisa menguji identitas Anda, terutama pada transaksi bisnis. Sebagai contoh, siapa yang lebih Anda percaya vampire_boy93853@hotmail.com, atau seseorang dengan alamat e-mail profesional dengan email yang ditandatangani secara digital?

Bagian ini menjelaskan bagaimana Anda bisa menggunakan GnuPG dengn mutt, untuk informasi lebih lanjut tentang GnuPG baca Bagian 8.9, “Mengenkripsi dan menandatangani berkas”.

Konfigurasi

Sebuah contoh konfigurasi untuk menggunakan GnuPG pada mutt dapat ditemukan pada `/usr/share/doc/mutt/samples/gpg.rc`. Pada umumnya, isi berkas ini cukup untuk menjadi konfigurasi mutt Anda. Dari shell, Anda bisa menambahkan isi berkas ini pada `.muttrc` dengan perintah berikut:

```
$ cat /usr/share/doc/mutt/samples/gpg.rc >> ~/.muttrc
```

Terdapat beberapa parameter yang berguna yang bisa Anda tambahkan. Sebagai contoh, jika Anda selalu ingin untuk menandatangani e-mail, tambahkan baris berikut pada konfigurasi mutt Anda:

```
set crypt_autosign = yes
```

Opsi lain yang cukup berguna adalah *crypt_replyencrypt*, yang akan secara otomatis mengenkripsi balasan pesan yang dienkripsi. Untuk mengaktifkan opsi ini, tambahkan baris berikut pada konfigurasi mutt Anda:

```
set crypt_replyencrypt = yes
```

Penggunaan

Jika Anda telah menentukan beberapa opsi otomatisasi, seperti *crypt_autosign*, penggunaan GnuPG pada mutt biasanya sudah otomatis. Jika tidak, Anda bisa menekan kunci <p> selama langkah terakhir dari pengiriman e-mail. Pada bagian bawah layar, beberapa opsi GnuPG/PGP akan muncul, yang bisa Anda akses dengan huruf yang berada didalam tanda kurung. Sebagai contoh, <s> menandatangani e-mail, dan <e> mengenkripsi e-mail. Anda bisa menghapus opsi GnuPG dengan menekan <p> lalu <c>.

Bab 13. Sendmail

13.1. Perkenalan

Sendmail adalah Mail Transfer Agent (MTA) default yang digunakan Slackware Linux. sendmail awalnya dibuat oleh Eric Allman, yang masih mengelola sendmail. Peran utama dari MTA sendmail adalah mengirimkan pesan, baik secara lokal maupun remote. Pengiriman biasanya dilakukan melalui protokol SMTP. Hal ini berarti sendmail bisa menerima e-mail dari situs remote melalui port SMTP, sendmail mengirimkan e-mail yang ditujukan untuk situs remote ke server SMTP lain.

13.2. Installasi

Sendmail tersedia sebagai paket *sendmail* pada set disk "n". Jika Anda hendak membuat berkas konfigurasi sendmail Anda sendiri, paket *sendmail-cf* juga diperlukan. Untuk informasi tentang bagaimana menginstall paket pada Slackware Linux, lihat Bab 17, *Manajemen paket*.

Anda bisa membuat Slackware Linux menjalankan sendmail setiap kali boot dengan membuat berkas `/etc/rc.d/rc.sendmail` memiliki bit executable. Anda bisa melakukannya dengan :

```
# chmod a+x /etc/rc.d/rc.sendmail
```

Anda bisa menjalankan, menghentikan, dan menjalankan ulang sendmail dengan menggunakan *start*, *stop*, dan *restart* sebagai parameter pada script inisialisasi sendmail. Sebagai contoh, Anda bisa menjalankan ulang sendmail dengan cara berikut:

```
# /etc/rc.d/rc.sendmail restart
```

13.3. Konfigurasi

Berkas konfigurasi utama sendmail adalah `/etc/mail/sendmail.cf`; berkas ini adalah tempat dimana semua pengaturan sendmail dikonfigurasi, dan dimana berkas lain disertakan. Sintaks dari `/etc/mail/sendmail.cf` sedikit aneh, karena berkas ini dikompilasi dari berkas `.mc` yang lebih sederhana yang menggunakan makro M4 yang didefinisikan untuk sendmail.

Beberapa definisi bisa dengan mudah diganti pada berkas `/etc/mail/sendmail.cf`, tetapi untuk perubahan lain, akan lebih baik untuk membuat berkas `.mc` Anda sendiri. Contoh pada bab ini akan berfokus pada pembuatan berkas mc yang terkustomisasi.

Bekerja dengan berkas mc

Pada bagian ini, kita akan melihat bagaimana Anda bisa mulai dengan sebuah berkas mc, dan bagaimana mengkompilasi berkas .mc Anda menjadi .cf. Terdapat banyak contoh berkas mc yang tersedia pada `/usr/share/sendmail/cf/cf`. Contoh yang paling menarik adalah `sendmail-slackware.mc` (yang digunakan untuk membuat `sendmail.cf` deault untuk Slackware Linux), dan `sendmail-slackware-tls.mc` yang menambahkan dukungan TLS pada konfigurasi sendmail Slackware Linux standar. Jika Anda hendak membuat konfigurasi sendmail Anda sendiri, merupakan ide yang bagus untuk mulai dengan salinan dari berkas mc standar milik

Slackware Linux. Sebagai contoh, misalkan kita hendak membuat berkas konfigurasi untuk server bernama *straw*, kita bisa menjalankan :

```
# cd /usr/share/sendmail/cf/cf
# cp sendmail-slackware.mc sendmail-straw.mc
```

dan mulai mengedit `sendmail-straw.mc`. Setelah berkas konfigurasi dimodifikasi sesuai selera, M4 bisa digunakan untuk mengkompilasi berkas mc menjadi .cf:

```
# m4 sendmail-straw.mc > sendmail-straw.cf
```

Jika kita hendak menggunakan berkas konfigurasi ini sebagai konfigurasi default, kita bisa menyalinnya ke `/etc/mail/sendmail.cf`:

```
# cp sendmail-straw.cf /etc/mail/sendmail.cf
```

Menggunakan smarthost

Jika Anda hendak menggunakan host lain untuk mengirimkan e-mail ke lokasi dimana server sendmail Anda tidak bisa mengirimkan, Anda bisa mengatur sendmail untuk menggunakan apa yang disebut “smart host”. Sendmail akan mengirimkan pesan e-mail yang tidak bisa dikirimkan ke smart host, yang akan menangani pengiriman e-mail. Anda bisa melakukannya dengan mendefinisikan *SMART_HOST* pada berkas mc Anda. Sebagai contoh, jika Anda menggunakan *smtp2.example.org* sebagai smart host, Anda bisa menambahkan baris berikut:

```
define(`SMART_HOST',`stmp2.example.org')
```

Nama host/domain alternatif

Secara default, sendmail akan menerima pesan yang ditujukan untuk localhost, dan nama host aktual dari sistem. Anda bisa menambahkan host atau domain tambahan yang mampu untuk menerima e-mail. Langkah pertama adalah memastikan bahwa baris berikut ditambahkan pada konfigurasi mc Anda:

```
FEATURE(`use_cw_file')dnl
```

Ketika opsi ini diaktifkan, Anda bisa menambahkan nama host dan domain yang mampu menerima mail ke `/etc/mail/local-host-names`. Berkas ini adalah basis data daftar nama yang dipisahkan oleh baris baru. Sebagai contoh, berkasnya bisa seperti berikut:

```
example.org
mail.example.org
www.example.org
```

Tabel pengguna virtual

Seringkali Anda harus memetakan alamat e-mail pada nama pengguna. Hal ini dibutuhkan ketika nama pengguna berbeda dengan bagian sebelum “@” dari alamat e-mail. Untuk mengaktifkan fungsi ini, pastikan baris berikut ditambahkan pada berkas mc Anda:

```
FEATURE(`virtusertable',`hash -o /etc/mail/virtusertable.db')dnl
```

Pemetaan akan membaca dari `/etc/mail/virtusertable.db`. Ini merupakan berkas basis data biner yang tidak boleh diedit secara langsung. Anda bisa mengedit `/etc/mail/virtusertable`, dan menghasilkan `/etc/mail/virtusertable.db` dari berkas tadi.

Berkas `/etc/mail/virtusertable` adalah berkas teks plain. Berkas ini memiliki pemetaan untuk setiap baris, dengan alamat e-mail dan nama pengguna dipisahkan oleh tab. Sebagai contoh:

```
john.doe@example.org    john
john.doe@mail.example.org        john
```

Pada contoh ini, e-mail untuk *john.doe@example.org* dan *john.doe@mail.example.org* akan dikirimkan ke akun *john*. Juga dimungkinkan untuk mengirimkan beberapa e-mail ke domain yang diletakkan pada server ke alamat e-mail lain, dengan menentukan alamat e-mail untuk mengirimkan pesan pada kolom kedua. Sebagai contoh

```
john.doe@example.org    john.doe@example.com
```

Setelah melakukan perubahan yang diperlukan pada berkas `virtusertable` Anda bisa menghasilkan basis data dengan perintah berikut:

```
# makemap hash /etc/mail/virtusertable < /etc/mail/virtusertable
```

Bagian V. Administrasi sistem

Daftar Isi

Bab 14. Manajemen Pengguna

14.1. Perkenalan

GNU/LINUX adalah sistem operasi multi pengguna. Ini berarti banyak pengguna dapat menggunakan sistem, dan mereka dapat menggunakan sistem tersebut secara bersamaan. Konsep manajemen pengguna dari GNU/Linux cukup sederhana. Pertama, terdapat beberapa account pengguna pada setiap sistem.Bahkan untuk sistem yang memiliki satu pengguna sekalipun terdapat beberapa account pengguna, karena GNU/Linux menggunakan account yang unik untuk melakukan beberapa tugas. Setiap pengguna dapat menjadi anggota dari sebuah grup. Grup digunakan untuk untuk perijinan yang lebih luas. Sebagai contoh, Anda dapat membuat berkas yang hanya bisa dibaca oleh grup tertentu. Dalam sebuah sistem terdapat beberapa pengguna dan grup yang namanya telah dialokasikan. Diantara nama pengguna yang dialokasikan tersebut, yang paling penting adalah *root*. Pengguna *root* merupakan administrator dari sistem. Adalah hal yang baik untuk menghindari login sebagai root, karena dengan login sebagai root akan meningkatkan resiko keamanan. Anda dapat login sebagai pengguna biasa, kemudian melakukan pekerjaan administrasi sistem menggunakan perintah **su** dan perintah **sudo**

Account pengguna yang tersedia diletakkan di `/etc/passwd`. Anda dapat melihat pada berkas ini untuk menentukan account pengguna mana yang akan diperintah. Satu hal yang perlu dicatat, pada berkas ini tidak disertakan password pengguna. Password pengguna dipisah ke sebuah berkas yang telah dienkripsi yaitu `/etc/shadow`. Informasi untuk grup disimpan di `/etc/group`. Mengubah isi berkas ini secara langsung merupakan ide yang buruk. Terdapat beberapa aplikasi tangguh yang digunakan untuk melakukan administrasi pengguna dan grup. Bab ini akan menjelaskan beberapa dari aplikasi tersebut.

14.2. Menambah dan mengurangi pengguna

useradd

Perintah **useradd** digunakan untuk menambah account pengguna ke sistem. Menjalankan perintah **useradd** dengan menambahkan nama pengguna sebagai parameter akan menciptakan pengguna di dalam sistem. Contoh:

```
# useradd bob
```

Menciptakan account pengguna *bob*. Untuk Anda ketahui dengan perintah diatas tidak akan menciptakan direktori home untuk pengguna tersebut. Tambahkan parameter `-m` parameter untuk menciptakan direktori home. Contoh :

```
# useradd -m bob
```

Perintah ini akan menambah pengguna *bob* ke dalam sistem, dan menciptakan direktori home untuk pengguna ini yaitu `/home/bob`. Normalnya sebuah pengguna langsung menjadi anggota dari grup *users*. Misalkan kita ingin mengubah keanggotaan dengan membuat *crew* menjadi grup primer dari pengguna *bob*. Hal ini dapat dilakukan dengan menambahkan `-g` parameter. Contoh:

```
# useradd -g crew -m bob
```

Selain itu dimungkinkan juga untuk menambah grup sekunder dalam proses pembuatan account pengguna tersebut dengan dengan menambah opsi *-G*. Nama-nama grup sekunder lainnya dapat dipisahkan menggunakan koma. Contoh

perintah di bawah ini akan membuat pengguna *bob*, yang memiliki grup primer dari grup*crew*, dan *www-admins* dan *ftp-admins* sebagai grup sekundernya:

```
# useradd -g crew -G www-admins,ftp-admins -m bob
```

Secara default perintah **useradd** hanya membuat pengguna, dan tidak membuat password dalam proses pembuatan pengguna. Password dapat di buat menggunakan perintah **passwd**

passwd

Mungkin Anda telah menebak lebih dulu perintah **passwd** digunakan untuk mengubah password pengguna. Menjalankan perintah ini sebagai pengguna tanpa memberikan parameter akan mengubah password untuk pengguna tersebut. Perintah **passwd** akan menanyakan password lama sekali, kemudian meminta password baru sebanyak dua kali.

```
$ passwd
Old password:
Enter the new password (minimum of 5, maximum of 127 characters)
Please use a combination of upper and lower case letters and numbers.
New password:
Re-enter new password:
Password changed.
```

pengguna *root* dapat mengubah password pengguna lainnya dengan menspesifikasikan nama pengguna yang ingin diubah passwordnya sebagai parameter. Dalam hal ini perintah **passwd** hanya akan meminta password yang baru. Contoh :

```
# passwd bob
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

adduser

Perintah **adduser** mengkombinasikan **useradd** dan **passwd** dalam sebuah skrip yang interaktif. Skrip ini akan meminta Anda untuk mengisi informasi tentang account yana akan dibuat. Setelah itu skrip ini akan membuat account berdasarkan informasi yang telah Anda berikan. Listing di bawah ini akan menunjukkan contohnya.

```
# adduser

Login name for new user []: john

User ID ('UID') [ defaults to next available ]: <Enter>

Initial group [ users ]: <Enter>
```

```
Additional groups (comma separated) []: staff

Home directory [ /home/john ] <Enter>

Shell [ /bin/bash ] <Enter>

Expiry date (YYYY-MM-DD) []: <Enter>

New account will be created as follows:

---------------------------------------
Login name.......:  john
UID..............:  [ Next available ]
Initial group....:  users
Additional groups:  [ None ]
Home directory...:  /home/john
Shell............:  /bin/bash
Expiry date......:  [ Never ]

This is it... if you want to bail out, hit Control-C.  Otherwise, press
ENTER to go ahead and make the account.
<Enter>

Creating new account...


Changing the user information for john
Enter the new value, or press ENTER for the default
        Full Name []: John Doe
        Room Number []: <Enter>
        Work Phone []: <Enter>
        Home Phone []: <Enter>
        Other []: <Enter>
Changing password for john
Enter the new password (minimum of 5, maximum of 127 characters)
Please use a combination of upper and lower case letters and numbers.
New password: password
Re-enter new password: password


Account setup complete.
```

Jika Anda ingin menggunakan nilai standar yang disediakan, cukup dengan mengosongkan isian yang diminta, dengan menekan tombol <Enter>

userdel

Terkadang adalah hal yang perlu untuk menghapus acount pengguna dari sistem. GNU/Linux menyediakan perintah **userdel** untuk melakukan hal ini. Sebagai contoh perintah berikut akan menghapus account pengguna *bob* dari sistem.

```
# userdel bob
```

Perintah ini hanya akan menghapus account pengguna, namun tidak direktori home pengguna dan kumpulan email yang diterima. Cukup tambahkan parameter `-r` untuk menghapus direktori home pengguna serta kumpulan email yang diterima. Contoh :

```
# userdel -r bob
```

14.3. Menghindari penggunaan root dengan su

Merupakan ide bagus untuk menghindari login sebagai *root*. Ada banyak alasan untuk tidak melakukan hal ini. Mengetik perintah yang salah secara tidak sengaja dapat berakibat terjadi hal yang tidak diinginkan, dan program yang salah berpotensi untuk melakukan banyak kerusakan ketika kita login sebagai *root*. Akan tetapi, terdapat beberapa situasi yang membutuhkan akses root. Sebagai contoh, untuk melakukan administrasi sistem, atau menginstall program baru. Untungnya perintah **su** dapat memberikan Anda hak akses root untuk sementara.

Menggunakan perintah **su** sangat sederhana/ Cukup dengan mengeksekusi perintah **su**, kemudian Anda akan ditanyakan password root. Setelah password dimasukkan dengan benar, maka Anda akan memulai shell dengan hak akses root.

```
$ whoami
bob
$ su
Password:
# whoami
root
# exit
exit
$ whoami
bob
```

Pada contoh ini pengguna login sebagai *bob*. Keluaran perintah **whoami** merefleksikan hal ini. Pengguna mengeksekusi su dan memasukkan password *root*. Perintah **su** menjalankan sheel dengan hak akses root, hal ini dikonfirmasikan oleh keluaran perintah **whoami**. Setelah keluar dari shell *root*, kendali shell akan kembali ke pengguna sebelumnya yaitu *bob*.

Dalam hal ini memungkinkan juga untuk mengeksekusi satu perintah sebagai *root* dengan parameter `-c`. Berikut ini adalah contoh menjalankan perintah *lilo*:

```
$ su -c lilo
```

Jika Anda ingin memberikan parameter pada perintah yang ingin Anda jalankan, gunakan tanda kutip (contoh **su -c " ls -l /"**). Tanpa tanda petik **su** tidak dapat mengenali apakah parameter tersebut akan digunakan oleh perintah tertentu, atau oleh **su** itu sendiri

Membatasi hak akses su

Anda dapat menyaring akses ke **su** dengan *suauth*.Hal ini merupakan latihan keamanan yang bagus dengan hanya mengijinkan anggota dari grup tertentu untuk melakukan **su** ke *root*. Sebagai contoh, Anda dapat memembatasi perintah **su** ke root dengan gaya BSD yang hanya membolehkan anggota dari grup *wheel* dengan menambahkan baris berikut ke `/etc/suauth`:

```
root:ALL EXCEPT GROUP wheel:DENY
```

14.4. Kuota disk

Perkenalan

Kuota disk adalah mekanisme yang mengijinkan administrator sistem untuk membatasi jumlah blok disk dan node yang boleh digunakan oleh pengguna dan grup tertentu. Tidak semua sistem berkas yang didukung linux mendukung kuota. Secara luas sistem berkas yang mendukung kuota adalah ext2. ext3 dan XFS. Kuota diaktifkan dan diatur dalam tiap dasar sistem berkas.

Mengaktifkan kuota

Kuota dapat diaktifkan di tiap sistem berkas dalam `/etc/fstab`, dengan menambahkan opsi pada sistem berkas yaitu *`usrquota`* dan *`grpquota`*. Misalkan kita memiliki catatan berikut untuk partisi `/home` di `/etc/fstab`:

```
/dev/hda8        /home           xfs         defaults         1   2
```

Kini kita dapat mengaktifkan kuota pengguna dengan menambahkan *`usrquota`* pada opsi sistem berkas :

```
/dev/hda8        /home           xfs         defaults,usrquota 1   2
```

Pada titik ini, mesin dapat direboot untuk membiarkan Slackware Linux menginisialisasi skrip yang mengaktifkan kuota. Anda juga dapat mengaktifkan kuota tanpa harus mereboot mesin, dengan melakukan mount ulang partisi, dan menjalankan perintah **quotaon**:

```
# mount -o remount /home
# quotaon -avug
```

Mengedit kuota

Kuota pengguna dan grup dapat diedit menggunakan fasilitas “edquota”. Program ini memungkinkan Anda untuk mengedit kuota secara interaktif dengan editor **vi**. Sintaks yang paling dasar dari perintah ini adalah **edquota username**. Sebagai contoh:

```
# edquota joe
```

Perintah ini akan menjalankan editor **vi** dengan informasi kuota untuk pengguna *joe*. Hal ini akan tampak seperti di bawah ini:

```
Disk quotas for user joe (uid 1143):
  Filesystem                   blocks       soft       hard     inodes     soft    hard
```

```
  /dev/hda5                       2136          0          0         64        0        0
```

Dalam contoh ini kuota hanya diaktifkan untuk satu sistem berkas, yaitu pada sistem berkas di `/dev/hda5`. Sebagaimana yang Anda lihat disana terdapat banyak kolom. Kolom *blocks* menunjukkan banyak blok yang digunakan pengguna dalam sistem berkas, dan kolom *inodes* menunjukkan nomor inode yang ditinggali oleh pengguna. Disamping itu terdapat kolom *soft* dan *hard* setelah kolom *blocks* dan *inodes*. Kolom ini menentukan batas soft dan batas hard dalam blok dan inode. Pengguna dapat melewati batas soft pada periode perpanjangan yang ditentukan, tetapi pengguna tidak akan pernah melewati batas hard. Jika jumlah limit adalah *0*, maka tidak akan ada limit.

Catatan

Istilah “blok” mungkin agak membingungkan dalam konteks ini. Dalam konfigurasi kuota satu blok adalah 1KB, bukan jumlah ukuran satu blok dalam sistem berkas.

Mari kita lihat contoh sederhana. Misalkan kita ingin menentukan batas soft untuk pengguna *joe* menjadi *250000*, dan limit hard ke *300000*. Kita dapat mengubah contoh daftar kuota contoh diatas menjadi:

```
Disk quotas for user joe (uid 1143):
  Filesystem                   blocks       soft       hard     inodes     soft     hard
  /dev/hda5                      2136     250000     300000         64        0        0
```

Kuota yang baru untuk pengguna ini akan aktif setelah berkas disimpan, dan keluar dari **vi**.

Mendapatkan informasi tentang kuota

Seringkali merupakan hal yang berguna mendapatkan statistik terbaru tentang penggunaan kuota. Perintah **repquota** dapat digunakan untuk mendapatkan informasi tentang kuota apa saja yang ditentukan untuk tiap pengguna, dan berapa banyak kuota yang telah digunakan. Anda dapat melihat konfigurasi kuota untuk partisi tertentu dengan memberikan nama partisi sebagai parameter. Parameter *-a* akan menunjukkan informasi untuk semua partisi yang koutanya diaktifkan. Misalkan Anda ingin melihat informasi kuota untuk `/dev/hda5`, anda dapat menggunakan perintah berikut:

```
# repquota /dev/hda5
*** Report for user quotas on device /dev/hda5
Block grace time: 7days; Inode grace time: 7days
                        Block limits                File limits
User            used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root      --       0       0       0              3     0     0
[..]
joe       --    2136  250000  300000             64     0     0
[..]
```

Bab 15. Konfigurasi printer

15.1. Perkenalan

GNU/Linux mendukung sebagian besar jenis printer paralel dan USB, bahkan printer jaringan. Linux Slackware menyediakan dua jenis sistem pencetakan,CUPS (Common UNIX Printing System) dan LPRNG (LPR Next Generation). Bab ini membahas tentang sistem yang menggunakan CUPS.

Terlepas dari sistem pencetakan apa yang akan Anda gunakan, adalah hal bagus untuk menginstall beberapa koleksi filter printer. Koleksi ini dapat Anda temukan di kumpulan aplikasi “ap”. Jika Anda ingin memiliki dukungan untuk sebagian besar printer, pastikan paket berikut ini terinstall:

```
a2ps
enscript
espgs
gimp-print
gnu-gs-fonts
hpijs
ifhp
```

Kedua sistem pencetakan ini memiliki kelebihan dan kekurangan. Jika Anda tidak memiliki banyak pengalaman dengan melakukan konfigurasi printer di GNU/Linux, adalah ide bagus untuk menggunakan CUPS, karena CUPS menyediakan kenyamanan antarmuka web yang dapat diakses melalui web browser.

15.2. Persiapan

Untuk bisa menggunakan CUPS, maka paket “cups” dari kumpulan aplikasi “a” harus sudah diinstall. Setelah melakukan instalasi CUPS dapat dimulai secara otomatis setiap kali sistem di reboot dengan membuat `/etc/rc.d/rc.cups` dapat dieksekusi. Ini dapat diselesaikan dengan perintah berikut :

```
# chmod a+x /etc/rc.d/rc.cups
```

Setelah melakukan reboot pada sistem, CUPS juga akan dijalankan ulang secara otomatis. Anda dapat memulai CUPS dalam sistem yang sedang berjalan dengan mengeksekusi perintah berikut :

```
# /etc/rc.d/rc.cups start
```

15.3. Konfigurasi

CUPS dapat dikonfigurasikan lewat antarmuka web. Antarmuka konfigurasi ini dapat diakses melalui web browser dengan URL sebagai berikut : http://localhost:631/. Beberapa bagian dari antarmuka web ini membutuhkan autentifikasi Anda. Jika halaman autentifikasi mencul, Anda dapat memasukkan “root” sebagai nama pengguna, dan mengisi password pengguna root.

Untuk menambahkan printer ke dalam konfigurasi CUPS cukup dengan mengklik tombol “Administrate”, dan mengklik tombol “Add Printer” setelahnya. Antarmuka web akan menanyakan tiga jenis pertanyaan :

- *Name* - nama dari printer. Gunakan nama yang sederhana, sebagai contoh “epson”.
- *Location* - alamat fisik dari printer. Konfigurasi ini tidak krusial, tapi untuk organisasi yang besar informasi ini akan sangat berguna.
- *Description* - sebuah deskripsi dari printer, sebagai contoh “Epson Stylus Color C42UX”.

Anda dapat memproses dengan mengklik tombol “Continue”. Di halaman selanjutnya Anda dapat melakukan konfigurasi bagaimana printer tersebut terhubung. Jika Anda memiliki printer USB yang sedang tehubung, antarmuka web akan menunjukkan nama printer berikut dengan port USB yang digunakan. Setelah melakukan konfigurasi Anda dapat memilih merek dan model printer. Setelah konfigurasi printer selesai, printer akan ditambahkan ke dalam konfigurasi CUPS.

Ringkasan dari printer-printer yang telah dikonfigurasikan dapat ditemukan di halaman “Printers”. Di halaman ini Anda juga dapat melakukan beberapa operasi pada printer. Sebagai contoh, “Print Test Page” dapat digunakan untuk mengecek konfigurasi printer dengan mencetak halaman contoh.

15.4. Kendali akses

Sistem pencetakan dengan CUPS menyediakan konfigurasi dengan antarmuka web, dan melakukan pencetakan jarak jauh menggunakan Internet Printing Protocol (IPP). File konfigurasi CUPS memungkinkan Anda untuk melakukan konfigurasi dengan menyaring kendali akses ke printer. Secara default akses ke printer dibatasi hanya ke *localhost* (*127.0.0.1*).

Anda dapat membatasi kendali akses ke dalam pusat file konfigurasi daemon CUPS, `/etc/cups/cupsd.conf`, file ini memiliki sintaks yang dapat dibandingkan ke file konfigursi Apache. Kendali akses dikonfigurasikan dibagian *Location*. Sebagai contoh, bagian default global (direktori root IPP) terlihat seperti ini:

```
<Location />
Order Deny,Allow
Deny From All
Allow From 127.0.0.1
</Location>
```

Sebagaimana yang Anda lihat pernyataan deny diatur terlebih dulu, kemudian pernyataan allow. Dalam konfigurasi default akses ditolak dari semua host, kecuali untuk *127.0.0.1*, *localhost*. Andaikan Anda ingin mengijinkan host dari jaringan lokal, yang menggunakan ruang alamat *192.168.1.0/24*, untuk menggunakan printer dalam sistem yang Anda konfigurasikan menggunakan CUPS. Dalam kasus ini Anda dapat menambahkan baris yang dicetak tebal:

```
<Location />
Order Deny,Allow
Deny From All
Allow From 127.0.0.1
Allow From 192.168.1.0/24
</Location>
```

Anda dapat menyaring lokasi lainnya dalam ruang alamat diatas dengan menambahkan bagian lokasi tambahan. Konfigurasi untuk subdirektori tidak akan memperhatikan konfigurasi global. Sebagai contoh, Anda dapat membatasi akses ke printer *epson* dari host-host dengan alamat *127.0.0.1* dan *192.168.1.203* dengan menambah bagian berikut:

```
<Location /printers/epson>
Order Deny,Allow
Deny From All
Allow From 127.0.0.1
Allow From 192.168.1.203
</Location>
```

15.5. Ukuran kertas Ghostscript

Ghostscript adalah sebuah penterjemah jenis dokumen Portable Document Format (PDF) dan PostScript. PostScript dan PDF keduanya adalah bahasa yang mendeskripsikan data yang dapat di cetak. Ghostscript digunakan untuk mengubah PostScript dan PDF ke dalam bentuk format yang bisa ditampilkan di layar atau dicetak. Kebanyakan program UNIX menggunakan keluaran PostScript, Spooler CUPS menggunakan GhostScript untuk mengubah format PostScript ke dalam bentuk yang dapat dimengerti oleh printer tertentu.

Terdapat beberapa setting konfigurasi Ghostscript yang berguna untuk diubah dalam situasi tertentu. Bagian ini menjelaskan bagaimana Anda dapat mengubah ukuran kertas default yang digunakan Ghostscript. uses.

Catatan

Beberapa printer tingkat menengah keatas dapat menterjemahkan secara langsung format PostScript. Pengubahan pola Postscript tidak diperlukan untuk printer jenis ini.

Secara default Ghostscript menggunakan kertas US letter sebagai ukuran kertas standarnya. Ukuran kertas dikonfigurasikan dalam berkas `/usr/share/ghostscript/x.xx/lib/gs_init.ps`, nilai *x.xx* diganti dengan nomor versi Ghostscript. Tidak jauh dari baris pertama, terdapat dua baris yang di buat menjadi komentar dengan tanda persen (%), yang tampak seperti di bawah ini:

```
% Optionally choose a default paper size other than U.S. letter.
% (a4) /PAPERSIZE where { pop pop } { /PAPERSIZE exch def } ifelse
```

Anda dapat mengganti konfigursi Ghostscript untuk menjadikan A4 sebagai kertas standar dengan menghapus tanda persen dan spasi yang ada pada awal baris kedua, ubah menjadi:

```
% Optionally choose a default paper size other than U.S. letter.
(a4) /PAPERSIZE where { pop pop } { /PAPERSIZE exch def } ifelse
```

Dimungkinkan juga untuk menggunakan ukuran kertas lainnya selain Letter atau A4 dengan mengganti *a4* pada contoh diatas dengan ukuran kertas yang Anda inginkan untuk digunakan. Sebagai contoh, Anda dapat mengeset ukuran kertas standar ke US Legal dengan:

```
% Optionally choose a default paper size other than U.S. letter.
(legal) /PAPERSIZE where { pop pop } { /PAPERSIZE exch def } ifelse
```

Selain itu dimungkinkan juga untuk mengeset ukuran kertas sesuai permintaan kepada Ghostscript dengan menggunakan parameter *`-sPAPERSIZE=size`* pada perintah **gs**. Sebagai contoh, Anda dapat menambahkan parameter *-sPAPERSIZE=a4* ketika Anda memulai perintah **gs** untuk menggunakan ukuran kertas A4 sebagai permintaan ke Ghostscript.

Penjelasan ukuran kertas yang didukung dapat ditemukan dalam berkas `gs_statd.ps`, terdapat di direktori yang sama dengan berkas `gs_init.ps`.

Bab 16. X11

16.1. Konfigurasi X

Konfigurasi X11 disimpan di `/etc/X11/xorg.conf`. Banyak distribusi menyediakan aplikasi konfigurasi spesial untuk X, tapi Slackware Linux hanya menyediakan aplikasi standar (yang cukup gampang digunakan). Pada banyak kasus X dapat dikonfigurasikan secara otomatis, tetapi terkadang perlu untuk mengedit `/etc/X11/xorg.conf` secara manual.

Konfigurasi otomatis

Server X11 menyediakan pilihan untuk menghasilkan berkas konfigurasi secara otomatis. X11 akan memuat semua modul driver yang tersedia, dan akan mencoba mendeteksi perangkat keras kemudian menghasilkan sebuah berkas konfigurasi. Jalankan perintah berikut ini untuk menghasilkan berkas konfigurasi sebuah `xorg.conf` :

```
$ X -configure
```

Jika,X tidak menampilkan kesalahan apapun, berkas konfigurasi yang dihasilkan dapat di salin ke direktori `/etc/X11`. Dan X dapat dimulai untuk mengetes konfigurasi:

```
$ cp /root/xorg.conf /etc/X11/
$ startx
```

Konfigurasi interaktif

X11 menyediakan dua aplikasi untuk melakukan konfigurasi X secara interaktif, **xorgcfg** dan **xorgconfig**. **xorgcfg** berusaha untuk mendeteksi kartu grafis secara otomatis, kemudian memulai aplikasi yang dapat digunakan untuk menyetel konfigurasi. Terkadang **xorgcfg** memindahkan ke mode tampilan layar yang tidak didukung oleh monitor. Untuk kasus seperti ini, **xorgcfg** dapat juga digunakan dalam mode teks, dengan menggunakan perintah **xorgcfg -textmode**.

xorgconfig berbeda dengan aplikasi yang dijelaskan di atas, aplikasi ini tidak mendeteksi perangkat keras dan akan menanyakan pertanyaan secara mendetail mengenai perangkat keras Anda. Jika Anda hanya memiliki sedikit pengalaman melakukan konfigurasi X11 adalah ide bagus untuk menghindari **xorgconfig**.

16.2. Window manajer

“Tampilan dan sentuhan” dari X11 diatur oleh aplikasi yang disebut window manajer. Slackware Linux menyediakan beragam jenis window manajer yang dapat digunakan untuk berbagai keperluan, antara lain :

- WindowMaker: Sebuah window manajer yang relatif ringan yang merupakan bagian dari proyak GNUStep.
- BlackBox: Window manajer yang ringan, BlackBox tidak memiliki dependensi kecuali pustaka X11.
- KDE: Merupakan lingkungan desktop yang lengkap, mencakup browser, program e-mail dan paket aplikasi perkantoran (KOffice).
- Xfce: Sebuah lingkungan desktop yang ringan. Xfce merupakan lingkungan yang ideal jika Anda menginginkan desktop yang ramah pengguna dan dapat dijalankan pada mesin yang kemampuannya kurang tinggi.

Jika Anda terbiasa menggunakan lingkungan desktop, menggunakan KDE atau Xfce merupakan pilihan yang logis. Akan tetapi merupakan ide yang bagus untuk mencoba beberapa window manajer lainnya yang lebih ringan. Mereka lebih cepat, dan menkonsumsi sedikit memori, disamping itu, hampir semua aplikasi KDE dan Xfce dapat digunakan secara sempurna di window manajer lainnya.

Dalam Slackware Linux perintah **xwmconfig** dapat digunakan untuk mengatur window manajer mana yang akan menjadi standar. Program ini menunjukkan window manajer yang terinstall, dari sini Anda dapat memilih salah satu. Anda dapat mengatur window manajer secara global dengan mengeksekusi **xwmconfig** sebagai root.

Bab 17. Manajemen paket

17.1. Pkgtools

Perkenalan

Slackware Linux tidak menggunakan sistem paket yang kompleks, tidak seperti distribusi linux lainnya. Paket memiliki ekstensi `.tgz`, dan biasanya merupakan paket tarball biasa dengan dua berkas tambahan yaitu : sebuah skrip installasi dan berkas deskripsi paket tersebut. Sehubungan dengan kesederhanaan dari paketnya, aplikasi manajemen Slackware Linux tidak melakukan penanganan terhadap ketergantungan (dependensi). Tetapi banyak pengguna Slackware Linux memilih cara ini, karena penanganan dependensi sering mengakibatkan masalah dibandingkan pemecahannya.

Slackware Linux memiliki beberapa aplikasi yang digunakan untuk menangani paket. Pada bab ini akan dibahas aplikasi penanganan paket yang paling penting. Untuk belajar memahami aplikasi tersebut, penting untuk melihat penamaan paket terlebih dulu. Mari kita lihat sebuah contoh, bayangkan kita memiliki paket dengan nama berkas `bash-2.05b-i386-2.tgz`. Dalam kasus ini nama paket adalah bash-2.05b-i386-2. Dalam nama paket informasi tentang paket dipisahkan menggunakan karakter '-'. Sebuah nama paket memiliki arti sebagai berikut : *namaprogram-versi-arsitektur-versipaket*

pkgtool

Perintah **pkgtool** menyediakan antarmuka dalam bentuk menu untuk melakukan beberapa operasi terhadap paket. Bagian menu yang paling penting adalah *Remove* dan *Setup*. Opsi *Remove* menampilkan daftar paket yang telah terinstall. Anda dapat memilih paket mana yang Anda ingin hapus dengan menekan tombol spasi kemudian mengkonfirmasi pilihan Anda dengan menekan tombol Enter. Anda juga dapat membatalkan pilihan pada paket yang ingin Anda hapus dengan menekan kembali tombol spasi.

Opsi *Setup* menyediakan akses ke beberapa aplikasi yang dapat membantu melakukan konfigurasi terhadap sistem Anda, sebagai contoh: **netconfig**, **pppconfig** and **xwmconfig**.

installpkg

Perintah **installpkg** digunakan untuk menginstall paket. **installpkg** membutuhkan berkas paket sebagai parameter. Sebagai contoh, jika Anda ingin menginstall paket `bash-2.05b-i386-2.tgz` jalankan perintah sebagai berikut:

```
# installpkg bash-2.05b-i386-2.tgz
```

upgradepkg

upgradepkg dapat digunakan untuk mengupgrade paket. Berbeda dengan **installpkg**, aplikasi ini hanya menginstall paket jika dalam sistem terdapat versi yang lebih lama dari paket tersebut. Sintaks perintahnya dapat dibandingkan dengan **installpkg**. Sebagai contoh, jika Anda ingin mengupgrade paket menggunakan paket yang ada di dalam sebuah direktori, jalankan perintah berikut :

```
# upgradepkg *.tgz
```

Seperti yang telah dikatakan, paket-paket tersebut akan terinstall jika versi lain telah terinstall pada sistem tersebut.

removepkg

Perintah **removepkg** dapat digunakan untuk menghapus paket yang telah terinstall. Sebagai contoh, jika Anda ingin menghapus paket “bash” (Sangat tidak dianjurkan untuk melakukan hal ini !), Anda dapat mengeksekusi perintah:

```
# removepkg bash
```

Seperti yang Anda lihat, hanya nama program yang disebutkan dalam contoh ini. Anda juga dapat menghapus sebuah paket dengan menyebutkan nama lengkap dari paket tersebut :

```
# removepkg bash-2.05b-i386-2
```

17.2. Slackpkg

Perkenalan

Slackpkg adalah aplikasi pengolah paket yang ditulis oleh Roberto F. Batista dan Evaldo Gardenali. Aplikasi ini membantu pengguna untuk melakukan installasi dan upgrade paket Slackware Linux menggunakan salah satu dari mirror Slackware Linux. Slackpkg dimasukkan di direktori `extra/` dalam CD kedua dari kumpulan CD Slackware Linux.

Konfigurasi

Slackpkg dikonfigurasikan melalui sebuah berkas di `/etc/slackpkg`. Hal pertama yang harus Anda lakukan adalah mengkonfigurasikan mirror mana yang akan digunakan oleh slackpkg. Hal ini dapat diselesaikan dengan mengedit `/etc/slackpkg/mirrors`. Berkas ini telah berisi daftar mirror-mirror, Anda hanya perlu menghilangkan tanda komentar pada mirror yang dekat dengan Anda. Sebagai contoh:

```
ftp://ftp.nluug.nl/pub/os/Linux/distr/slackware/slackware-12.0/
```

Contoh ini akan menggunakan pohon Slackware Linux 12.0 dalam mirror ftp.nluug.nl. Pastikan untuk menggunakan pohon yang cocok dengan versi Slackware linux Anda. Jika Anda ingin menggunakan slackware-current Anda dapat menghilangkan tanda komentar pada baris berikut (jika Anda ingin menggunakan mirror NLUUG):

```
ftp://ftp.nluug.nl/pub/os/Linux/distr/slackware/slackware-current/
```

Slackpkg hanya akan menerima satu mirror. Menghilangkan tanda komentar lebih dari satu mirror akan membuat aplikasi ini tidak bekerja.

Mengimport kunci GPG Slackware Linux

Secara default slackpkg menguji paket menggunakan paket signature dan kunci GPG publik Slackware Linux. Karena hal ini merupakan ide yang bagus dari sudut pandang keamanan, Anda mungkin tidak ingin mengganti kebiasaan ini. Untuk mampu melakukan pengujian paket, Anda harus mengimport kunci GPG *security@slackware.com*. Jika Anda

tidak pernah menggunakan GPG sebelumnya, Anda harus membuat direktori GPG pada direktori home dari pengguna *root*:

```
# mkdir ~/.gnupg
```

Langkah selanjutnya adalah mencari kunci publik untuk *security@slackware.com*. Kita akan melakukan hal ini dengan memasukkan server *pgp.mit.edu* :

```
# gpg --keyserver pgp.mit.edu --search security@slackware.com
gpg: keyring `/root/.gnupg/secring.gpg' created
gpg: keyring `/root/.gnupg/pubring.gpg' created
gpg: searching for "security@slackware.com" from HKP server pgp.mit.edu
Keys 1-2 of 2 for "security@slackware.com"
(1)     Slackware Linux Project <security@slackware.com>
          1024 bit DSA key 40102233, created 2003-02-25
(2)     Slackware Linux Project <security@slackware.com>
          1024 bit DSA key 40102233, created 2003-02-25
Enter number(s), N)ext, or Q)uit >
```

Sebagaimana yang Anda lihat terdapat dua (identik) pilihan. Pilih yang pertama dengan memasukkan nilai “1”. GnuPG akan mengimport kunci ini ke dalam pengguna *root* :

```
Enter number(s), N)ext, or Q)uit > 1
gpg: key 40102233: duplicated user ID detected - merged
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: key 40102233: public key "Slackware Linux Project <security@slackware.com>" import
gpg: Total number processed: 1
gpg:               imported: 1
```

Pastikan untuk melakukan pengecekan ulang terhadap kunci yang Anda terima. ID kunci dan sidikjari dari kunci ini dapat ditemukan di internet pada berbagai situs yang dapat dipercaya. ID kunci adalah *40102233* seperti yang dijelaskan diatas. Anda bisa mendapatkan sidikjari dengan memasukkan parameter `--fingerprint` :

```
# gpg --fingerprint security@slackware.com
pub  1024D/40102233 2003-02-26 Slackware Linux Project <security@slackware.com>
     Key fingerprint = EC56 49DA 401E 22AB FA67  36EF 6A44 63C0 4010 2233
sub  1024g/4E523569 2003-02-26 [expires: 2012-12-21]
```

Sekali Anda mengimport dan mengecek kunci ini, Anda sudah bisa memulai menggunakan slackpkg, dan pastinya menginstall paket.

Mengupdate daftar paket

Sebelum mengupgrade dan menginstal paket Anda harus membiarkan slackpkg mendownload daftar paket dari mirror yang Anda gunakan. Adalah ide bagus untuk melakukan hal ini secara reguler untuk memastikan daftar tersebut selalu update. Daftar paket terakhir dapat diambil dengan :

```
$ slackpkg update
```

Mengupgrade paket

Parameter *upgrade* digunakan untuk mengupgrade paket yang telah terinstall. Anda harus menambahkan parameter tambahan untuk memberitahu **slackpkg** paket apa yang ingin Anda upgrade, hal ini berbeda untuk versi stabil Slackware Linux dan slackware current. Untuk upgrade keluaran stabil Slackware Linux terdapat di direktori `patches` dari mirror FTP. Anda dapat mengupdate instalasi versi stabil slackware (misalnya Slackware Linux 12.0) dengan :

```
# slackpkg upgrade patches
```

Pada kasus ini **slackpkg** akan menggunakan paket dari direktori `patches`. Dalam versi slackware current paket yang terbaru diletakkan dalam normal subdirektori paket yaitu `slackware`. Jadi, kita dapat melewatkannya sebagai parameter dalam perintah **slackpkg upgrade**:

```
# slackpkg upgrade slackware
```

Anda juga dapat mengupgrade paket secara individu dengan menspesifikasikan nama paket yang akan diupgrade, sebagai contoh:

```
# slackpkg upgrade pine
```

Menginstall paket

Parameter *install* digunakan untuk menginstall paket:

```
# slackpkg install rexima
```

Untuk diperhatikan, baik slackpkg ataupun aplikasi paket Slackware Linux tidak melakukan pengujian dependensi. Jika terdapat program yang tidak bekerja karena kehilangan pustaka tertentu, Anda dapat menambahkannya sendiri dengan slackpkg

17.3. Melakukan update menggunakan rsync

Metode populer lainnya untuk membuat Slackware Linux tetap update adalah dengan menyimpan ke mirror lokal. Langkah ideal untuk melakukan ini adalah lewat rsync. rsync adalah program yang dapat melakukan sinkronisasi dari dua pohon berkas. Keuntungannya adalah rsync hanya akan mentransfer perbedaan dalam sebuah berkas, yang membuat proses update menjadi sangat cepat. Setelah melakukan sinkronisasi dengan mirror Anda dapat mengupgrade Slackware Linux dengan **upgradepkg**, atau membuat CD instalasi yang baru. Contoh berikut ini melakukan sinkronisasi pohon direktori lokal dengan pohon yang terbaru dari mirror.

```
# rsync -av --delete \
```

```
--exclude=slackware/kde \
--exclude=slackware/kdei \
--exclude=slackware/gnome \
--exclude=bootdisks \
--exclude=extra \
--exclude=testing \
--exclude=pasture \
--exclude=rootdisks \
--exclude=source \
--exclude=zipslack \
rsync://fill-in-mirror/pub/slackware/slackware-current/ \
/usr/share/mirrors/slackware-current
```

Parameter *-a* seperti pada opsi lain mencoba membuat salinan yang sama persis (dalam hal mempertahankan symlink, hak akses, dan pemilik). Parameter *--delete* menghapus berkas yang tidak lagi tersedia pada mirror. Merupakan ide yang bagus untuk menggunakan parameter ini, karena pohon Anda mungkin dipenuhi dengan versi paket lama. Dengan parameter *--exclude* Anda bisa menentukan berkas atau direktori mana yang harus diabaikan.

Setelah melakukan sinkronisasi, Ana bisa menggunakan **upgradepkg** untuk mengupdate installasi Slackware Linux Anda. Sebagai contoh:

```
# upgradepkg /usr/share/mirrors/slackware-current/slackware/*/*.tgz
```

17.4. Tagfile

Perkenalan

Tagfile adalah fitur yang jarang diketahui dari Slackware Linux. Sebuah tagfile adalah berkas yang bisa digunakan untuk memberitahukan kepada **installpkg** paket apa yang harus diinstall dari sekumpulan paket-paket. Sebagai contoh, installer Slackware Linux menghasilkan tagfile selama mode installasi *Expert* dan *Menu* untuk menyimpan paket mana yang harus diinstall selama proses installasi.

Aspek yang bagus dari tagfile adalah Anda bisa membuat tagfile Anda sendiri. Dengan menulis tagfile Anda sendiri, Anda bisa mengotomatisasikan installasi paket, yang ideal untuk jumlah pengguna yang besar atau penggantian server (atau pengaturan yang lebih kecil jika Anda lebih nyaman dibandingkan menginstall paket secara manual). Cara termudah untuk membuat tagfile Anda adalah dengan menggunakan tagfile yang merupakan bagian dari distribusi resmi Slackware Linux. Pada bagian berikut, kita akan melihat bagaimana hal ini bisa dilakukan.

Membuat tagfile

Tagfile adalah berkas plain-text sederhana. Setiap baris terdiri dari nama paket dan sebuah flag, yang dipisahkan dengan tanda titik dua dan spasi. Flag menentukan apa yang harus dilakukan dengan sebuah paket. Kolom akan dijelaskan pada Tabel 17.1, “Kolom Tagfile”. Mari kita lihat pada beberapa baris pada tagfile dari set disk “a” :

```
aaa_base: ADD
aaa_elflibs: ADD
acpid: REC
apmd: REC
bash: ADD
```

```
bin: ADD
```

Harus dicatat bahwa Anda bisa menambahkan komentar pada tagfile dengan karakter komentar seperti biasa (#). Seperti yang Anda lihat pada potongan diatas, terdapat beberapa flag. Tabel dibawah menjelaskan empat flag yang berbeda.

Tabel 17.1. Kolom Tagfile

Flag	Arti
ADD	Sebuah paket yang ditandai dengan ini akan secara otomatis diinstall.
SKP	Sebuah paket yang ditandai dengan ini akan secara otomatis dilewati.
REC	Menanyakan kepada pengguna apa yang harus dilakukan, merekomendasikan installasi paket.
OPT	Menanyakan kepada pengguna apa yang harus dilakukan, paket akan dijelaskan sebagai opsional.

Seperti yang Anda baca dari tabel, **installpkg** hanya akan menginstall secara otomatis jika *ADD* atau *SKP* digunakan.

Akan menjadi pekerjaan membosankan untuk menulis tagfile untuk setiap set disk Slackware Linux. Distribusi resmi Slackware Linux berisi sebuah tagfile pada direktori untuk setiap set disknya. Anda bisa menggunakan tagfile ini sebagai awalnya. Script pendek berikut bisa digunakan untuk menyalin tagfile pada direktori aktual, sambil mempertahankan strukturnya.

```
#!/bin/sh

if [ ! $# -eq 1 ] ; then
  echo "Syntax: $0 [directory]"
  exit
fi

for tagfile in $1/*/tagfile; do
  setdir=`echo ${tagfile} | egrep -o '\w+/tagfile$' | xargs dirname`
  mkdir ${setdir}
  cp ${tagfile} ${setdir}/tagfile.org
  cp ${tagfile} ${setdir}
done
```

Setelah script di tulis ke dalam berkas, Anda dapat mengeksekusinya, kemudian menentukan direktori `slackware` yang menyediakan kumpulan disk. Sebagai contoh :

```
$ sh copy-tagfiles.sh /mnt/flux/slackware-current/slackware
```

Setelah melakukan hal ini, direktori aktual akan berisi struktur direktori seperti berikut, dimana Anda bisa mengedit berkas tagfile secara individu:

```
a/tagfile
a/tagfile.org
ap/tagfile
ap/tagfile.org
d/tagfile
d/tagfile.org
e/tagfile
e/tagfile.org
[...]
```

Berkas yang diakhiri dengan *.org* adalah cadangan, yang bisa Anda gunakan sebagai referensi ketika mengedit tagfile. Selain itu, mereka juga digunakan pada script yang akan dijelaskan pada bagian berikutnya.

Pembuatan tagfile secara otomatis

Dengan script yang sederhana, dimungkinkan untuk membangun tagfile berdasarkan paket yang diinstall pada sistem aktual. Saya berterima kasih kepada Peter Kaagman yang hadir dengan ide ini!

Pertama-tama, bangun direktori tagfile dari media installasi the Slackware Linux, seperti yang dijelaskan pada bagian sebelumnya. Setelah selesai, Anda bisa membuat script berikut:

```
#!/bin/sh

if [ ! $# -eq 1 ] ; then
  echo "Syntax: $0 [directory]"
  exit
fi

for tforg in $1/*/tagfile.org ; do
  tf=${tforg%.org}
  rm -f ${tf}
  for package in $(grep -v '^#' ${tforg} | cut -d ':' -f 1) ; do
    if ls /var/log/packages/${package}-[0-9]* &> /dev/null ; then
      echo "${package}: ADD" >> ${tf}
    else
      echo "${package}: SKP" >> ${tf}
    fi
  done
done
```

Misalkan Anda sudah menyimpannya sebagai `build-tagfiles.sh`, Anda bisa menggunakannya dengan menentukan direktori yang berisi tagfile sebagai argumen pertama:

```
$ sh build-tagfiles.sh .
```

Script akan menandai paket-paket yang terinstall sebagai *ADD*, dan paket-paket yang tidak terinstall sebagai *SKP*.

Menggunakan tagfile

Pada sistem yang terinstall, Anda bisa membiarkan **installpkg** menggunakan tagfile dengan parameter *-tagfile*. Sebagai contoh:

```
# installpkg -infobox -root /mnt-small -tagfile a/tagfile /mnt/flux/slackware-current/slackwa
```

Catatan

Anda harus menggunakan opsi *-infobox*, jika tidak, tagfile tidak akan digunakan.

Tentu saja, tagfile akan menjadi tidak berguna jika mereka tidak digunakan selama installasi dari Slackware Linux. Hal ini jelas mungkin: setelah memilih set disk mana yang hendak Anda install, Anda bisa memilih bagaimana Anda memilih paket (dialog bernama *SELECT PROMPTING MODE*. Pilih *tagpath* dari menu ini. Anda akan mendapatkan pertanyaan tentang path pada struktur direktori dengan tagfile. Cara umum untuk menyediakan tagfile pada installasi Slackware Linux adalah dengan meletakkannya pada disket atau media lain, dan me-mountnya sebelum atau saat installasi. Misalnya, Anda bisa berpindah pada konsol kedua dengan menekan kunci <ALT> and <F2>, dan membuat sebuah titik mount dan me-mount disk dengan tagfile:

```
# mkdir /mnt-tagfiles
# mount /dev/fd0 /mnt/mnt-tagfiles
```

Setelah me-mount disk, Anda bisa kembali ke konsol virtual dimana Anda menjalankan **setup**, dengan menekan <ALT> and <F1>.

Bab 18. Membangun kernel

18.1. Perkenalan

Kernel Linux secara singkat dibahas pada Bagian 2.1, “Apa itu Linux?”. Salah satu keunggulan dari Linux adalah ketersediaan dari kode sumbernya (seperti pada sistem Slackware Linux). Hal ini berarti Anda bisa mengkompilasi ulang kernel. Terdapat banyak situasi dimana mengkompilasi ulang kernel menjadi berguna. Sebagai contoh:

- **Membuat kernel lebih ringkas**: Anda dapat menonaktifkan beberapa fungsionalitas dari kernel, untuk mengurangi ukurannya. Hal ini menjadi penting pada lingkungan dimana memori sangat terbatas.
- **Mengoptimalkan kernel**: dimungkinkan untuk mengoptimalkan kernel. Sebagai contoh, dengan mengkompilasinya untuk jenis prosesor yang spesifik.
- **Dukungan perangkat keras**: Dukungan untuk beberapa perangkat keras tidak diaktifkan secara default pada kernel Linux yang disediakan oleh Slackware Linux. Contoh sederhana adalah dukungan untuk sistem SMP.
- **Menggunakan perbaikan khusus**: Terdapat banyak perbaikan tidak resmi untuk kernel Linux. Secara umum merupakan ide bagus untuk menghindari perbaikan tidak resmi. Tetapi beberapa perangkat lunak pihak ketiga, seperti Win4Lin [http://www.netraverse.com], memaksa Anda menginstall perbaikan kernel tambahan.
- Membuat header dan infrastruktur pembangunan yang benar untuk membangun modul pihak ketiga.

Bab ini berfokus pada seri kernel default yang digunakan pada Slackware Linux 12.0, Linux 2.6. Mengkompilasi ulang kernel tidaklah susah, asalkan Anda membuat salinan kernel yang bisa Anda gunakan ketika terjadi masalah. Kompilasi Linux melibatkan langkah-langkah berikut:

- Mengkonfigurasi kernel.
- Membangun kernel.
- Membangun modul-modul.
- Menginstall kernel dan modul-modul.
- Mengupdate konfigurasi LILO.

Pada bab ini, kami mengasumsikan bahwa kode sumber kernel tersedia pada `/usr/src/linux`. Jika Anda telah menginstall paket kernel-sources dari set disk “k”, kode sumber kernel tersedia pada `/usr/src/linux-versikernel`, dan `/usr/src/linux` adalah link simbolis pada direktori kode sumber kernel yang sebenarnya. Jadi, jika Anda menggunakan paket kernel standar dari Slackware Linux, Anda sudah siap.

Berbeda dengan versi kernel yang lama, tidak lagi diperlukan untuk menggunakan link simbolik `/usr/src/linux`. Jika Anda telah menguraikan kode sumber kernel y ang baru pada `/usr/src`, Anda bisa membangun kernel pada `/usr/src/linux-<version>`, dan menggunakan direktori tersebut pada contoh di bab ini.

18.2. Konfigurasi

Seperti yang dijelaskan diatas, langkah pertama adalah mengkonfigurasi kode sumber kernel. Untuk mempermudah konfigurasi kernel, merupakan ide yang bagus untuk menyalin konfigurasi kernel default Slackware Linux ke kode sumber kernel. Berkas konfigurasi kernel standar Slackware Linux disimpan pada media distribusi sebagai `kernels/<namakernel>/config`. Misalkan Anda hendak menggunakan konfigurasi kernel `hugesmp.s` sebagai awal (yang merupakan kernel default), dan Anda telah memiliki CD-ROM Slackware Linux yang sudah di-mount pada `/mnt/cdrom`, Anda bisa menyalin konfigurasi kernel Slackware Linux dengan :

```
# cp /mnt/cdrom/kernels/hugesmp.s/config /usr/src/linux/.config
```

Konfigurasi kernel yang berjalan juga bisa didapatkan pada `/proc/config.gz` jika kernel dikompilasi dengan opsi *CONFIG_IKCONFIG* dan *CONFIG_IKCONFIG_PROC*. Kernel default Slackware Linux sudah mengaktifkan opsi tersebut. Jadi, jika Anda hendak menggunakan konfigurasi dari kernel yang berjalan, Anda bisa menjalankan:

```
# zcat /proc/config.gz > /usr/src/linux/.config
```

Jika Anda menggunakan berkas konfigurasi untuk versi kernel lain yang hendak Anda kompilasi, pada biasanya kedua versi kernel tidak memiliki opsi yang sama. Opsi baru ditambahkan (misalnya driver baru ditambahkan), dan terkadang beberapa komponen kernel dihapus. Anda bisa mengkonfigurasikan opsi baru (dan menghapus opsi yang tidak dipakai) dengan perintah **make oldconfig** :

```
# cd /usr/src/linux ; make oldconfig
```

Hal ini akan menanyakan Anda tentang opsi-opsi, apakah Anda hendak mengkompilasinya secara penuh (*Y*), mengkompilasi sebagai modul (*M*), atau tidak menyertakannya (*N*). Sebagai contoh:

```
IBM ThinkPad Laptop Extras (ACPI_IBM) [N/m/y/?] (NEW)
```

Seperti yang Anda lihat, opsi yang mungkin ditampilkan, dengan pilihan default dengan huruf kapital. Jika Anda menekan <Enter>, maka opsi yang berhuruf besar akan digunakan. Jika Anda hendak melihat informasi tentang sebuah opsi, Anda bisa masukkan tanda tanya (*?*):

```
    IBM ThinkPad Laptop Extras (ACPI_IBM) [N/m/y/?] (NEW) ?

This is a Linux ACPI driver for the IBM ThinkPad laptops. It adds
support for Fn-Fx key combinations, Bluetooth control, video
output switching, ThinkLight control, UltraBay eject and more.
For more information about this driver see <file:Documentation/ibm-acpi.txt>
and <http://ibm-acpi.sf.net/> .

If you have an IBM ThinkPad laptop, say Y or M here.

    IBM ThinkPad Laptop Extras (ACPI_IBM) [N/m/y/?] (NEW)
```

Hasil keluaran dari perintah ini bisa cukup panjang, karena opsi yang digunakan pada berkas konfigurasi, dan yang tersedia pada kernel yang berjalan juga ditampilkan, tetapi konfigurasinya akan diisi sesuai dengan berkas konfigurasi.

Pada tahap ini, Anda bisa mulai mengkonfigurasi kernel secara detail. Terdapat tiga cara untuk mengkonfigurasi kernel. Pertama adalah *`config`*, yang hanya menanyakan kepada Anda apa yang hendak Anda lakukan untuk setiap opsi kernel. Hal ini memakan waktu sangat lama. Jadi biasanya ini bukan cara yang baik untuk mengkonfigurasi kernel. Cara yang lebih ramah adalah dengan *`menuconfig`*, yang menggunakan sistem berbasis menu yang bisa Anda gunakan untuk mengkonfigurasi kernel. Terdapat sistem berbasis X, yang bernama *`xconfig`*. Anda bisa menjalankannya dengan berpindah ke direktori kode sumber kernel, dan menjalankan **make <tampilan>**. Sebagai contoh, untuk mengkonfigurasi kernel dengan tampilan berbasis menu, Anda bisa menggunakan perintah berikut:

```
# cd /usr/src/linux
# make menuconfig
```

Tentu saja, jika Anda suka, Anda bisa mengedit berkas `.config` secara langsung dengan editor teks Anda.

Seperti yang kita lihat sebelumnya, pada konfigurasi kernel terdapat tiga opsi untuk setiap pilihan: “n” menonaktifkan fungsionalitas, “y” mengaktifkan fungsionalitas, dan “m” mengkompilasi fungsionalitas sebagai modul. Konfigurasi default kernel pada Slackware Linux adalah konfigurasi yang sangat bagus, ia menyertakan dukungan untuk semua pengontrol disk dan sistem berkas yang umum, dan sisanya dikompilasi sebagai modul. Apapun pilihan Anda, Anda harus memastikan bahwa driver untuk pengontrol disk dan dukungan untuk sistem berkas yang berisi sistem berkas root Anda harus disertakan. Jika tidak, kernel tidak akan bisa me-mount sistem berkas root, dan kernel akan panik karena tidak bisa menyerahkan inisialisasi pada program **init**.

Catatan

Selalu merupakan ide bagus untuk menyimpan kernel lama Anda, jika terjadi kesalahan pada saat konfigurasi. Jika kernel yang hendak dikompilasi memiliki nomor versi yang sama dengan kernel yang berjalan, Anda harus dengan serius mempertimbangkan untuk memodifikasi opsi *CONFIG_LOCALVERSION*. String yang ditentukan pada opsi ini ditambahkan pada nama versi. Sebagai contoh, jika kernel memiliki versi 2.6.21.6, dan *CONFIG_LOCALVERSION* diisi dengan *"-smp-ddk"*, versi kernel akan menjadi *2.6.21.6-smp-ddk*.

Jika Anda tidak memodifikasi versi dengan cara ini, installasi dari modul kernel baru akan menimpa modul dari kernel yang berjalan. Hal ini tentunya tidak baik jika Anda harus kembali ke kernel lama.

18.3. Kompilasi

Kompilasi kernel dahulu terdiri dari beberapa langkah, tetapi kernel Linux 2.6 bisa dikompilasi dengan menjalankan **make** pada direktori kode sumber kernel. Perintah ini akan menghitung ketergantungan, membangun kernel, dan membangun dan menghubungkan modul kernel.

```
# cd /usr/src/linux
# make
```

Setelah kompilasi selesai, pohon yang berisi modul yang sudah dikompilasi, dan citra kernel yang terkompresi bernama `bzImage` akan berada pada `/usr/src/linux/arch/i386/boot`. Anda bisa melanjutkan dengan installasi dari kernel dan modulnya.

18.4. Installasi

Menginstall kernel

Langkah berikutnya adalah untuk menginstall kernel dan modul kernel. Kita akan mulai dengan menginstall modul kernel, karena hal ini bisa dilakukan dengan satu perintah dari pohon kode sumber kernel:

```
# make modules_install
```

Hal ini akan menginstall modul pada `/lib/modules/<versikernel>`. Jika Anda mengganti kernel dengan nomor versi yang sama, maka merupakan ide bagus untuk menghapus modul lama sebelum menginstall yang baru. Misalnya:

```
# rm -rf /lib/modules/2.6.21.5-smp
```

Anda bisa “menginstall” kernel dengan menyalinnya pada direktori `/boot`. Anda bisa memberikan nama sesuai selera Anda, tetapi alangkah baiknya menggunakan konvensi nama. Misalnya `vmlinuz-versi`. Sebagai contoh, jika Anda menamainya `vmlinuz-2.6.21.5-smp-ddk`, Anda bisa menyalinnya dari pohon kode sumber kernel dengan:

```
# cp arch/i386/boot/bzImage /boot/vmlinuz-2.6.21.5-smp-ddk
```

Pada tahap ini, Anda sudah hampir selesai. Langkah terakhir adalah menambahkan kernel baru pada konfigurasi Linux boot loader (LILO).

Configuring LILO

LILO (Linux Loader) adalah boot loader default yang digunakan Slackware Linux. Konfigurasi dari LILO bekerja dalam dua langkah; langkah pertama adalah mengubah konfigurasi LILO pada `/etc/lilo.conf`. Langkah kedua adalah menjalankan **lilo**, yang akan menulis konfigurasi LILO pada boot loader. Konfigurasi LILO sudah memiliki informasi untuk kernel aktual yang Anda gunakan. Merupakan ide bagus untuk mempertahankan informasi ini, sebagai opsi kembali jika kernel baru Anda tidak bekerja. Jika Anda melihat akhir dari `/etc/lilo.conf` Anda akan melihat informasi yang serupa dengan ini:

```
# Linux bootable partition config begins
image = /boot/vmlinuz
  root = /dev/hda5
  label = Slack
  read-only # Non-UMSDOS filesystems should be mounted read-only for checking
# Linux bootable partition config ends
```

Cara termudah untuk menambahkan kernel baru adalah menduplikasi informasi yang ada, lalu mengedit informasi pertama, mengubah opsi *`image`*, dan *`label`*. Setelah mengubah, contoh diatas menjadi :

```
# Linux bootable partition config begins
image = /boot/vmlinuz-2.6.21.5-smp-ddk
  root = /dev/hda5
  label = Slack
  read-only # Non-UMSDOS filesystems should be mounted read-only for checking

image = /boot/vmlinuz
  root = /dev/hda5
  label = SlackOld
  read-only # Non-UMSDOS filesystems should be mounted read-only for checking
# Linux bootable partition config ends
```

Seperti yang Anda lihat, informasi *`image`* pertama mengarah pada kernel baru, dan label dari informasi kedua diganti ke “SlackOld”. LILO akan secara otomatis boot ke yang pertama. Anda sekarang bisa menginstall konfigurasi LILO baru dengan perintah **lilo** :

```
# lilo
Added Slack *
Added SlackOld
```

Pada saat Anda boot berikutnya, kedua informasi akan tersedia, dan bagian “Slack” akan dipakai sebagai default.

Catatan

Jika Anda hendak LILO menampilkan menu dengan informasi yang dikonfigurasikan pada `lilo.conf` pada setiap kali boot, pastikan Anda menambahkan baris yang berisi

```
prompt
```

pada `lilo.conf`. Jika tidak, LILO akan mem-boot informasi default yang ditentukan dengan *default=<name>*, atau informasi pertama ketika tidak ada kernel default yang ditentukan. Anda bisa mengakses menu kapan saja dengan menekan kunci <Shift> ketika LILO dijalankan.

Bab 19. Inisialisasi sistem

Bab ini menjelaskan inisialisasi Slackware Linux. Berbagai berkas konfigurasi yang digunakan untuk memanipulasi proses inisialisasi juga dijelaskan.

19.1. Bootloader

Bagian paling penting dari sebuah sistem operasi adalah kernelnya. Kernel mengelola sumber daya perangkat keras dan proses perangkat lunak. Kernel dijalankan oleh perekat antara sistem BIOS (Basic Input/Output System) dan kernel, yang disebut bootloader. Bootloader menangani komplikasi yang hadir bersamaan dengan memuat kernel tertentu.

Sebagian besar bootloader bekerja dalam dua tahap. Tahap pertama loader memuat loader tahap kedua, yang melakukan tugas sebenarnya. Boot loader dibagi menjadi dua tahap pada mesin x86, karena BIOS hanya mengacu pada satu sektor (maka disebut boot sector) yang berukuran 512 byte.

Slackware Linux menggunakan boot loader LILO (LInux LOader). Bootloader ini telah dikembangkan sejak 1992, dan secara spesifik ditulis untuk memuat kernel Linux. Belakangan LILO telah digantikan oleh GRUB (GRand Unified Bootloader) pada sebagian besar distribusi GNU/Linux. GRUB tersedia sebagai paket ekstra pada media distribusi Slackware Linux.

Konfigurasi LILO

LILO dikonfigurasikan melalui berkas konfigurasi `/etc/lilo.conf`. Slackware Linux menyediakan utilitas yang mudah untuk mengkonfigurasikan LILO. Utilitas konfigurasi ini bisa dijalankan dengan perintah **liloconfig**, seperti yang dijelaskan pada bab installasi (Bagian 5.3, “Menginstall Slackware Linux”).

Konfigurasi manual dari LILO sangatlah mudah. Berkas konfigurasi LILO biasanya diawali dengan pengaturan global:

```
# Start LILO global section
boot = /dev/sda ❶
#compact        # faster, but won't work on all systems.
prompt ❷
timeout = 50 ❸
# Normal VGA console
vga = normal ❹
```

❶ Opsi *boot* menentukan dimana bootloader LILO harus diinstall. Jika Anda hendak menggunakan LILO sebagai bootloader utama untuk menjalankan Linux dan atau sistem operasi lain, merupakan ide yang bagus untuk menginstall LILO pada MBR (Master Boot Record) dari hard disk yang Anda gunakan untuk mem-boot sistem. LILO diinstall pada MBR dengan mengabaikan nomor partisi, misalnya `/dev/hda` atau `/dev/sda`. Jika Anda hendak menginstall LILO pada partisi tertentu, tambahkan nomor partisi, seperti `/dev/sda1`. Pastikan Anda sudah memiliki bootloader lain pada MBR, atau partisi tadi sudah dibuat aktif menggunakan **fdisk**. Jika tidak, Anda mungkin berakhir dengan sistem yang tidak bisa boot.

Hati-hati jika Anda menggunakan partisi dengan sistem berkas XFS ! Menulis LILO pada partisi XFS akan menimpa sebagian dari sistem berkas. Jika Anda menggunakan XFS untuk sistem berkas root (`/`), buat sebuah sistem berkas `/boot` yang bukan XFS yang dipakai untuk menginstall LILO, atau tulis LILO pada MBR.

❷ Opsi *prompt* akan memaksa LILO menampilkan menu boot. Dari menu ini, Anda bisa memilih kernel atau sistem operasi mana yang akan di-boot. Jika Anda tidak menggunakan opsi ini, Anda masih bisa mengakses menu bootloader dengan menekan kunci <Shift> ketika bootloader dijalankan.

❸ Nilai *timeout* menentukan seberapa lama LILO harus menunggu sebelum kernel atau sistem operasi default akan di-boot. Waktu ditentukan dalam sepersepuluh detik, jadi pada contoh diatas, LILO akan menunggu 5 detik sebelum melanjutkan dengan boot.
❹ Anda bisa menentukan mode video mana yang harus digunakan kernel dengan opsi *vga*. Ketika opsi ini diisi dengan *normal* kernel akan menggunakan mode teks 80x25.

Opsi global diikuti dengan bagian yang menambahkan kernel Linux atau sistem operasi lain. Sebagian besar bagian untuk kernel Linux berisi seperti berikut:

```
image = /boot/vmlinuz ❶
  root = /dev/sda5 ❷
  label = Slack ❸
  read-only ❹
```

❶ Opsi *image* menentukan citra kernel yang harus dimuat untuk bagian LILO ini.
❷ Parameter *root* dikirimkan pada kernel, dan digunakan oleh kernel sebagai sistem berkas root (`/`).
❸ Teks *label* digunakan sebagai label untuk informasi ini pada menu boot LILO.
❹ *read-only* menentukan bahwa sistem berkas root harus dimount sebagai read-only. Sistem berkas harus dimount pada kondisi read-only untuk melakukan pengujian sistem berkas.

Installasi LILO

LILO tidak membaca berkas `/etc/lilo.conf` selama tahap kedua. Jadi Anda harus menulis perubahan pada loader tahap kedua ketika Anda telah mengubah konfigurasi LILO. Hal ini juga diperlukan jika Anda menginstall kernel baru dengan nama berkas yang sama, karena posisi dari kernel pada disk mungkin berubah. Menginstall ulang LILO bisa dilakukan dengan mudah dengan perintah **lilo**:

```
# lilo
Added Slack26 *
Added Slack
```

19.2. init

Setelah kernel dimuat dan dijalankan, kernel akan menjalankan perintah **init**. **init** adalah induk dari semua proses, dan mengurusi script inisialisasi sistem, membuat konsol login melalui **agetty**. Tingkah laku dari **init** dikonfigurasi melalui `/etc/inittab`.

Berkas `/etc/inittab` memiliki dokumentasi yang cukup baik. Berkas ini menentukan script mana yang harus dijalankan untuk setiap runlevel. Sebuah runlevel adalah kondisi pada saat sistem berjalan. Sebagai contoh, runlevel 1 adalah mode pengguna tunggal, dan runlevel 3 adalah mode banyak-pengguna. Kita akan melihat sebuah baris pada `/etc/inittab` untuk melihat cara kerjanya:

```
rc:2345:wait:/etc/rc.d/rc.M
```

Baris ini menunjukan bahwa **/etc/rc.d/rc.M** harus dijalankan ketika sistem berpindah ke runlevel 2, 3, 4 atau 5. Satu-satunya baris yang perlu Anda perhatikan adalah runlevel default:

```
id:3:initdefault:
```

Pada contoh ini, runlevel default diisi dengan *3* (mode banyak pengguna). Anda bisa mengisinya dengan runlevel yang lain dengan menggantgi 3 dengan nilai runlevel default yang baru. Runlevel 4 bisa cukup menarik pada mesin desktop, karena Slackware Linux akan mencoba menjalankan manajemen tampilan GDM, KDM atau XDM (dalam urutan). Manajemen tampilan ini menyediakan login grafis, dan merupakan bagian dari GNOME, KDE dan X11.

Bagian lain yang cukup menarik adalah baris yang menentukan perintah yang digunakan untuk mengurus sebuah konsol. Sebagai contoh:

```
c1:1235:respawn:/sbin/agetty 38400 tty1 linux
```

Baris ini menyatakan bahwa **agetty** akan dijalankan pada *tty1* (terminal virtual pertama) pada runlevel 1, 2, 3 dan 5. Perintah **agetty** akan membuka port tty, dan prompt untuk nama login. **agetty** akan menjalankan **login** untuk menangani proses login. Seperti yang Anda lihat dari informasi tersebut, Slackware Linux hanya menjalankan satu konsol pada runlevel 6, bernama *tty6*. Orang mungkin bertanya apa yang terjadi pada *tty0*, *tty0* masih ada, dan merepresentasikan konsol yang aktif.

Karena */etc/inittab* merupakan tempat yang cocok untuk menjalankan **agetty** untuk menangani login, Anda juga bisa mengijinkan lebih banyak agetty untuk mengurusi port serial. Hal ini menjadi berguna ketika Anda memiliki lebih dari satu terminal yang terkoneksi ke sebuah mesin. Anda bisa menambahkan informasi seperti berikut untuk menjalankan **agetty** yang mendengarkan pada COM1:

```
s1:12345:respawn:/sbin/agetty -L ttyS0 9600 vt100
```

19.3. Script inisialisasi

Seperti yang dijelaskan pada **init** (Bagian 19.2, “init”), **init** menjalankan beberapa script yang menangani runlevel yang berbeda-beda. Script ini melakukan pekerjaan dan mengubah pengaturan yang diperlukan untuk runlevel tertentu, tetapi mereka juga menjalankan script lain. Mari kita lihat contoh dari `/etc/rc.d/rc.M`, script yang dijalankan **init** ketika sistem berpindah ke runlevel banyak-pengguna:

```
# Start the sendmail daemon:
if [ -x /etc/rc.d/rc.sendmail ]; then
  . /etc/rc.d/rc.sendmail start
fi
```

Baris ini menyatakan “eksekusi **/etc/rc.d/rc.sendmail start** jika `/etc/rc.d/rc.sendmail` bersifat executable”. Hal ini mengindikasikan kesederhanaan dari script inisialisasi Slackware Linux. Berbagai fungsionalitas, seperti layanan jaringan, dapat diaktifkan atau dinonaktifkan, dengan mengganti tanda executable pada script inisialisasi. Jika script bersifat executable, layanan akan dijalankan, dan tidak akan dijalankan jika sebaliknya. Pengaturan flag dijelaskan pada bagian bernama “Mengganti bit hak akses berkas”, tetapi kita akan melihat pada contoh sederhana bagaimana Anda bisa mengaktifkan dan menonaktifkan sendmail.

Untuk menjalankan sendmail ketika sistem melakukan inisialisasi, jalankan:

```
# chmod +x /etc/rc.d/rc.sendmail
```

Untuk menonaktifkan sendmail ketika sistem melakukan inisialisasi, jalankan:

```
# chmod -x /etc/rc.d/rc.sendmail
```

Sebagian besar script inisialisasi layanan menerima tiga parameter untuk mengubah status dari layanan: *start*, *restart* dan *stop*. Parameter-parameter ini cukup deskriptif. Sebagai contoh, jika Anda hendak menjalankan ulang sendmail, Anda bisa menjalankan:

```
# /etc/rc.d/rc.sendmail restart
```

Jika script tidak bersifat executable, Anda harus memberitahukan kepada shell bahwa Anda hendak menjalankan berkas dengan **sh**. Sebagai contoh:

```
# sh /etc/rc.d/rc.sendmail start
```

19.4. Hotplugging dan manajemen node perangkat

Slackware Linux sudah mendukung hotplugging sejak Slackware Linux 9.1. Ketika diaktifkan, kernel mengirimkan notifikasi tentang kejadian tentang perangkat pada perintah userspace. Sejak Slackware Linux 11.0, sekumpulan utilitas **udev** menangani notifikasi ini. **udev** juga menangani direktori dinamis `/dev`.

Mode operasi dari **udev** untuk menangani proses hotplugging dari perangkat cukup sederhana. Ketika sebuah perangkat ditambahkan pada sistem, kernel memberitahukan userspace hotplug event listeners. **udev** akan menerima notifikasi bahwa sebuah perangkat telah ditambahkan, dan mencari apakah terdapat modul yang dipetakan untuk perangkat tersebut. Jika ada, maka modul driver yang sesuai akan dimuat secara otomatis. **udev** akan menghapus modul ketika menerima notifikasi bahwa perangkat telah dilepas, dan tidak ada perangkat yang menggunakan modul tersebut.

Sub sistem udev diinisialisasi pada `/etc/rc.d/rc.S` dengan menjalankan **/etc/rc.d/rc.udev start**. Seperti fungsionalitas yang lain, Anda bisa mengaktifkan atau menonaktifkan udev dengan mengubah flag executable dari script `/etc/rc.d/rc.udev` (lihat Bagian 19.3, “Script inisialisasi”).

Jika udev secara otomatis memuat modul yang tidak Anda perlukan, Anda bisa menambahkan *blacklist* pada konfigurasi **modprobe** Anda pada `/etc/modprobe.d/blacklist`. Sebagai contoh, jika Anda mencegah sistem memuat modul *8139cp*, Anda bisa menambahkan baris ini (sebetulnya, modul ini sudah di-blacklist pada Slackware Linux):

```
blacklist 8139cp
```

19.5. Firmware perangkat keras

Perkenalan

Beberapa perangkat keras membutuhkan sistem untuk mengupload firmware. Firmware adalah bagian dari perangkat lunak yang digunakan untuk mengontrol perangkat keras. Biasanya, firmware disimpan secara permanen pada ROM

(read-only memory) atau media non-volatile seperti flash memory. Namun, banyak perangkat keras baru menggunakan memori volatile untuk menyimpan firmware, yang berarti firmware harus dimuat ulang pada memori perangkat keras ketika sistem dijalankan ulang.

Driver untuk perangkat keras yang membutuhkan firmware memiiliki tabel dari berkas firmware yang dibutuhkan. Untuk setiap berkas firmware yang dibutuhkan driver, ia akan membuat sebuah event penambahan firmware. Jika udev menangani hotplugging events, maka ia akan mencoba menangani hal tersebut. Aturan udev yang berisi informasi tersebut berada pada `/etc/udev/rules.d/50-udev.rules`:

```
# firmware loader
SUBSYSTEM=="firmware", ACTION=="add", RUN+="/lib/udev/firmware.sh"
```

Hal ini berarti pada saat terjadi sebuah penambahan firmware, script `/lib/udev/firmware.sh` harus dijalankan. Script ini mencari direktori `/lib/firmware` dan `/usr/local/lib/firmware` untuk firmware yang diminta. Jika berkas firmware ditemukan, maka akan dimuat dengan menyalin isi berkas pada node sysfs spesial.

Menambahkan firmware

Seperti yang dijelaskan pada bagian sebelumnya, beberapa perangkat keras membutuhkan firmware untuk diupload ke perangkat keras oleh sistem operasi. Jika pada kasus ini tidak ada firmware yang diinstall, maka kernel akan mengeluarkan pesan kesalahan ketika driver untuk perangkat keras tersebut dimuat. Anda bisa melihat hasil keluaran kernel dengan perintah **dmesg** atau pada berkas log `/var/log/messages`. Pesan kesalahan tersebut secara eksplisit menyebutkan bahwa firmware tidak bisa dimuat, seperti berikut:

```
ipw2100: eth1: Firmware 'ipw2100-1.3.fw' not available or load failed.
ipw2100: eth1: ipw2100_get_firmware failed: -2
```

Pada kasus ini, Anda harus mencari firmware untuk perangkat keras Anda. Hal ini bisa ditemukan dengan mencari web untuk chipset atau driver untuk perangkat keras (pada kasus ini *ipw2100*) dan istilah “firmware”. Arsip firmware biasanya berisi berkas dengan instruksi installasi. Biasanya Anda cukup menyalin berkas firmware pada `/lib/firmware`.

Setelah menginstall firmware, Anda bisa memuat ulang driver dengan **rmmod** dan **modprobe**, atau dengan menjalankan ulang sistem.

Bab 20. Keamanan

20.1. Perkenalan

Dengan peningkatkan penggunaan dari Internet dan jaringan nirkabel, keamanan menjadi semakin penting setiap harinya. Tidaklah mungkin untuk membahas masalah ini dalam sebuah bab pada perkenalan kepada GNU/Linux. Bab ini membahas beberapa teknis keamanan dasar yang menyediakan langkah awal untuk keamanan desktop dan server.

Sebelum kita mulai pada subyek yang lebih spesifik, merupakan ide yang bagus untuk membuat beberapa catatan tentang kata sandi. Otorisasi komputer pada umumnya bergantung pada kata sandi. Pastikan memilih kata sandi yang bagus untuk segala situasi. Hindari menggunakan kata-kata, nama, tanggal lahir, dan kata sandi yang pendek. Kata sandi semacam ini bisa dengan mudah dibobol dengan serangan kamus atau brute force. Gunakan kata sandi yang panjang, idealnya delapan karakter atau lebih, terdiri dari huruf acak (termasuk huruf kapital) dan angka.

20.2. Menutup layanan

Perkenalan

Banyak GNU/Linux menjalankan beberapa layanan yang terbuka untuk jaringan lokal maupun Internet. Host lain bisa terkoneksi pada layanan ini dengan menghubungkan diri ke port tertentu. Sebagai contoh, port 80 digunakan untuk jalur WWW. Berkas `/etc/services` berisi tabel dengan semua layanan yang digunakan, dan nomor port yang digunakan untuk layanan tersebut.

Sistem yang aman seharusnya hanya menjalankan layanan yang diperlukan. Jadi, misalkan sebuah host berfungsi sebagai server web, maka seharusnya ia tidak memiliki port terbuka (dan melayani) FTP atau SMTP. Dengan banyak port yang terbuka, maka resiko keamanan meningkat dengan cepat, karena terdapat kemungkinan besar perangkat lunak yang memberikan layanan memiliki kecacatan, atau dikonfigurasi dengan tidak benar. Beberapa bagian berikut akan membantu Anda melacak port yang terbuka dan menutupnya.

Mencari port yang terbuka

Port yang terbuka dapat ditemukan menggunakan pemindai port. Salah satu pemindai port yang paling terkenal untuk GNU/Linux adalah **nmap**. **nmap** tersedia pada set disk “n”.

Sintaks dasar **nmap** adalah: **nmap host**. Parameter *host* bisa berupa nama host atau alamat IP. Misalkan kita hendak memindai host dimana **nmap** terpasang. Pada kasus ini, kita bisa menentukan alamat IP *localhost*, *127.0.0.1*:

```
$ nmap 127.0.0.1

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on localhost (127.0.0.1):
(The 1596 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
22/tcp     open        ssh
23/tcp     open        telnet
80/tcp     open        http
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 0 seconds
```

Pada kasus ini, Anda bisa melihat bahwa host memiliki lima port yang terbuka yang melayani; ftp, ssh, telnet, http dan X11.

inetd

Terdapat dua cara untuk menawarkan layanan TCP/IP: dengan menjalankan server aplikasi stand-alone (mandiri) sebagai daemon atau dengan menggunakan server internet super, **inetd**. inetd adalah sebuah daemon yang memonitor sekumpulan port. Jika sebuah klien mencoba untuk menghubungkan diri ke port, inetd akan menangani koneksi tersebut dan meneruskan koneksi pada perangkat lunak yang menangani jenis koneksi tersebut. Keunggulan dari pendekatan ini adalah adanya tambahan lapisan keamanan dan membuatnya lebih mudah untuk mencatat koneksi yang masuk. Kelemahannya adalah lebih lambat dibandingkan daemon yang bersifat stand-alone. Oleh sebab itu merupakan ide bagus untuk menjalankan daemon yang bersifat stand-alone pada server FTP yang sibuk.

Anda bisa menguji apakah **inetd** berjalan pada sebuah host atau tidak dengan **ps**, sebagai contoh:

```
$ ps ax | grep inetd
 2845 ?        S      0:00 /usr/sbin/inetd
```

Pada kasus ini, inetd berjalan dengan PID (process ID) 2845. **inetd** bisa dikonfigurasikan menggunakan berkas `/etc/inetd.conf`. Mari kita lihat contoh sebuah contoh dari inetd.conf:

```
# File Transfer Protocol (FTP) server:
ftp     stream  tcp     nowait  root    /usr/sbin/tcpd  proftpd
```

Baris ini menjelaskan bahwa inetd harus menerima koneksi FTP dan mengirimkannya pada **tcpd**. Hal ini mungkin cukup aneh, karena **proftpd** biasanya menangani koneksi FTP. Anda bisa menentukan untuk langsung menggunakan proftpd pada `inetd.conf`, tetapi merupakan ide bagus untuk memberikan koneksi pada **tcpd**. Program ini mengirimkan koneksi pada **proftpd**, seperti yang ditentukan. **tcpd** digunakan untuk memonitor layanan dan menyediakan kontrol akses berbasis host.

Layanan bisa dinonaktifkan dengan menambahkan karakter kres (#) di awal baris. Merupakan ide bagus untuk menonaktifkan semua layanan dan mengaktifkan layanan yang Anda butuhkan satu persatu. Setelah mengubah `/etc/inetd.conf` **inetd** harus dijalankan ulang untuk mengaktifkan perubahan. Hal ini bisa dilakukan dengan mengirimkan sinyal HUP pada proses inetd:

```
# ps ax | grep 'inetd'
   2845 ?        S      0:00 /usr/sbin/inetd
# kill -HUP 2845
```

Jika Anda tidak membutuhkan **inetd** sama sekali, merupakan ide bagus untuk menghapusnya. Jika Anda tetap ingin agar paket ini terinstall, tetapi tidak ingin agar Slackware Linux menjalankannya saat proses boot, jalankan perintah berikut sebagai root:

```
# chmod a-x /etc/rc.d/rc.inetd
```

Bab 21. Lain lain

21.1. Menjadwalkan tugas dengan cron

Murray Stokely

Slackware Linux menyertakan sebuah implementasi dari daemon cron UNIX klasik yang mengijinkan pengguna untuk menjadwalkan tugas untuk eksekusi pada interval yang rutin. Setiap pengguna bisa membuat, menghapus, atau memodifikasi berkas crontab individu. Berkas crontab ini menentukan perintah atau script yang harus dijalankan pada interval waktu tertentu. Baris kosong pada crontab atau baris yang diawali dengan hash (“#”) akan diabaikan.

Setiap informasi pada berkas crontab harus berisi 6 kolom yang dipisahkan dengan spasi. Kolom-kolom ini adalah menit, jam, tanggal, bulan, hari dalam minggu, dan perintah. Setiap bagian pada lima kolom pertama bisa berisi waktu atau wildcard “*” agar cocok dengan semua waktu untuk kolom tersebut. Sebagai contoh, untuk mendapatkan perintah **date** dijalankan setiap hari pada pukul 6:10 pagi, informasi berikut bisa digunakan.

```
10 6 * * * date
```

Crontab pengguna bisa dilihat dengan perintah **crontab -l**. Untuk perkenalan yang lebih dalam tentang sintaks berkas crontab, mari kita lihat dari crontab default *root*.

```
# crontab -l
# If you don't want the output of a cron job mailed to you, you have to direct
# any output to /dev/null.  We'll do this here since these jobs should run
# properly on a newly installed system, but if they don't the average newbie
# might get quite perplexed about getting strange mail every 5 minutes. :^)
#
# Run the hourly, daily, weekly, and monthly cron jobs.
# Jobs that need different timing may be entered into the crontab as before,
# but most really don't need greater granularity than this.  If the exact
# times of the hourly, daily, weekly, and monthly cron jobs do not suit your
# needs, feel free to adjust them.
#
# Run hourly cron jobs at 47 minutes after the hour:
47❶ *❷ *❸ *❹ *❺ /usr/bin/run-parts /etc/cron.hourly 1> /dev/null❻
#
# Run daily cron jobs at 4:40 every day:
40 4 * * * /usr/bin/run-parts /etc/cron.daily 1> /dev/null
#
# Run weekly cron jobs at 4:30 on the first day of the week:
30 4 * * 0 /usr/bin/run-parts /etc/cron.weekly 1> /dev/null
#
# Run monthly cron jobs at 4:20 on the first day of the month:
20 4 1 * * /usr/bin/run-parts /etc/cron.monthly 1> /dev/null
```

❶ Kolom pertama, 47, menentukan bahwa tugas ini harus terjadi pada menit ke-47 setelah informasi jam.

❷ Kolom kedua, *, adalah wildcard, sehingga tugas ini harus dilakukan *setiap* jam.

❸ Kolom ketiga, *, adalah wildcard, sehingga tugas ini harus dilakukan *setiap* hari.

❹ Kolom keempat, *, adalah wildcard, sehingga tugas ini harus dilakukan *setiap* bulan.

❺ Kolom kelima, *, adalah wildcard, sehingga tugas ini harus dilakukan *setiap* hari dalam minggu.

❻ Kolom keenam, **/usr/bin/run-parts /etc/cron.hourly 1> /dev/null**, menentukan perintah yang harus dijalankan pada spesifikasi waktu yang ditentukan pada lima kolom pertama.

Crontab default root diset untuk menjalankan script pada `/etc/cron.monthly` dalam interval bulanan, script pada `/etc/cron.weekly` dalam interval mingguan script pada `/etc/cron.daily` dalam interval harian, dan script pada `/etc/cron.hourly` dalam interval setiap jam. Untuk alasan inilah, tidaklah diperlukan bagi seorang administrator untuk memahami cara kerja cron secara mendalam. Dengan Slackware Linux, Anda bisa dengan menambahkan script baru pada salah satu direktori diatas untuk menjadwalkan tugas periodik yang baru. Memang, menggunakan direktori diatas akan memberikan ide yang bagus yang dilakukan oleh Slackware Linux dibelakang layar untuk menjaga agar beberapa hal, seperti basis data **slocate** tetap terupdate.

21.2. Parameter hard disk

Banyak disk modern menawarkan fitur yang berbeda-beda untuk meningkatkan performa dan integritas. Kebanyakan dari fitur ini bisa dimodifikasi dengan perintah **hdparm**. Hati-hati dengan mengubah pengaturan disk dengan utilitas ini, karena beberapa perubahan bisa merusak data pada disk Anda.

Anda bisa mendapatkan gambaran dari pengaturan yang aktif untuk sebuah disk dengan menyediakan node device dari disk sebagai parameter bagi **hdparm**:

```
# hdparm /dev/hda

/dev/hda:
 multcount    =  0 (off)
 IO_support   =  1 (32-bit)
 unmaskirq    =  1 (on)
 using_dma    =  1 (on)
 keepsettings =  0 (off)
 readonly     =  0 (off)
 readahead    = 256 (on)
 geometry     = 65535/16/63, sectors = 78165360, start = 0
```

Penyebab performa disk yang jelek pada umumnya adalah karena DMA tidak digunakan oleh kernel. DMA akan meningkatkan jumlah I/O dan penggunaan CPU yang offline, dengan cara mengirimkan data transfer langsung dari disk ke memori sistem. Jika DMA dinonaktifkan, gambaran pengaturan akan ditampilkan seperti baris berikut:

```
using_dma    =  0 (off)
```

Anda bisa dengan mudah mengaktifkan DMA untuk disk ini dengan parameter *-d* dari **hdparm**:

```
# hdparm -d 1 /dev/hda

/dev/hda:
 setting using_dma to 1 (on)
 using_dma    =  1 (on)
```

Anda bisa melakukannya setiap kali boot dengan menambahkan perintah hdparm pada `/etc/rc.d/rc.local`.

Parameter *-i* dari **hdparm** juga sangat berguna, karena memberikan informasi detail tentang sebuah disk:

```
# hdparm -i /dev/hda
```

```
/dev/hda:

 Model=WDC WD400EB-00CPF0, FwRev=06.04G06, SerialNo=WD-WCAAT6022342
 Config={ HardSect NotMFM HdSw>15uSec SpinMotCtl Fixed DTR>5Mbs FmtGapReq }
 RawCHS=16383/16/63, TrkSize=57600, SectSize=600, ECCbytes=40
 BuffType=DualPortCache, BuffSize=2048kB, MaxMultSect=16, MultSect=off
 CurCHS=16383/16/63, CurSects=16514064, LBA=yes, LBAsects=78163247
 IORDY=on/off, tPIO={min:120,w/IORDY:120}, tDMA={min:120,rec:120}
 PIO modes:  pio0 pio1 pio2 pio3 pio4
 DMA modes:  mdma0 mdma1 mdma2
 UDMA modes: udma0 udma1 udma2 udma3 udma4 *udma5
 AdvancedPM=no WriteCache=enabled
 Drive conforms to: device does not report version:

 * signifies the current active mode
```

21.3. Memonitor penggunaan memori

Pada beberapa situasi, sangatlah berguna untuk mendiagnosa informasi tentang bagaimana memori digunakan. Sebagai contoh, pada server yang memiliki performa jelek, Anda mungkin hendak melihat apakah kekurangan RAM menyebabkan sistem untuk melakukan proses swap, atau mungkin Anda melakukan pengaturan layanan jaringan, dan hendak mencari parameter cache yang optimal. Slackware Linux menyediakan beberapa perangkat yang bisa membantu Anda menganalisa bagaimana memori digunakan.

vmstat

vmstat adalah perintah yang bisa menyediakan statistik tentang banyak bagian dari sistem memori virtual. Tanpa parameter tambahan, **vmstat** menampilkan ringkasan dari beberapa statistik yang relevan:

```
# vmstat
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
 0  0      0 286804   7912  98632    0    0   198     9 1189   783  5  1 93  1
```

Karena kita hanya akan melihat pada penggunaan memori pada bagian ini, kita hanya akan melihat pada kolom *memory* dan *swap*.

- *swpd*: Jumlah memori virtual yang digunakan.
- *free*: Jumlah memori yang tidak digunakan saat ini..
- *buff*: Jumlah memori yang digunakan sebagai buffer.
- *cache*: Jumlah memori yang digunakan sebagai cache.
- *si*: Jumlah memori yang di-swap dari disk per detik.
- *so*: Jumlah memori yang di-swap ke disk per detik.

Seringkali berguna untuk melihat bagaimana penggunaan memori berubah seiring dengan waktu. Anda bisa menambahkan sebuah interval sebagai parameter pada **vmstat**, untuk menjalankan **vmstat** secara terus menerus,

mencetak statistik aktual. Interval ini dalam detik. Jadi, jika Anda ingin mendapatkan statistik yang baru setiap detik, Anda bisa menjalankan :

```
# vmstat 1
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 315132   8832  99324    0    0   189    10 1185   767  5  1 93  1
 1  0      0 304812   8832  99324    0    0     0     0 1222  6881 24  8 68  0
 0  0      0 299948   8836  99312    0    0     0     0 1171  1824 41  9 49  0
 [...]
```

Tambahan lain, Anda bisa memberitahukan **vmstat** berapa banyak ia harus mengeluarkan statistik ini (daripada melakukannya terus menerus). Sebagai contoh, jika Anda hendak mencetak statistik ini setiap dua detik, dan lima kali totalnya, Anda bisa menjalankan **vmstat** dengan cara berikut:

```
# vmstat 2 5
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 300996   9172  99360    0    0   186    10 1184   756  5  1 93  1
 0  1      0 299012   9848  99368    0    0   336     0 1293  8167 20  8 21 51
 1  0      0 294788  11976  99368    0    0  1054     0 1341 12749 14 11  0 76
 2  0      0 289996  13916  99368    0    0   960   176 1320 17636 22 14  0 64
 2  0      0 284620  16112  99368    0    0  1086   426 1351 21217 25 18  0 58
```

Bagian VI. Administrasi jaringan

Daftar Isi

Bab 22. Konfigurasi jaringan

22.1. Perangkat Keras

Kartu jaringan (NIC)

Driver untuk NIC sudah terinstall sebagai modul kernel. Modul untuk NIC Anda harus sudah dimuat saat inisialisasi Slackware Linux. Pada sebagian besar sistem, NIC sudah otomatis terdeteksi dan terkonfigurasi saat installasi Slackware Linux. Anda bisa mengkonfigurasi ulang NIC Anda dengan perintah **netconfig**. **netconfig** menambahkan driver (modul) untuk kartu yang terdeteksi pada `/etc/rc.d/rc.netdevice`.

Juga dimungkinkan untuk mengkonfigurasi secara manual modul-modul mana yang hendak dimuat selama inisialisasi sistem. Hal ini bisa dilakukan dengan menambahkan baris **modprobe** pada `/etc/rc.d/rc.modules`. Sebagai contoh, jika Anda hendak memuat modul untuk NIC 3Com 59x NIC (3c59x.o), tambahkan baris berikut pada `/etc/rc.d/rc.modules`

```
/sbin/modprobe 3c59x
```

Kartu PCMCIA

Kartu PCMCIA yang didukung akan terdeteksi secara otomatis oleh perangkat lunak PCMCIA. Paket pcmcia-cs dari set disk “a” menyediakan fungsionalitas PCMCIA untuk Slackware Linux.

22.2. Konfigurasi antarmuka

Kartu jaringan tersedia pada Linux melalui “antarmuka”. Perintah **ifconfig** digunakan untuk menampilkan antarmuka yang tersedia:

```
# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:20:AF:F6:D4:AD
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1301 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1529 errors:0 dropped:0 overruns:0 carrier:0
          collisions:1 txqueuelen:100
          RX bytes:472116 (461.0 Kb)  TX bytes:280355 (273.7 Kb)
          Interrupt:10 Base address:0xdc00

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:77 errors:0 dropped:0 overruns:0 frame:0
          TX packets:77 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:8482 (8.2 Kb)  TX bytes:8482 (8.2 Kb)
```

Kartu jaringan mendapatkan nama ethn, dimana n adalah sebuah angka, dimulai dari 0. Pada contoh diatas, kartu jaringan pertama (eth0) sudah memiliki alamat IP. Tetapi antarmuka yang belum terkonfigurasi tidak

akan mendapatkan alamat IP, perintah **ifconfig** tidak akan menampilkan alamat IP untuk antarmuka yang belum dikonfigurasi. Antarmuka bisa dikonfigurasi pada berkas `/etc/rc.d/rc.inet1.conf`. Anda bisa membaca komentar yang ada, dan mengisi informasi yang diperlukan. Sebagai contoh:

```
# Config information for eth0:
IPADDR[0]="192.168.1.1"
NETMASK[0]="255.255.255.0"
USE_DHCP[0]=""
DHCP_HOSTNAME[0]=""
```

Pada contoh ini, alamat IP 192.168.1.1 dengan netmask 255.255.255.0 diberikan pada antarmuka ethernet pertama (eth0). Jika Anda menggunakan server DHCP, Anda bisa mengganti baris *USE_DHCP=""* menjadi *USE_DHP[n]="yes"* (ganti “n” dengan nomor antarmuka). Variabel lain, selain *DHCP_HOSTNAME* akan diabaikan ketika menggunakan DHCP. Sebagai contoh:

```
IPADDR[1]=""
NETMASK[1]=""
USE_DHCP[1]="yes"
DHCP_HOSTNAME[1]=""
```

Hal yang sama berlaku pada antarmuka yang lain. Anda bisa mengaktifkan pengaturan dengan me-reboot sistem atau menjalankan **/etc/rc.d/rc.inet1**. Juga dimungkinkan untuk mengkonfigurasi ulang satu antarmuka dengan **/etc/rc.d/rc.inet1 ethX_restart**, dimana *ethX* harus diganti dengan nama antarmuka yang hendak dikonfigurasi ulang.

22.3. Konfigurasi pada antarmuka (IPv6)

Perkenalan

IPv6 adalah protokol internet generasi berikutnya. Salah satu keuntungannya adalah ia memiliki ruang alamat yang jauh lebih besar. Pada IPv4 (protokol internet yang umum digunakan saat ini) alamat berjumlah 32-bit, ruang alamat ini hampir sepenuhnya terisi dan adanya kekurangan alamat IPv4. IPv6 menggunakan alamat 128-bit, yang menyediakan ruang alamat yang sangat besar (2^128 alamat). IPv6 menggunakan notasi alamat yang berbeda, pertama-tama menggunakan nomor heksadesimal dan bukan lagi menggunakan nomor desimal, dan alamat dinotasikan dalam pasangan 16-bit yang dipisahkan oleh tanda titik dua (“:”). Mari kita lihat pada sebuah contoh:

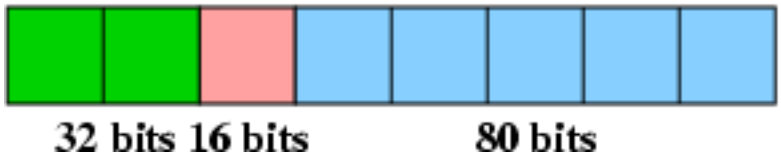
```
fec0:ffff:a300:2312:0:0:0:1
```

Blok yang bernilai nol bisa digantikan dengan dua titik dua (“::”). Sehingga, alamat diatas bisa ditulis sebagai:

```
fec0:ffff:a300:2312::1
```

Setiap alamat IPv6 memiliki prefix. Biasanya berisi dua elemen: 32 bit yang mengidentifikasi ruang alamat yang disediakan oleh penyedia, dan angka 16-bit yang menunjukan jaringan. Dua elemen ini membentuk sebuah prefix, dan pada kasus ini panjang prefix adalah 32 + 16 = 48 bit. Sehingga, jika Anda memiliki prefix /48, Anda bisa membuat 2^16 subnet dan 2^80 host pada setiap subnet. Gambar dibawah menunjukan struktur dari alamat IPv6 dengan prefix 48-bit.

Gambar 22.1. Anatomi alamat IPv6



Terdapat beberapa prefix yang sudah dipakai, misalnya:

Tabel 22.1. Prefix IPv6 Penting

Prefix	Deskripsi
fe80::	Link alamat lokal, yang tidak dirutekan.
fec0::	Alamat lokal situs, yang dirutekan secara lokal, tetapi tidak pada atau ke Internet.
2002::	alamat 6to4, yang digunakan untuk transisi dari IPv4 ke IPv6.

Dukungan IPv6 pada Slackware Linux

Kernel Linux yang disertakan pada Slackware Linux tidak mendukung IPv6 secara default, tetapi dukungan disertakan sebagai modul kernel. Modul ini bisa dimuat menggunakan **modprobe**:

```
# modprobe ipv6
```

Anda bisa memverifikasi jika dukungan IPv6 sudah dimuat dengan benar dengan melihat hasil keluaran kernel menggunakan **dmesg**:

```
$ dmesg
[..]
IPv6 v0.8 for NET4.0
```

Dukungan IPv6 bisa diaktifkan secara permanen dengan menambahkan baris berikut pada `/etc/rc.d/rc.modules`:

```
/sbin/modprobe ipv6
```

Antarmuka bisa dikonfigurasi menggunakan **ifconfig**. Tetapi direkomendasikan untuk menggunakan pengaturan IPv6 menggunakan perintah **ip**, yang merupakan bagian dari paket “iputils” yang dapat ditemukan pada direktori `extra/` dari pohon Slackware Linux.

Menambahkan alamat IPv6 pada antarmuka

Jika terdapat sembarang router pada jaringan, maka ada kemungkinan bahwa antarmuka pada jaringan tersebut sudah menerima sebuah alamat IPv6 ketika dukungan kernel untuk IPv6 sudah dimuat. Jika tidak, sebuah alamat IPv6

bisa ditambahkan pada sebuah antarmuka menggunakan utilitas **ip**. Misalkan kita hendak menambahkan alamat "fec0:0:0:bebe::1" dengan panjang prefix 64 (berarti "fec0:0:0:bebe" adalah prefix). Hal ini bisa dilakukan dengan sintaks berikut:

```
# ip -6 addr add <ip6addr>/<prefixlen> dev <device>
```

Sebagai contoh:

```
# ip -6 addr add fec0:0:0:bebe::1/64 dev eth0
```

22.4. Antarmuka nirkabel

Antarmuka nirkabel biasanya membutuhkan konfigurasi tambahan, seperti pengaturan ESSID, kunci WEP dan mode nirkabel. Pengaturan antarmuka yang khusus pada antarmuka nirkabel bisa ditentukan pada berkas `/etc/rc.d/rc.wireless.conf`. Script **/etc/rc.d/rc.wireless** mengkonfigurasikan antarmuka nirkabel berdasarkan deskripsi dari `/etc/rc.d/rc.wireless.conf`. Pada `rc.wireless.conf` pengaturan dibuat untuk setiap alamat MAC antarmuka. Secara default, berkas ini memiliki sebuah bagian yang cocok dengan setiap antarmuka:

```
## NOTE : Comment out the following five lines to activate the samples below ...
## --------- START SECTION TO REMOVE -----------
## Pick up any Access Point, should work on most 802.11 cards
*)
    INFO="Any ESSID"
    ESSID="any"
    ;;
## ---------- END SECTION TO REMOVE ------------
```

Merupakan ide bagus untuk menghapus bagian ini untuk membuat pengaturan per kartu. Jika Anda malas dan hanya memiliki satu kartu nirkabel, Anda bisa membiarkan bagian ini dan menambahkan parameter konfigurasi yang Anda perlukan. Karena bagian ini cocok dengan sembarang antarmuka nirkabel, maka kartu nirkabel Anda akan cocok dengan terkonfigurasi. Anda bisa menambahkan bagian untuk antarmuka kartu nirkabel Anda. Setiap bagian memiliki format sebagai berikut:

```
<MAC address>)
    <settings>
;;
```

Anda bisa menemukan alamat MAC dari sebuah antarmuka dengan melihat hasil keluaran **ifconfig** untuk sebuah antarmuka. Sebagai contoh, jika kartu nirkabel memiliki nama antarmuka *eth1*, Anda bisa mencari alamat MAC dengan cara berikut:

```
# ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 00:01:F4:EC:A5:32
          inet addr:192.168.2.2  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:1 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:504 (504.0 b)
```

```
          Interrupt:5 Base address:0x100
```

Alamat heksadesimal yang dicetak setelah *HWaddr* adalah alamat MAC, dalam kasus ini *00:01:F4:EC:A5:32*. Ketika Anda telah menemukan alamat MAC dari antarmuka, Anda bisa menambahkan bagian untuk perangkat keras ini pada `/etc/rc.d/rc.wireless.conf`. Sebagai contoh:

```
00:01:F4:EC:A5:32)
    INFO="Cabletron Roamabout WLAN NIC"
    ESSID="home"
    CHANNEL="8"
    MODE="Managed"
    KEY="1234-5678-AB"
    ;;
```

Hal ini mengatur antarmuka dengan alamat MAC *00:01:F4:EC:A5:32* untuk menggunakan ESSID *home*, bekerja pada mode *Managed* pada kanal *8*. Kunci yang digunakan untuk enkripsi WEP adalah *1234-5678-AB*. Terdapat banyak parameter lain yang bisa ditentukan. Untuk gambaran dari semua parameter, lihat pada contoh terakhir pada `rc.wireless.conf`.

Setelah mengkonfigurasi antarmuka nirkabel, Anda bisa mengaktifkan perubahan dengan menjalankan script inisialisasi jaringan **/etc/rc.d/rc.inet1**. Anda bisa melihat pengaturan nirkabel aktual dengan perintah **iwconfig** :

```
eth1      IEEE 802.11-DS  ESSID:"home"  Nickname:"HERMES I"
          Mode:Managed  Frequency:2.447 GHz  Access Point: 02:20:6B:75:0C:56
          Bit Rate:2 Mb/s   Tx-Power=15 dBm   Sensitivity:1/3
          Retry limit:4   RTS thr:off   Fragment thr:off
          Encryption key:1234-5678-AB
          Power Management:off
          Link Quality=0/92  Signal level=134/153  Noise level=134/153
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:27  Invalid misc:0   Missed beacon:0
```

22.5. Resolve

Nama host

Setiap komputer pada Internet memiliki sebuah nama host. Jika Anda tidak memiliki nama host yang dapat di-resolve oleh DNS, merupakan ide yang bagus untuk mengkonfigurasi nama host Anda, karena beberapa perangkat lunak menggunakannya. Anda bisa mengkonfigurasikan nama host pada `/etc/HOSTNAME`. Sebuah baris tunggal dengan nama host mesin sudah cukup. Biasanya sebuah nama host memiliki bentuk seperti berikut: host.domain.tld, sebagai contoh darkstar.slackfans.org. Hati-hati bahwa nama host harus dapat di-resolve, yang berarti GNU/Linux harus bisa mengkonversi nama host menjadi alamat IP. Anda bisa memastikan nama host dapat di-resolve dengan menambahkan pada `/etc/hosts`. Baca bagian berikutnya untuk informasi tentang berkas ini.

/etc/hosts

`/etc/hosts` adalah tabel alamat IP dengan nama host yang berhubungan. Berkas ini bisa digunakan untuk memberi nama komputer pada jaringan yang kecil. Mari kita lihat contoh dari berkas `/etc/hosts` :

```
127.0.0.1               localhost
192.168.1.1             tazzy.slackfans.org tazzy
192.168.1.169           flux.slackfans.org
```

Baris *localhost* harus selalu ada. Bari ini memberi nama *localhost* pada antarmuka khusus, loopback. Pada contoh ini, nama *tazzy.slackfans.org* dan *tazzy* diberi alamat IP 192.168.1.1, dan nama *flux.slackfans.org* diberi alamat IP 192.168.1.169. Pada sistem dimana berkas ini berada, kedua komputer akan tersedia sesuai dengan nama host yang diberikan.

Juga dimungkinkan untuk menambahkan alamat IPv6, yang akan digunakan jika sistem Anda terkonfigurasi untuk IPv6. Berikut adalah contoh berkas `/etc/hosts` untuk alamat IPv4 dan IPv6 :

```
# IPv4 entries
127.0.0.1               localhost
192.168.1.1             tazzy.slackfans.org tazzy
192.168.1.169           gideon.slackfans.org

# IPv6 entries
::1   localhost
fec0:0:0:bebe::2 flux.slackfans.org
```

Harap diperhatikan bahwa “::1” adalah loopback IPv6 default.

/etc/resolv.conf

Berkas `/etc/resolv.conf` digunakan untuk menentukan nameserver mana yang harus digunakan sistem. Sebuah nameserver mengkonversi nama host menjadi alamat IP. Penyedia Internet Anda seharusnya memberikan paling tidak dua alamat nameserver (server DNS). Anda bisa menambahkan nameserver ini pada `/etc/resolv.conf` dengan menambahkan baris *nameserver alamatIP* untuk setiap nameserver. Sebagai contoh:

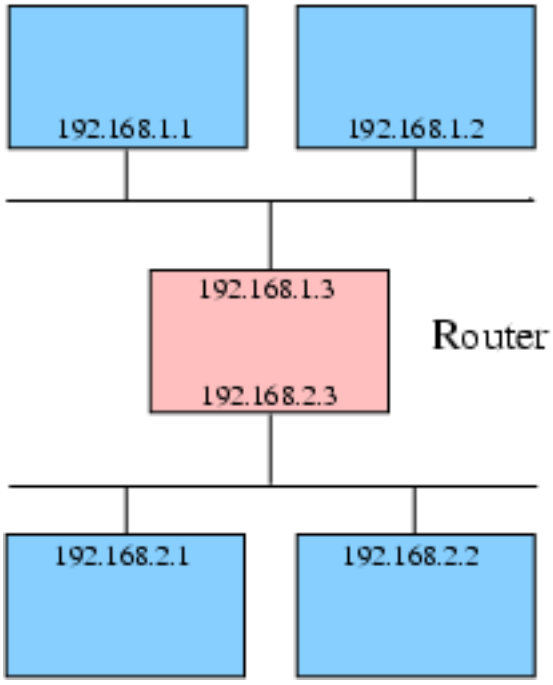
```
nameserver 192.168.1.1
nameserver 192.168.1.169
```

Anda bisa menguji apakah nama host ditranslasikan dengan benar atau tidak dengan perintah **host namahost**. Ganti *namahost* dengan nama hostname yang ada., sebagai contoh website dari penyedia layanan Internet Anda.

22.6. IPv4 Forwarding

IPv4 forwarding menghubungkan dua atau lebih jaringan dengan mengirimkan paket yang tiba pada satu antarmuka ke antarmuka lain. Hal ini memungkinkan sebuah mesin GNU/Linux berfungsi sebagai sebuah router. Sebagai contoh, Anda bisa menghubungkan banyak jaringan, atau jaringan rumah dengan Internet. Mari kita lihat pada sebuah contoh:

Gambar 22.2. Contoh Router



Pada contoh terdapat dua jaringan, 192.168.1.0 dan 192.168.2.0. Tiga host terhubung pada kedua jaringan. Salah satu dari host ini terhubung ke kedua jaringan dengan beberapa antarmuka. Antarmuka pada jaringan 192.168.1.0 memiliki alamat IP 192.168.1.3, antarmuka pada jaringan 192.168.2.0 memiliki alamat IP 192.168.2.3. Jika host bertindak sebagai router antar jaringan, maka ia mengirimkan paket dari jaringan 192.168.1.0 ke jaringan 192.168.2.0 dan sebaliknya. Routing dari paket TCP/IP IPv4 normal bisa diaktifkan dengan mengaktifkan IPv4 forwarding.

IPv4 forwarding bisa diaktifkan atau dinonaktifkan pada Slackware Linux dengan mengubah bit executable dari berkas `/etc/rc.d/rc.ip_forward`. Jika bit executable diaktifkan pada berkas ini, IP forwarding akan diaktifkan pada saat sistem boot, selain itu, tidak akan diaktifkan. Anda bisa menguji apakah bit executable sudah diaktifkan dengan **ls -l** (deskripsi dari perintah **ls** bisa ditemukan pada bagian bernama “Menampilkan berkas”).

Juga mungkin untuk mengaktifkan IPv4 forwarding pada sistem yang berjalan dengan perintah berikut (0 menonaktifkan forwarding, 1 mengaktifkan forwarding):

```
# echo 0 > /proc/sys/net/ipv4/ip_forward
```

Hati-hati! Secara default, tidak ada filter paket yang aktif. Hal ini berarti bahwa semua orang bisa mengakses jaringan lain. Beban bisa difilter dan dicatat dengan filter paket kernel iptables. Iptables bisa dikelola melalui perintah **iptables**. NAT (Network Address Translation) juga bagian dari iptables, dan bisa dikontrol dan diaktifkan melalui perintah **iptables**. NAT memungkinkan untuk “menyembunyikan” sebuah jaringan dibelakang sebuah alamat IP. Hal ini memungkinkan Anda menggunakan Internet pada jaringan yang lengkap hanya dengan satu alamat IP.

Bab 23. IPsec

23.1. Teori

IPsec adalah standar untuk mengamankan komunikasi IP melalui authentikasi, dan enkripsi. Selain itu, juga bisa mengkompresi paket, mengurangi beban. Protokol berikut merupakan bagian dari standar IPsec:

- AH (Authentication Header) menyediakan jaminan authentikasi untuk paket yang dikirimkan. Hal ini dilakukan dengan menghitung checksum dari paket menggunakan algoritma kriptografi. Jika checksum benar, maka penerima bisa yakin bahwa paket tidak dimodifikasi, dan paket benar-benar berasal dari pengirim (diasumsikan bahwa kunci hanya diketahui oleh pengirim dan penerima).

- ESP (Encapsulating Security Payload) digunakan untuk mengenkripsi paket. Hal ini membuat data pada paket aman, dan hanya dapat dibaca oleh host yang memiliki kunci dekripsi yang benar.

- IPcomp (IP payload compression) menyediakan kompresi sebelum paket dienkripsi. Hal ini berguna, karena data yang dienkripsi biasanya terkompresi lebih jelek daripada data yang tidak terenkripsi.

- IKE (Internet Key Exchange) menyediakan media untuk melakukan negosiasi kunci secara aman. Harap diperhatikan bahwa IKE bersifat opsinal, kunci bisa dikonfigurasikan secara manual.

Terdapat dua mode operasi: *mode transport* digunakan untuk mengenkripsi koneksi normal antar dua host, *mode tunnel* mengenkapsulasi paket asli pada header baru. Pada bab ini, kita akan melihat pada mode transport, karena tujuan utama dari bab ini adalah menunjukan bagaimana membuat pengaturan koneksi yang aman antara dua host.

Terdapat dua metode utama untuk otentikasi. Anda bisa menggunakan kunci manual, atau daemon Internet Key Exchange (IKE) seperti racoon, yang secara otomatis bertukar kunci diantara dua host secara aman. Pada kedua kasus, Anda harus menentukan aturan pada Security Policy Database (SPD). Basis data ini digunakan oleh kernel untuk menentukan jenis aturan keamanan apa yang diperlukan untuk berkomunikasi dengan host lain. Jika Anda menggunakan kunci manual, Anda harus mengatur Security Association Database (SAD), yang menentukan algoritma enkripsi dan kunci yang harus digunakan untuk komunikasi yang aman dengan host lain. Jika Anda menggunakan daemon IKE, asosiasi keamanan akan dibuat secara otomatis.

23.2. Konfigurasi Linux

Dukungan native IPsec hanya tersedia pada kernel Linux 2.6.x. Kernel versi sebelumnya tidak memiliki dukungan native terhadap IPsec. Jadi, pastikan Anda sudah memiliki kernel 2.6.x. Kernel 2.6 tersedia pada Slackware Linux 10.0, 10.1, dan 10.2 dari direktori `testing` pada CD2 dari CD Slackware Linux, atau sembarang mirror resmi Slackware Linux. Kernel 2.6 adalah kernel default semenjak Slackware Linux 12.0. Default kernel 2.6 Slackware Linux sudah mendukung AH, ESP dan IPcomp pada IPv4 dan IPv6. Jika Anda mengkompilasi kernel sendiri, aktifkan paling tidak opsi-opsi berikut pada konfigurasi kernel Anda:

```
CONFIG_INET_AH=y
CONFIG_INET_ESP=y
CONFIG_INET_IPCOMP=y
```

Atau, Anda bisa mengkompilasi dukungan untuk protokol IPsec sebagai modul:

```
CONFIG_INET_AH=m
```

```
CONFIG_INET_ESP=m
CONFIG_INET_IPCOMP=m
```

Pada bab ini, kita hanya akan menggunakan transformasi AH dan ESP, tetapi bukan ide yang jelek untuk mengaktifkan transformasi IPComp untuk konfigurasi lebih lanjut dari IPsec. Selain dukungan untuk protokol IPsec, Anda harus mengkompilasi dukungan kernel untuk algoritma enkripsi dan hash yang akan digunakan oleh AH atau ESP. Linux atau dukungan modul untuk algoritma ini bisa diaktifkan pada opsi *CONFIG_CRYPTO*. Tidaklah merepotkan untuk mengkompilasi semua algoritma cipher dan hash sebagai sebuah modul.

Ketika Anda memilih untuk mengkompilasi dukungan IPsec sebagai modul, pastikan bahwa modul yang diperlukan sudah dimuat. Sebagai contoh, jika Anda hendak menggunakan ESP untuk koneksi IPv4, muat modul *esp4*.

Kompilasi kernel seperti biasa dan boot kernel tersebut.

23.3. Installing IPsec-Tools

Langkah berikutnya adalah menginstall IPsec-Tools [http://ipsec-tools.sourceforge.net]. Aplikasi ini merupakan port dari utilitas IPsec KAME [http://www.kame.net]. Download kode terbaru dan uraikan, konfigurasi, dan install:

```
# tar jxf ipsec-tools-x.y.z.tar.bz2
# cd ipsec-tools-x.y.z
# CFLAGS="-O2 -march=i486 -mcpu=i686" \
  ./configure --prefix=/usr \
              --sysconfdir=/etc \
              --localstatedir=/var \
              --enable-hybrid \
              --enable-natt \
              --enable-dpd \
              --enable-frag \
              i486-slackware-linux
# make
# make install
```

Ganti *x.y.z* dengan kode versi yang didownload. Flag yang kita tentukan selama konfigurasi kode sumber adalah:

- *--enable-dpd*: mengaktifkan dead peer detection (DPD). DPD adalah metode untuk mendeteksi apakah sembarang host dimana asosiasi keamanan sudah ditentukan tidak dapat dapat dicapai. Jika benar, maka asosiasi keamanan untuk host tersebut bisa dihapus.
- *--enable-natt*: mengaktifkan NAT traversal (NAT-T). Karena NAT mengganti header IP, hal ini menimbulkan masalah untuk penjaminan authentikasi dari paket. NAT-T adalah metode yang membantu mengatasi masalah ini. Mengkonfigurasi NAT-T diluar batasan dari artikel ini.

23.4. Mengatur IPsec dengan kunci manual

Perkenalan

Kita akan menggunakan sebuah contoh sebagai panduan untuk mengatur koneksi tidak terenkripsi diantara dua host. Host akan memiliki alamat IP *192.168.1.1* dan *192.168.1.169*. “Mode transport ” akan digunakan dengan transformasi AH dan ESP beserta kunci manual.

Menulis berkas konfigurasi

Langkah pertama adalah menulis berkas konfigurasi yang akan kita beri nama `/etc/setkey.conf`. Pada host pertama (192.168.1.1) berkas konfigurasi `/etc/setkey.conf` berikut akan digunakan:

```
#!/usr/sbin/setkey -f

# Flush the SAD and SPD
flush;
spdflush;

add 192.168.1.1 192.168.1.169 ah 0x200 -A hmac-md5
0xa731649644c5dee92cbd9c2e7e188ee6;
add 192.168.1.169 192.168.1.1 ah 0x300 -A hmac-md5
0x27f6d123d7077b361662fc6e451f65d8;

add 192.168.1.1 192.168.1.169 esp 0x201 -E 3des-cbc
0x656c8523255ccc23a66c1917aa0cf30991fce83532a4b224;
add 192.168.1.169 192.168.1.1 esp 0x301 -E 3des-cbc
0xc966199f24d095f3990a320d749056401e82b26570320292

spdadd 192.168.1.1 192.168.1.169 any -P out ipsec
           esp/transport//require
           ah/transport//require;

spdadd 192.168.1.169 192.168.1.1 any -P in ipsec
           esp/transport//require
           ah/transport//require;
```

Baris pertama (baris yang diakhiri “;”) menambahkan kunci untuk pengujian checksum header untuk paket yang datang dari 192.168.1.1 dan menuju 192.168.1.169. Baris kedua menambahkan hal yang sama untuk paket datang dari 192.168.1.169 menuju 192.168.1.1. Baris keempat dan kelima mendefinisikan kunci untuk enkripsi data seperti yang dilakukan oelh dua baris pertama. Akhirnya, baris “spadd” mendefinisikan dua aturan, yaitu paket yang keluar dari 192.168.1.1 menuju 192.168.1.169 harus (require) di-enkodekan (esp) dan “signed” dengan header otorisasi. Aturan kedua adalah untuk paket yang masuk dan melakukan hal yang sama dengan paket keluar.

Harap Anda perhatikan bahwa Anda tidak seharusnya menggunakan kunci ini, tetapi kunci unik Anda sendiri. Anda bisa menghasilkan kunci menggunakan perangkat `/dev/random` :

```
# dd if=/dev/random count=16 bs=1 | xxd -ps
```

Perintah ini menggunakan dd untuk menghasilkan 16 byte dari `/dev/random`. Jangan lupa untuk menambahkan “0x” pada awal baris pada berkas konfigurasi. Anda bisa menggunakan 16 byte (128 bit) untuk algoritma hmac-md5 yang digunakan untuk AH. Algoritma 3des-cbc yang digunakan untuk ESP pada contoh harus menggunakan kunci 24 byte (192 bit). Kunci-kunci ini bisa dihasilkan dengan :

```
# dd if=/dev/random count=24 bs=1 | xxd -ps
```

Pastikan bahwa berkas `/etc/setkey.conf` hanya bisa dibaca oleh pengguna root. Jika pengguna biasa dapat membaca IPsec tidak menyediakan keamanan sama sekali. Anda bisa melakukannya dengan cara:

```
# chmod 600 /etc/setkey.conf
```

Berkas `/etc/setkey.conf` yang sama bisa dibuat pada host 192.168.1.169, dengan opsi *`-P in`* dan *`-P out`* dibalik. Jadi, berkas `/etc/setkey.conf` akan tampak seperti berikut :

```
#!/usr/sbin/setkey -f

# Flush the SAD and SPD
flush;
spdflush;

add 192.168.1.1 192.168.1.169 ah 0x200 -A hmac-md5
0xa731649644c5dee92cbd9c2e7e188ee6;
add 192.168.1.169 192.168.1.1 ah 0x300 -A hmac-md5
0x27f6d123d7077b361662fc6e451f65d8;

add 192.168.1.1 192.168.1.169 esp 0x201 -E 3des-cbc
0x656c8523255ccc23a66c1917aa0cf30991fce83532a4b224;
add 192.168.1.169 192.168.1.1 esp 0x301 -E 3des-cbc
0xc966199f24d095f3990a320d749056401e82b26570320292

spdadd 192.168.1.1 192.168.1.169 any -P in ipsec
           esp/transport//require
           ah/transport//require;

spdadd 192.168.1.169 192.168.1.1 any -P out ipsec
           esp/transport//require
           ah/transport//require;
```

Mengaktifkan konfigurasi IPsec

Konfigurasi IPsec bisa diaktifkan dengan perintah **setkey** :

```
# setkey -f /etc/setkey.conf
```

Jika Anda hendak mengaktifkan IPsec secara permanen, Anda bisa menambahkan baris berikut pada `/etc/rc.d/rc.local` pada kedua host :

```
/usr/sbin/setkey -f /etc/setkey.conf
```

Setelah mengkonfigurasi IPsec, Anda bisa menguji koneksi dengan menjalankan **tcpdump** dan melakukan ping ke host yang lain. Anda bisa melihat jika AH dan ESP digunakan pada hasil keluaran perintah **tcpdump** :

```
# tcpdump -i eth0
tcpdump: listening on eth0
11:29:58.869988 terrapin.taickim.net > 192.168.1.169: AH(spi=0x00000200,seq=0x40f): ESP
11:29:58.870786 192.168.1.169 > terrapin.taickim.net: AH(spi=0x00000300,seq=0x33d7): ES
```

23.5. Mengatur IPsec dengan pertukaran kunci otomatis

Perkenalan

Judul dari pertukaran kunci otomatis sudah dibahas sebentara pada perkenalan dari bab ini. Secara sederhana, IPsec dengan IKE bekerja seperti berikut.

1. Beberapa proses pada host hendak terhubung ke host lain. Kernel menguji apakah terdapat aturan keamanan yang sudah ditentukan untuk host lain. Jika sudah terdapat asosiasi keamanan yang berhubungan dengan aturan, koneksi bisa dibuat, dan akan diautentikasi, dienkripsi, dan atau dikompresi sesuai dengan definisi. Jika tidak ada asosiasi keamanan, kernel akan meminta daemon IKE untuk membuat asoasiasi keamanan yang diperlukan.

2. Pada tahap pertama dari pertukaran kunci, daemon IKE mencoba memverifikasi authentikasi dari host lain. Hal ini biasanya dilakukan dengan kunci yang sudah dipersiapkan (preshared) atau sertifikat. Jika authentikasi berhasil, maka kanal rahasia akan dibentuk diantara dua host, yang biasanya disebut asosiasi keamananan IKE, untuk melanjutkan dengan pertukaran kunci.

3. Selama fase kedua dari pertukaran kunci, asosiasi keamanan untuk komunikasi dengan host lain akan dibentuk. Hal ini melibatkan pemilihan algoritma enkripsi yang akan digunakan, dan menghasilkan kunci yang akan digunakan untuk mengenkripsi komunikasi.

4. Pada tahap ini, langkah pertama akan diulangi kembali, tetapi karena sudah ada asosiasi keamanan, komunikasi bisa dilanjutkan.

Daemon IKE **racoon** disertakan dengan aplikasi KAME IPsec, bagian berikut menjelaskan bagaimana mengatur racoon.

Menggunakan dengan kunci yang preshared

Seperti biasa, langkah pertama untuk mengatur IPsec adalah untuk mendefinisikan aturan keamanan. Kebalikan dari kunci manual, Anda tidak perlu membuat asosiasi keamanan, karena racoon akan membuatkan untuk Anda. Kita akan menggunakan IP host yang sama dengan contoh diatas. Aturan keamanan tampak seperti berikut:

```
#!/usr/sbin/setkey -f

# Flush the SAD and SPD
flush;
spdflush;

spdadd 192.168.1.1 192.168.1.169 any -P out ipsec
 esp/transport//require;

spdadd 192.168.1.169 192.168.1.1 any -P in ipsec
 esp/transport//require;
```

Orang yang cermat akan melihat bahwa aturan AH tidak ada pada contoh ini. Pada sebagian besar kasus, hal ini bukanlah masalah, ESP bisa menyediakan authentikasi. Tetapi Anda harus waspada bahwa authentikasi menjadi lebih lebar; ia tidak menjaga informasi diluar header ESP. Tetapi lebih efisien daripada mengenkapsulasi paket ESP pada AH.

Dengan aturan keamanan sudah terbentuk, Anda bisa mengkonfigurasi **racoon**. Karena informasi yang spesifik untuk koneksi, seperti metode authentikasi ditentukan pada konfigurasi fase satu, kita bisa menggunakan konfigurasi fase dua yang lebih umum. Juga dimungkinkan untuk membuat pengaturan fase dua yang spesifik untuk beberapa host tertentu. Tetapi secara umum, konfigurasi umum akan cukup untuk pengaturan sederhana. Kita juga akan menambahkan path untuk berkas kunci preshared, dan direktori sertifikat. Ini adalah contoh dari `/etc/racoon.conf` dengan path dan aturan tahap kedua yang sudah dibuat:

```
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

sainfo anonymous {
{
 pfs_group 2;
 lifetime time 1 hour;
 encryption_algorithm 3des, blowfish 448, rijndael;
 authentication_algorithm hmac_sha1, hmac_md5;
 compression_algorithm deflate;
}
```

Penanda *`sainfo`* digunakan untuk membuat blok yang menentukan pengaturan untuk asosiasi keamanan. Parameter *`anonymous`* digunakan untuk menentukan bahwa pengaturan ini harus digunakan untuk semua host yang tidak memiliki konfigurasi spesifik. *`pfs_group`* menentukan grup mana dari eksponensial Diffie-Hellman yang harus digunakan. Grup yang berbeda menyediakan panjang angka bilangan prima yang berbeda yang akan digunakan untuk proses authentikasi. Grup 2 menyediakan panjang kunci 1024 bit jika Anda hendak menggunakan panjang yang lebih besar. Untuk keamanan yang lebih baik, Anda bisa menggunakan grup lain (seperti 14 untuk panjang 2048 bit). *`encryption_algorithm`* menentukan algoritma enkripsi apa yang hendak digunakan oleh host untuk enkripsi ESP. *`authentication_algorithm`* menentukan algoritma yang digunakan untuk Authentikasi ESP atau AH. Akhirnya, *`compression_algorithm`* digunakan untuk menentukan algoritma kompresi ketika IPcomp ditentukan pada asosiasi.

Langkah berikutnya adalah menambahkan konfigurasi fase satu untuk pertukaran kunci dengan host lain pada berkas konfigurasi `racoon.conf`. Sebagai contoh:

```
remote 192.168.1.169
{
 exchange_mode aggressive, main;
 my_identifier address;
 proposal {
  encryption_algorithm 3des;
  hash_algorithm sha1;
  authentication_method pre_shared_key;
  dh_group 2;
 }
}
```

Blok *remote* menentukan konfigurasi fase satu. Opsi *exchange_mode* digunakan untuk mengkonfigurasi mode pertukaran apa yang harus digunakan untuk fase. Anda bisa menyebutkan lebih dari satu mode, tetapi mode pertama akan digunakan jika host ini adalah pelaku awal dari pertukaran kunci. Opsi *my_identifier* menentukan penanda apa yang harus dikirimkan pada host remote. Jika opsi ini diabaikan, *address* akan digunakan, yang mengirimkan alamat IP sebagai penanda. Blok *proposal* menentukan parameter yang akan diusulkan ke host lain selama authentikasi fase satu. Opsi *encryption_algorithm*, dan *dh_group* sudah dijelaskan diatas. Opsi *hash_algorithm* bersifat wajib, dan mengkonfigurasi algoritma hash yang harus dipakai. Hal ini bisa berupa *md5*, atau *sha1*. Opsi *authentication_method* sangat penting untuk konfigurasi ini, karena parameter ini digunakan untuk menentukan kunci preshared apa yang harus digunakan, dengan *pre_shared_key*.

Dengan racoon yang sudah diatur, terdapat satu hal lagi yang perlu dilakukan, kunci preshared harus ditambahkan pada `/etc/racoon/psk.txt`. Sintaksnya sangat sederhana, setiap baris berisi alamat IP host dan kunci. Parameter ini dipisahkan dengan tab. Sebagai contoh:

```
192.168.1.169 somekey
```

Mengaktifkan konfigurasi IPsec

Pada tahap ini, konfigurasi dari aturan keamanan dan racoon sudah selesai, dan Anda bisa mulai menguji konfigurasi. Merupakan ide yang bagus untuk menjalankan **racoon** dengan parameter *-F*. Hal ini akan menjalankan **racoon** didepan layar (foreground), membuatnya lebih mudah untuk menangkap pesan kesalahan. Untuk menjalankannya:

```
# setkey -f /etc/setkey.conf
# racoon -F
```

Sekarang karena Anda telah menambahkan aturan keamanan pada basis data, dan menjalankan **racoon**, Anda bisa menguji konfigurasi IPsec Anda. Sebagai contoh, Anda bisa melakukan ping ke host lain sebagai permulaan. Pertama kali Anda melakukan ping, Anda akan menemui kegagalan:

```
$ ping 192.168.1.169
connect: Resource temporarily unavailable
```

Alasannya adalah karena asosiasi keamanan harus dibentuk. Tetapi paket ICMP akan memicu pertukaran kunci. Ping akan memicu pertukaran kunci. Anda bisa melihat apakah pertukaran berhasil atau tidak dengan melihat pada pesan log **racoon** pada `/var/log/messages`, atau hasil keluaran dari terminal jika Anda menjalankan **racoon** pada foreground. Pertukaran kunci yang sukses akan tampak seperti berikut:

```
Apr  4 17:14:58 terrapin racoon: INFO: IPsec-SA request for 192.168.1.169 queued due t
Apr  4 17:14:58 terrapin racoon: INFO: initiate new phase 1 negotiation: 192.168.1.1[5
Apr  4 17:14:58 terrapin racoon: INFO: begin Aggressive mode.
Apr  4 17:14:58 terrapin racoon: INFO: received Vendor ID: DPD
Apr  4 17:14:58 terrapin racoon: NOTIFY: couldn't find the proper pskey, try to get on
Apr  4 17:14:58 terrapin racoon: INFO: ISAKMP-SA established 192.168.1.1[500]-192.168.
Apr  4 17:14:59 terrapin racoon: INFO: initiate new phase 2 negotiation: 192.168.1.1[0
Apr  4 17:14:59 terrapin racoon: INFO: IPsec-SA established: ESP/Transport 192.168.1.1
Apr  4 17:14:59 terrapin racoon: INFO: IPsec-SA established: ESP/Transport 192.168.1.1
```

Setelah pertukaran kunci, Anda bisa memastikan bahwa IPsec sudah dikonfigurasi dengan benar dengan menganalisa paket yang masuk dan keluar dengan **tcpdump**. **tcpdump** tersedia pada set disk *n*. Misalkan koneksi keluar ke host lain melalui antarmuka *eth0*, Anda bisa menganalisa paket yang melalui antarmuka *eth0* dengan **tcpdump -i eth0**. Jika paket keluar dienkripsi dengan ESP, Anda bisa melihatnya pada hasil keluaran **tcpdump**. Sebagai contoh:

```
# tcpdump -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
17:27:50.241067 IP terrapin.taickim.net > 192.168.1.169: ESP(spi=0x059950e8,seq=0x9)
17:27:50.241221 IP 192.168.1.169 > terrapin.taickim.net: ESP(spi=0x0ddff7e7,seq=0x9)
```

Bab 24. Super server internet

24.1. Perkenalan

Terdapat dua cara untuk menawarkan layanan TCP/IP: dengan menjalankan aplikasi server secara mandiri (standalone) sebagai daemon atau dengan menggunakan server super Internet, **inetd**. **inetd** adalah sebuah daemon yang mengawasi sejumlah port. Jika sebuah klien mencoba untuk menghubungkan diri pada sebuah port **inetd** akan menangani koneksi dan mengirimkan koneksi pada perangkat lunak server yang menangani jenis koneksi tersebut. Keuntungan dari pendekatan ini adalah adanya tambahan sebuah layer keamanan dan membuatnya lebih mudah untuk mencatat log pada koneksi yang masuk. Kerugiannya adalah lebih lambat dibandingkan dengan daemon yang sifatnya standalone. Disarankan untuk menjalankan daemon secara standalone pada, misalnya server FTP yang memiliki beban yang tinggi.

24.2. Konfigurasi

inetd dapat dikonfigurasi menggunakan berkas `/etc/inetd.conf`. Mari kita lihat pada contoh baris dari `inetd.conf`:

```
# File Transfer Protocol (FTP) server:
ftp     stream  tcp     nowait  root    /usr/sbin/tcpd  proftpd
```

Baris ini menyatakan bahwa **inetd** harus menerima koneksi FTP dan meneruskannya pada **tcpd**. Ini mungkin cukup aneh, karena **proftpd** biasanya menangani koneksi FTP. Anda juga bisa menyebutkan penggunaan **proftpd** secara langsung pada `inetd.conf`, tetapi Slackware Linux biasanya mengirimkan koneksi pada **tcpd**. Program ini pada gilirannya mengirimkan koneksi pada **proftpd**, seperti yang ditentukan. **tcpd** digunakan untuk memonitor layanan dan menyediakan kontrol akses berbasis host.

Layanan bisa dinonaktifkan dengan menambahkan karakter komentar (#) di awal baris. Disarankan untuk menonaktifkan semua layanan dan hanya mengaktifkan layanan yang Anda perlukan saja. Setelah mengganti `/etc/inetd.conf` **inetd** harus dijalankan kembali untuk mengaktifkan perubahan. Hal ini bisa dilakukan dengan mengirimkan sinyal HUP pada proses inetd:

```
# ps ax | grep 'inetd'
   64 ?        S      0:00 /usr/sbin/inetd
# kill -HUP 64
```

Atau Anda dapat menggunakan skrip inisialisasi `rc.inetd` untuk menjalankan kembali **inetd**:

```
# /etc/rc.d/rc.inetd restart
```

24.3. TCP wrappers

Seperti yang Anda lihat pada `/etc/inetd.conf` koneksi untuk sebagian besar protokol dibuat melalui **tcpd**, bukannya dengan akses langsung pada program layanan. Sebagai contoh:

```
# File Transfer Protocol (FTP) server:
ftp     stream  tcp     nowait  root    /usr/sbin/tcpd  proftpd
```

Pada contoh ini koneksi ftp dikirimkan melalui **tcpd**. **tcpd** mencatat koneksi melalui syslog dan mengijinkan pengujian tambahan. Salah satu fitur yang paling sering dipakai dari **tcpd** adalah kontrol akses berbasis host. Host yang harus

ditolak dapat dikontrol melalui `/etc/hosts.deny`, host yang diijinkan melalui `/etc/hosts.allow`. Kedua berkas ini memiliki satu aturan pada setiap barisnya dengan bentuk seperti berikut:

```
service: hosts
```

Host bisa ditentukan berdasarkan nama host atau alamat IP. Kata kunci ALL menentukan semua host atau semua layanan.

Seandainya kita hendak memblokir semua akses yang dikelola melalui **tcpd**, kecuali untuk host “trusted.example.org”. Untuk melakukan ini, berkas `hosts.deny` dan `hosts.allow` harus dibuat.

`/etc/hosts.deny`:

```
ALL: ALL
```

`/etc/hosts.allow`:

```
ALL: trusted.example.org
```

Dalam `hosts.deny` akses untuk semua layanan (ALL) dan semua host (ALL) diblokir. Tapi `hosts.allow` menentukan bahwa semua layanan (ALL) harus tersedia untuk “trusted.example.org”.

Bab 25. Apache

25.1. Perkenalan

Apache adalah server web yang paling populer sejak April 1996. Aplikasi ini awalnya berbasis pada NCSA httpd, dan telah berkembang menjadi server HTTP yang lengkap dengan fitur-fitur. Slackware Linux saat ini menggunakan seri 1.3.xdari Apache. Bab ini berdasarkan pada Apache 1.3.x.

25.2. Installasi

Apache bisa diinstall dengan menambahkan paket `apache` dari set disk "n". Jika Anda juga hendak menggunakan PHP, paket `php` (set disk "n") dan `mysql` (set disk "ap") juga diperlukan. MySQL diperlukan, karena PHP bergantung dari pustaka MySQL. Anda tidak harus menjalankan MySQL. Setelah diinstall Apache bisa djalankan secara otomatis saat boot dengan membuat berkas `/etc/rc.d/rc.httpd` bersifat executable. Anda bisa melakukan hal ini dengan mengeksekusi:

```
# chmod a+x /etc/rc.d/rc.httpd
```

Konfigurasi Apache bisa diubah pada berkas `/etc/apache/httpd.conf`. Apache bisa di stop/start/restart setiap saat dengan perintah **apachectl**, digabung dengan parameter-parameter *`stop`*, *`start`* dan *`restart`*. Sebagai contoh, jalankan perintah berikut untuk me-restart Apache:

```
# apachectl restart
/usr/sbin/apachectl restart: httpd restarted
```

25.3. Direktori pemakai (user)

Apache menyediakan dukungan untuk direktori pemakai. Ini berarti setiap pemakai mendapat ruang web dalam bentuk *http://host/~user/*. Isi dari "~user/" disimpan pada subdirektori dalam direktori home dari pemakai. Direktori ini dapat ditentukan dengan menggunakan pilihan "UserDir" pada `httpd.conf`, sebagai contoh:

```
UserDir public_html
```

Ini menerangkan bahwa direktori `public_html` digunakan untuk menyimpan halaman-halaman web. Sebagai contoh, halaman web pada URL *http://host/~snail/* disimpan di `/home/snail/public_html`.

25.4. Virtual hosts

Default documentroot untuk Apache di Slackware Linux adalah `/var/www/htdocs`. Tanpa menggunakan virtual hosts setiap klien yang terhubung ke server Apache akan menerima situs web dalam direktori ini. Jadi, jika kita mempunyai dua nama host yang menunjuk ke server "www.example.org" dan "forum.example.org", keduanya akan menampilkan situs web yang sama. Anda dapat membuat situs terpisah untuk nama host yang berbeda dengan menggunakan virtual hosts.

Dalam contoh ini kita akan melihat bagaimana membuat dua virtual host, satu untuk "www.example.org", dengan documentroot `/var/www/htdocs-www`, dan "forum.example.org", dengan documentroot `/var/www/htdocs-forum`. Petama-tama kita harus menentukan Apache harus mendengarkan (listen) di alamat IP mana. Di suatu bagian di berkas konfigurasi `/etc/apache/httpd.conf` Anda akan menemukan baris:

```
#NameVirtualHost *:80
```

Baris ini harus dihilangkan tanda komentarnya untuk bisa menggunakan name-based virtual hosts. Hapus komentar karakter (#) dan ganti parameternya ke “BindAddress IP:port”, atau “BindAddress *:port” jika Anda ingin mem-bind Apache ke semua alamat IP yang dipunyai host. Seandainya kita ingin menyediakan virtual host untuk alamat IP 192.168.1.201 port 80 (default port dari Apache), kita harus mengganti barisnya menjadi:

```
NameVirtualHost 192.168.1.201:80
```

Disuatu bagian dibawah baris NameVirtualHost Anda dapat menemukan contoh virtual host yang ditandai komentar karakter (#) :

```
#<VirtualHost *:80>
#    ServerAdmin webmaster@dummy-host.example.com
#    DocumentRoot /www/docs/dummy-host.example.com
#    ServerName dummy-host.example.com
#    ErrorLog logs/dummy-host.example.com-error_log
#    CustomLog logs/dummy-host.example.com-access_log common
#</VirtualHost>
```

Anda dapat menggunakan contoh ini sebagai pedoman. Sebagai contoh, jika kita ingin menggunakan semua nilai default, dan kita ingin menulis log-log dari kedua virtual host ke default log Apache, kita dapat menambahkan baris ini:

```
<VirtualHost 192.168.1.201:80>
 DocumentRoot /var/www/htdocs-www
 ServerName www.example.org
</VirtualHost>

<VirtualHost 192.168.1.201:80>
 DocumentRoot /var/www/htdocs-forum
 ServerName forum.example.org
</VirtualHost>
```

Bab 26. BIND

26.1. Perkenalan

Domain name system (DNS) digunakan untuk mengkonversi nama host yang mudah diingat (sebagai contoh www.slackware.com) menjadi alamat IP. BIND (Berkeley Internet Name Domain) adalah daemon DNS yang paling banyak digunakan, dan akan dibahas pada bab ini.

Delegasi

Salah satu fitur utama adalah permintaan DNS dapat didelegasikan. Sebagai contoh, misalkan Anda memiliki domain “linuxcorp.com”. Anda bisa menyediakan nameserver yang memiliki otoritas untuk domain ini, nameserver Anda adalah bersifat otoritatif untuk “linuxcorp.com”. Misalkan terdapat cabang dari perusahaan Anda, dan Anda hendak memberikan setiap cabang sebuah otoritas pada zona miliknya masing-masing, ini bukanlah masalah bagi DNS. Anda bisa mendelegasikan DNS untuk misalnya “sales.linuxcorp.com” pada nameserver lain dalam konfigurasi DNS untuk zona “linuxcorp.com”.

Sistem DNS memiliki apa yang disebut server root, yang mendelegasikan DNS untuk jutaan nama domain dan ekstensi (misalnya, ekstensi yang spesifik terhadap sebuah negara, seperti “.nl” atau “.uk”) untuk mengotorisasi server DNS. Sistem ini mengijinkan bentuk pohon delegasi, yang sangat fleksibel dan mendistribusikan beban DNS.

Jenis catatan DNS

Jenis-jenis berikut adalah jenis catatan DNS yang umum:

Tabel 26.1. Catatan DNS

Awalan	Deskripsi
A	Catatan A menunjuk pada alamat IPv4.
AAAA	Catatan AAAA menunjuk pada alamat IPv6.
CNAME	Catatan CNAME menunjuk pada daftar DNS lain.
MX	Catatan MX menentukan yang harus menangani pesan (e-mail) untuk domain tersebut.

Master dan slave

Dua jenis nameserver dapat disediakan untuk sebuah domain: master dan slave. Server DNS master bersifat otoritatif. Server slave mendownload catatan DNSnya dari server master. Menggunakan server slave disamping server master sangatlah disarankan untuk ketersediaan yang tinggi dan juga bisa digunakan untuk pembagian beban.

26.2. Membuat caching nameserver

Sebuah caching nameserver menyediakan layanan DNS untuk sebuah mesin atau jaringan, tetapi tidak menyediakan DNS untuk sebuah domain. Ini berarti bahwa nameserver jenis ini hanya bisa digunakan untuk mengkonversi nama

host menjadi alamat IP. Melakukan setting sebuah nameserver dengan Slackware Linux cukup mudah, karena BIND sudah terkonversi sebagai jenis ini secara default. Mengaktifkan caching nameserver hanya membutuhkan dua langkah: Anda harus menginstall BIND dan mengubah script inisialisasinya. BIND dapat diinstall dengan menambahkan paket BIND dari set disk “n”. Setelah itu, bind bisa dijalankan dengan menjalankan perintah **named**. Jika Anda hendak menjalankan BIND secara default, ganti berkas `/etc/rc.d/rc.bind` menjadi bersifat executable. Hal ini bisa dilakukan dengan menjalankan perintah berikut sebagai root:

```
# chmod a+x /etc/rc.d/rc.bind
```

Jika Anda hendak menggunakan nameserver pada mesin yang menjalankan BIND, Anda harus mengubah `/etc/resolv.conf`.