RPM HOWTO (RPM saat stasioner.....:-))

Donnie Barnes, djb@redhat.com diterjemahkan oleh Julian Adhi .N, adhi@semarang.wasantara.net.id, http://adhi.base.org
v2.0, 8 April 1997 diterjemahkan 30 Desember 1997

Contents

1 Perkenalan

RPM adalah ***R**ed Hat **P**ackage **M**anager* (Pengatur Paket Aplikasi dari Red Hat). Meskipun mengandung nama Red Hat dalam namanya, RPM dimaksudkan untuk menjadi sistem pemaketan terbuka yang tersedia luas untuk digunakan semua orang. RPM mengizinkan para pengguna untuk menggunakan kode sumber (*source code*) untuk perangkat lunak baru dan memaketkannya ke dalam bentuk *source* dan binernya, di mana bentuk binernya mudah dipasang (di-*install*) dan dilacak dan *source*nya dapat dibangun lagi dengan mudah. RPM juga memelihara sebuah database (informasi) dari semua paket dan file-filenya yang dapat digunakan untuk verifikasi paket dan meminta informasi tentang file dan / atau paket.

Perusahaan Perangkat Lunak Red Hat memberikan dorongan kepada vendor-vendor distribusi lain untuk menyempatkan waktu melihat RPM dan menggunakannya untuk distribusi-distribusi mereka. RPM agak fleksibel dan mudah digunakan, meskipun RPM menyediakan pondasi untuk sistem yang sangat luas. RPM juga benar-benar terbuka dan tersedia, namun kami sangat menghargai juga laporan *bug* dan perbaikan-perbaikan. Izin diberikan untuk menggunakan dan mendistribusikan RPM tanpa royalti di bawah GPL.

Dokumen yang lebih lengkap, tersedia pada buku tentang RPM, yang ditulis oleh Ed Bailey, yaitu "Maximum RPM". Buku itu tersedia untuk didownload atau dibeli di www.redhat.com

2 Sekilas tentang RPM

Pertama-tama, biarkan penulis mendudukkan beberapa filosofi di belakang RPM. Salah satu gol desainnya adalah memperbolehkan pengguna untuk menggunakan *source* yang mula-mula. Dengan RPP (pembentuk sistem pemaketan kami di mana tak sebuah RPM pun diperoleh), *source* paket kami adalah *source* yang telah di-*hack*. Secara teoritis, seseorang dapat memasang (meng*install*) sebuah *source* dari RPP dan kemudian membuatnya tanpa masalah. Tetapi *source* itu bukanlah yang asli, dan tak ada referensi seperti perubahan apa yang kami lakukan untuk membuatnya, untuk didapatkan. Seseorang harus mendownload *source* yang mula-mula secara terpisah. Dengan RPM, anda mendapatkan *source* yang asli dengan sebuah *patch* (penambal) yang kami gunakan untuk mengkompilasinya. Kami memandangnya sebagai sebuah keuntungan besar. Mengapa ? Ada beberapa alasan. Pertama, bila sebuah versi program baru datang, anda tidak perlu harus memulai dari *scratch* untuk mendapatkannya terkompilasi di bawah RHL. Anda dapat melihat *patch* untuk memeriksa apakah yang mungkin anda harus lakukan. Semua standar *compile-in* mudah ditampilkan dengan cara ini.

RPM juga didesain untuk mempunyai pilihan permintaan yang kuat. Anda dapat mengerjakan pencarian melewati seluruh database untuk paket atau hanya file tertentu. Anda juga dapat dengan mudah menemukan paket manakah yang memiliki sebuah file dan dari manakah datangnya. File-file RPM itu sendiri adalah arsip yang terkompres (termampatkan), tetapi anda dapat meminta paket secara individual dengan mudah dan cepat, karena sebuah kepala (*header*) biner yang lazim ditambahkan ke paket dengan segala sesuatu yang mungkin anda inginkan yaitu apa yang terkandung dalam bentuk terkompresnya. Ia juga memperbolehkan pencarian cepat.

Kemampuan (*feature*) lain yang kuat adalah kemampuannya untuk memeriksa (*verify*) paket-paket. Bila anda khawatir bahwa anda menghapus sebuah file penting untuk beberapa paket, lakukan verifikasi dengan RPM. Anda akan diberitahu penyimpangan-penyimpangan apa pun. Di saat itu, anda dapat memasang ulang (*reinstall*) paket bila diperlukan. File konfigurasi apa pun yang anda punyai, diamankan dengan baik.

Kami ingin mengucapkan terima kasih kepada orang-orang dari distribusi BOGUS untuk ide-ide dan konsep yang tergabungkan dalam RPM. Pada saat RPM telah ditulis secara lengkap oleh Red Hat Software, operasinya berdasarkan pada kode yang ditulis oleh BOGUS (PM dan PMS)

3 Informasi Umum

3.1 Cara mendapatkan RPM

Cara terbaik mendapatkan RPM adalah memasang (meng*install*) Red Hat Linux. Bila anda tidak ingin mengerjakan itu, anda masih dapat memperoleh dan menggunakan RPM. RPM dapat diperoleh dari *ftp://ftp.redhat.com* `<ftp://ftp.redhat.com/pub/redhat/code/rpm>`

3.2 Syarat-syarat yang diperlukan RPM

Syarat utama untuk menjalankan RPM adalah cpio 2.4.2 atau lebih tinggi. Pada saat sistem ini dimaksudkan untuk digunakan dengan Linux, mungkin sangat bagus digunakan di dalam sistem Unix yang lain. RPM punya ... dalam faktanya, kompilasi untuk SunOS, Solaris, AIX, Irix, AmigaOS, dan lainnya. Penulis ingatkan, paket biner yang dibuat di sistem Unix yang berbeda tidaklah saling kompatibel.

Itu semua adalah syarat minimal untuk memasang RPM. Untuk membangun (*build*) RPM dari *source*, anda juga membutuhkan segala sesuatu yang dalam keadaan normal diperlukan untuk membangun sebuah paket, seperti gcc, make, dll.

4 Menggunakan RPM

Dalam bentuknya yang paling sederhana, RPM dapat digunakan untuk meng*install* paket :

```
rpm -i foobar-1.0-1.i386.rpm
```

Perintah sederhana berikutnya adalah untuk meng-*uninstall* sebuah paket :

```
rpm -e foobar
```

Salah satu yang lebih kompleks tetapi perintah yang sangat berguna mengizinkan anda untuk meng*install* paket via FTP. Bila anda terkoneksi ke jaringan dan menginginkan meng*install* sebuah paket baru, semua yang anda inginkan adalah memerinci file itu dengan URL yang valid, seperti :

`rpm -i ftp://ftp.pht.com/pub/linux/redhat/rh-2.0-beta/RPMS/foobar-1.0-1.i386.rpm`

Catatan, RPM akan meminta dan / atau meng*install* lewat FTP.

Meskipun perintah tersebut adalah perintah yang sederhana, rpm dapat digunakan dalam sejumlah besar cara seperti yang dapat dilihat dari pesan penggunaan :

```
RPM version 2.3.9
Copyright (C) 1997 - Red Hat Software
This may be freely redistributed under the terms of the GNU Public License

usage: rpm {--help}
       rpm {--version}
       rpm {--initdb}   [--dbpath <dir>]
       rpm {--install -i} [-v] [--hash -h] [--percent] [--force] [--test]
                        [--replacepkgs] [--replacefiles] [--root <dir>]
                        [--excludedocs] [--includedocs] [--noscripts]
                        [--rcfile <file>] [--ignorearch] [--dbpath <dir>]
                        [--prefix <dir>] [--ignoreos] [--nodeps]
                        [--ftpproxy <host>] [--ftpport <port>]
                        file1.rpm ... fileN.rpm
       rpm {--upgrade -U} [-v] [--hash -h] [--percent] [--force] [--test]
                        [--oldpackage] [--root <dir>] [--noscripts]
                        [--excludedocs] [--includedocs] [--rcfile <file>]
                        [--ignorearch]  [--dbpath <dir>] [--prefix <dir>]
                        [--ftpproxy <host>] [--ftpport <port>]
                        [--ignoreos] [--nodeps] file1.rpm ... fileN.rpm
       rpm {--query -q} [-afpg] [-i] [-l] [-s] [-d] [-c] [-v] [-R]
                        [--scripts] [--root <dir>] [--rcfile <file>]
                        [--whatprovides] [--whatrequires] [--requires]
                        [--ftpuseport] [--ftpproxy <host>] [--ftpport <port>]
                        [--provides] [--dump] [--dbpath <dir>] [targets]
       rpm {--verify -V -y} [-afpg] [--root <dir>] [--rcfile <file>]
                        [--dbpath <dir>] [--nodeps] [--nofiles] [--noscripts]
                        [--nomd5] [targets]
       rpm {--setperms} [-afpg] [target]
       rpm {--setugids} [-afpg] [target]
       rpm {--erase -e} [--root <dir>] [--noscripts] [--rcfile <file>]
                        [--dbpath <dir>] [--nodeps] [--allmatches]
                        package1 ... packageN
       rpm {-b|t}[plciba] [-v] [--short-circuit] [--clean] [--rcfile  <file>]
                        [--sign] [--test] [--timecheck <s>] specfile
       rpm {--rebuild} [--rcfile <file>] [-v] source1.rpm ... sourceN.rpm
       rpm {--recompile} [--rcfile <file>] [-v] source1.rpm ... sourceN.rpm
       rpm {--resign} [--rcfile <file>] package1 package2 ... packageN
       rpm {--addsign} [--rcfile <file>] package1 package2 ... packageN
       rpm {--checksig -K} [--nopgp] [--nomd5] [--rcfile <file>]
                           package1 ... packageN
```

```
rpm {--rebuilddb} [--rcfile <file>] [--dbpath <dir>]
rpm {--querytags}
```

Anda dapat menemukan lebih detil dalam 'what' untuk pilihan-pilihan tersebut di atas dalam halaman manual (*man pages*) RPM.

5 Sekarang apa yang *dapat* anda lakukan dengan RPM ?

RPM adalah alat yang sangat berguna, dan seperti yang anda lihat, punya beberapa pilihan (*option*). Jalan terbaik untuk menjelaskannya adalah dengan melihat beberapa contoh. Saya menyertakan contoh pemasangan/penghapusan (*install/uninstall*) di atas, dengan demikian di sini adalah beberapa contoh lain :

- Marilah beranggapan anda menghapus beberapa file secara tak sengaja, tetapi anda tidak mengetahui dengan pasti apa yang anda hapus. Bila anda menginginkan melakukan verifikasi terhadap sistem anda secara keseluruhan dan melihat apa yang mungkin hilang, anda akan mengetik :

  ```
  rpm -Va
  ```

- Mari beranggapan anda menjalankan sebuah file yang anda tidak kenali. Untuk mencari milik paket mana file itu, ketikkan :

  ```
  rpm -qf /usr/X11R6/bin/xjewel
  ```

  Keluarannya mungkin seperti :

  ```
  xjewel-1.6-1
  ```

- Anda menemukan sebuah paket RPM koules, tetapi anda tak tahu apakah itu. Untuk menemukan beberapa informasi padanya, kerjakan :

  ```
  rpm -qpi koules-1.2-2.i386.rpm
  ```

  Keluarannya mungkin seperti ini :

  ```
  Name        : koules                      Distribution: Red Hat Linux Colgate
  Version     : 1.2                               Vendor: Red Hat Software
  Release     : 2                             Build Date: Mon Sep 02 11:59:12 1996
  Install date: (none)                        Build Host: porky.redhat.com
  Group       : Games                         Source RPM: koules-1.2-2.src.rpm
  Size        : 614939
  Summary     : SVGAlib action game with multiplayer, network, and sound support
  Description :
  This arcade-style game is novel in conception and excellent in execution.
  No shooting, no blood, no guts, no gore.  The play is simple, but you
  still must develop skill to play.  This version uses SVGAlib to
  run on a graphics console.
  ```

- Sekarang anda ingin melihat file apakah yang diisntall oleh RPM koules. Anda harus mengerjakan ini :

  ```
  rpm -qpl koules-1.2-2.i386.rpm
  ```

Keluarannya adalah :

```
/usr/doc/koules
/usr/doc/koules/ANNOUNCE
/usr/doc/koules/BUGS
/usr/doc/koules/COMPILE.OS2
/usr/doc/koules/COPYING
/usr/doc/koules/Card
/usr/doc/koules/ChangeLog
/usr/doc/koules/INSTALLATION
/usr/doc/koules/Icon.xpm
/usr/doc/koules/Icon2.xpm
/usr/doc/koules/Koules.FAQ
/usr/doc/koules/Koules.xpm
/usr/doc/koules/README
/usr/doc/koules/TODO
/usr/games/koules
/usr/games/koules.svga
/usr/games/koules.tcl
/usr/man/man6/koules.svga.6
```

Ini semua adalah contoh belaka. Orang yang kreatif dapat berpikir tentang kemudahannya, sekali anda telah mengenal (familiar) [dengan] RPM.

6 Membuat File RPM Sendiri

Membangun RPM adalah cukup mudah untuk dikerjakan, khususnya bila anda dapat mengambil perangkat lunak yang anda coba paketkan untuk membangun dirinya.

Prosedur dasar untuk membangun RPM sebagai berikut :

- Pastikan `/etc/rpmrc` telah diset untuk sistem anda.
- Pastikan kode sumber (*source code*) yang dibutuhkan untuk membangun RPM ada dalam sistem anda.
- Buat tambalan (*patch*) dari setiap perubahan yang harus anda buat ke dalam *source* untuk mendapatkannya dibangun dengan benar.
- Buat sebuah file spesikasi untuk paket.
- Pastikan segala sesuatunya dalam tempat yang benar.
- Bangun (*build*) paketnya dengan RPM.

Dalam operasi yang normal, RPM membangun baik paket dalam biner maupun dalam *source*.

6.1 File rpmrc

Sampai saat ini, konfigurasi RPM hanya tersedia via file `/etc/rpmrc`. Sebuah contoh, seperti :

```
require_vendor: 1
distribution: I roll my own!
require_distribution: 1
topdir: /usr/src/me
vendor: Mickiesoft
packager:  Mickeysoft Packaging Account <packages@mickiesoft.com>

optflags: i386 -O2 -m486 -fno-strength-reduce
optflags: alpha -O2
optflags: sparc -O2

signature: pgp
pgp_name: Mickeysoft Packaging Account
pgp_path: /home/packages/.pgp

tmppath: /usr/tmp
```

Baris `require_vendor` menyebabkan RPM mensyaratkan bahwa ia menemukan baris vendor. Ini dapat terjadi dari `/etc/rpmrc` atau dari *header* file spesifikasi itu sendiri. Untuk menonaktifkannya, ganti nomor ke 0. Hal yang sama untuk baris require_distribution dan require_group.

Baris berikutnya adalah baris distribution. Anda dapat mendefinisikannya di sini atau nanti di *header* file spesifikasi. Saat membangun sebuah distribusi tertentu, adalah ide bagus untuk memastikan baris ini adalah benar, bahkan meski tidak disyaratkan. Baris vendor bekerja dengan cara yang sama, tetapi dapat berupa apa pun (sebagai contoh Joe's Software and Rock Music Emporium).

RPM sekarang juga mendukung paket dalam banyak arsitektur. File /rpmrc dapat mengendalikan sebuah variabel "optflags" untuk membangun sesuatu yang memerlukan arsitektur yang spesifik. Lihat bagian berikutnya untuk mengetahui bagaimana menggunakan variabel ini.

Dalam tambahan untuk makro yang di atas, ada beberapa lagi. Anda dapat menggunakan :

```
rpm --showrc
```

untuk menemukan bagaimana tag ter-set dan flag apa saja yang tersedia.

6.2 File Spesifikasi

Kami akan memulai diskusi tentang file spesifikasi. File spesifikasi diperlukan untuk membangun sebuah paket. File spesisikasi adalah gambaran tentang perangkat lunak dengan instruksinya tentang bagaimana untuk membangunnya dan sebuah daftar file untuk semua file biner yang ter*install*.

Anda akan menginginkan menamai file spesifikasi anda, mengacu kepada konvensi standar. Namanya seharusnya seperti ini nama paket-tanda hubung-nomor versi-tanda hubung-nomor rilis-titik-spec.

Ini contoh file spesifikasi yang kecil (vim-3.0-1.spec):

```
Summary: ejects ejectable media and controls auto ejection
Name: eject
Version: 1.4
Release: 3
```

```
Copyright: GPL
Group: Utilities/System
Source: sunsite.unc.edu:/pub/Linux/utils/disk-management/eject-1.4.tar.gz
Patch: eject-1.4-make.patch
Patch1: eject-1.4-jaz.patch
%description
This program allows the user to eject media that is autoejecting like
CD-ROMs, Jaz and Zip drives, and floppy drives on SPARC machines.

%prep
%setup
%patch -p1
%patch1 -p1

%build
make RPM_OPT_FLAGS="$RPM_OPT_FLAGS"

%install
install -s -m 755 -o 0 -g 0 eject /usr/bin/eject
install -m 644 -o 0 -g 0 eject.1 /usr/man/man1

%files
%doc README COPYING ChangeLog

/usr/bin/eject
/usr/man/man1/eject.1
```

6.3 Header

Header mempunyai field standar yang anda perlu isi. Ada sedikit keberatan memang. Fieldnya harus diisi seperti ini :

- `Summary`: Ini adalah satu baris yang menggambarkan paket anda.
- `Name`: Ini harusnya nama string dari nama file rpm yang anda rencanakan untuk digunakan.
- `Version`: Ini adalah string versi dari file rpm yang anda rencanakan untuk dipakai.
- `Release`: Ini adalah nomor rilis untuk versi paket yang sama (contoh, bila kita membuat sebuah paket dan menjumpainya agak rusak dan ingin membuatnya lagi, paket berikutnya akan mempunyai nomor rilis 2).
- `Icon`: Ini adalah nama icon yang digunakan untuk perangkat instalasi level tinggi yang lain (seperti "glint"-nya Red Hat). Filenya harus berupa file .gif dan ditempatkan di direktori SOURCES.
- `Source`: Baris ini menunjuk ke lokasi HOME dari file *source* yang asli. Baris ini dipakai bila anda ingin mendapatkan *source* lagi atau mengecek untuk versi yang lebih baru. Keberatannya : Nama file pada baris ini HARUS cocok dengan nama file yang anda punyai dalam sistem anda (contoh, jangan mendownload file *source* dan mengganti namanya). Anda dapat memberi spesifikasi lebih dari satu file *source* menggunakan baris seperti :

```
Source0: blah-0.tar.gz
Source1: blah-1.tar.gz
Source2: fooblah.tar.gz
```

File-file ini akan ada di direktori SOURCES. (Struktur direktori akan didiskusikan dalam bagian berikutnya, "Hirarki Direktori 'Source'"

- `Patch`: Ini akan mengeset tempat dimana anda dapat menemukan tambalannya (*patch*-nya), bila anda membutuhkannya untuk mendowloadnya lagi. Keberatan : Nama filenya harus cocok dengan yang anda gunakan saat anda membuat *patch* anda. Anda mungkin juga menginginkan mempunyai lebih dari satu file *patch* seperti anda punya banyak *source*. Anda harus mengerjakan sesuatu seperti ini :

```
Patch0: blah-0.patch
Patch1: blah-1.patch
Patch2: fooblah.patch
```

File-file ini akan ada di direktori SOURCES.

- `Copyright`: Baris ini memberitahu bagaimana status hak cipta sebuah paket. Anda harus menggunakan sesuatu seperti GPL, BSD, MIT, public domain, distributable, atau commercial.
- `BuildRoot`: Baris ini mengizinkan anda untuk menspesifikasikan sebuah direktori sebagai *root* (akar) untuk membangun dan meng*install*paket baru. Anda dapat menggunakannya untuk mengetes paket anda sebelum meng*install*nya dalam mesin anda.
- `Group`: Baris ini digunakan untuk memberitahu program instalasi level tinggi (seperti "glint"-nya Red Hat) ke mana menempatkan program tertentu dalam hirarki struktur. Grup pohon (*tree*) kelihatan seperti ini:

```
Applications
    Communications
    Editors
        Emacs
    Engineering
    Spreadsheets
    Databases
    Graphics
    Networking
    Mail
    Math
    News
    Publishing
        TeX
Base
    Kernel
Utilities
    Archiving
    Console
    File
    System
    Terminal
    Text
Daemons
```

```
Documentation
X11
    XFree86
        Servers
    Applications
        Graphics
        Networking
    Games
        Strategy
        Video
    Amusements
    Utilities
    Libraries
    Window Managers
Libraries
Networking
    Admin
    Daemons
    News
    Utilities
Development
    Debuggers
    Libraries
        Libc
    Languages
        Fortran
        Tcl
    Building
    Version Control
    Tools
Shells
Games
```

- `%description` Ini bukanlah benar-benar item *header*, tetapi seharusnya digambarkan di akhir *header*. Anda membutuhkan satu tag gambaran per paket dan / atau sub-paket. Ini adalah field multi-baris yang harus digunakan untuk memberikan sebuah gambaran yang lengkap dari paket itu.

6.4 Persiapan

Ini adalah bagian kedua dalam file spesifikasi. Ini digunakan untuk mendapatkan *source* yang siap dibangun. Di sini anda memerlukan apa pun yang dibutuhkan untuk mendapatkan *source* telah tertambal dan setup seperti yang mereka inginkan dikerjakan dengan 'make'.

Satu hal yang harus dicatat : Setiap bagian ini adalah benar-benar hanya sebuah tempat untuk mengeksekusi script shell. Anda dapat membuat sebuah script sh dengan mudah dan menaruhnya setelah tag `%prep` untuk melakukan unpack dan menambal *source* anda. Kami membuat makro untuk membantu dalam hal ini, bagaimana pun juga.

Yang pertama dari makro-makro ini adalah makro `%setup`. Dalam bentuk yang paling sederhana (tanpa pilihan baris perintah), dia akan melakukan unpack *source* dan melakukan cd ke direktori *source*. Hal itu juga akan disertai pilihan (*option*) :

- `-n name` akan mengeset nama dari direktori yang dibuat kepada nama yang terdaftar. Standarnya adalah `$NAME-$VERSION`. Kemungkinan lain termasuk `$NAME, ${NAME}${VERSION}`, atau apa pun file tar utama gunakan. (Harap dicatat bahwa variabel "$" di sini adalah bukan variabel yang nyata yang tersedia di bawah file spesifikasi. Mereka sesungguhnya digunakan di tempat nama sampel. Anda perlu menggunakan nama asli dan versi dalam paket anda, bukan sebuah variabel.)
- `-c` akan membuat dan melakukan cd kepada direktori bernama sebelum mengerjakan proses untar.
- `-b` akan melakukan untar Source# sebelum berpindah direktori ke dalamnya (dan ini membuat tak berlaku dengan -c karena itu jangan kerjakan ini). Ini hanya berguna untuk paket yang mempunyai *source* banyak.
- `-a` # akan melakukan untar Source# sesudah berpindah ke direktorinya.
- `-T` pilihan ini melakukan overrides aksi standar dari proses untarring *source* dan memerlukan pilihan -b 0 atau -a 0 untuk mendapatkan file *source* utama dikenai proses untar. Anda memerlukannya saat ada *source* kedua.
- `-D` Jangan menghapus direktori sebelum unpacking. Ini hanya berguna di mana anda mempunyai lebih dari satu makro setup. Ini seharusnya digunakan dalam makro setup sesudah yang pertama (tetapi jangan pernah di makro yang pertama).

Makro berikutnya yang tersedia adalah makro `%patch`. Makro ini membantu mengotomatiskan proses penerapan *patch* ke *source*. Makro ini memerlukan beberapa pilihan, diperlihatkan di bawah ini :

- `#`; akan menerapkan Patch# sebagai file penambal (*patch*).
- `-p #`; memerinci sejumlah direktori untuk dibuka untuk perintah patch(1).
- `-P` Aksi standarnya adalah menerapkan Patch (atau Patch0). Tanda ini menahan aksi standar dan akan memerlukan sebuah 0 untuk mendapatkan file *source* utama di-untar. Pilihan ini berguna dalam sedetik atau lebih makro *patch* yang memerlukan sebuah nomor berbeda dibandingkan dengan makro pertama.
- Anda dapat juga mengerjakan `%patch#` sebagai ganti dari mengetik perintah asli: `%patch # -P`

Itu seharusnya adalah semua makro yang anda perlukan. Setelah anda meyakinkan semuanya benar, anda dapat juga mengerjakan setup yang lain yang anda perlukan untuk dikerjakan via script bertipe `sh`. Apa pun yang anda sertakan sampai makro %build (didiskusikan dalam bagian berikutnya) adalah menjalankannya via sh. Lihat contoh di atas untuk ketikan dari sesuatu yang anda mungkin inginkan untuk dikerjakan di sini.

6.5 Membangun (Build)

Pada dasarnya tak ada makro apa pun untuk bagian ini. Anda seharusnya hanya meletakkan perintah apa pun di sini yang akan anda pakai untuk membangun perangkat lunak, sekali anda melakukan untar *source*nya, menambalnya dan melakukan cd ke dalam direktori. Ini hanyalah set perintah yang lain yang dilewatkan ke sh, sehingga perintah sah (legal) apa pun dapat pergi ke sini (termasuk komentar). Direktori kerja anda saat ini di-reset dalam setiap bagian ini ke tingkat atas dari direktori sumber, jadi ingatlah itu baik-baik. Anda dapat melakukan cd ke dalam subdirektori bila diperlukan.

6.6 Memasang (*Install*)

Sebenarnya tak ada makro apa pun di sini. Anda hanya ingin meletakkan perintah apa pun di sini yang diperlukan untuk meng*install*. Bila anda punya 'make install' tersedia untuk anda dalam paket yang anda bangun, letakkan itu di sini. Bila tidak, anda dapat menambal makefile untuk sebuah make install dan hanya mengerjakan sebuah 'make install' di sini, atau anda dapat meng*install* mereka secara manual dengan perintah sh. Anda dapat mempertimbangkan direktori anda saat ini untuk menjadi tingkat atas dari direktori *source*.

6.7 Pilihan Script *Install*/ *Uninstall* pre dan post

Anda dapat membuat script dijalankan sebelum dan sesudah instalasi dan uninstalasi dari paket biner. Sebuah alasan utama untuk itu adalah mengerjakan sesuatu seperti menjalankan ldconfig setelah instalasi atau memindahkan paket yang berisikan libraries / kepustakaan yang dipakai bersama (*shared*). Makro untuk setiap script adalah sebagai berikut :

- `%pre` adalah makro untuk mengerjakan script pre-install.
- `%post` adalah makro untuk mengerjakan script post-install.
- `%preun` adalah makro untuk mengerjakan script pre-uninstall.
- `%postun` adalah makro untuk mengerjakan script post-uninstall.

Isi dari bagian ini seharusnya hanyalah beberapa bentuk dari script, meski anda tidak membutuhkan `#!/bin/sh`.

6.8 File-file

Ini adalah bagian di mana anda harus melihat file-file paket biner. RPM tak punya cara lain untuk mengetahui file biner apa yang sudah di*install* sebagai hasil dari make install. Ada jalan untuk mengetahuinya. Beberapa menyarankan untuk melakukan find sebelum dan sesudah paket di*install*. Dengan sistem *multiuser* (banyak pemakai), ini tak dapat diterima seperti file lain mungkin telah dibuat selama sebuah proses pembangunan paket yang menyebabkan tak ada yang dikerjakan dengan paket itu sendiri.

Ada beberapa makro tersedia untuk mengerjakan beberapa hal yang istimewa. Mereka terdaftar dan digambarkan di sini :

- `%doc` digunakan untuk menandai dokumentasi dalam paket *source* yang anda ingin *install* di dalam instalasi biner. Dokumen akan dipasang dalam /usr/doc/$NAME-$VERSION-$RELEASE. Anda dapat melihat banyak dokumen dengan perintah baris dengan makro ini, atau anda dapat melihatnya semua secara terpisah dengan menggunakan sebuah makro untuk setiap dokumennya.
- `%config` digunakan untuk menandai file konfigurasi dalam sebuah paket. Ini menyertakan file seperti sendmail.cf, passwd, dll. Bila anda kemudian melakukan *uninstall* sebuah paket berisikan file konfigurasi, semua file yang tak berubah akan dihapus dan file yang berubah akan dipindahkan ke nama lama mereka dengan sebuah .rpmsave ditambahkan ke nama file-nya. Anda dapat melihat banyak file dengan makro ini.

- `%dir` menandai sebuah direktori tunggal dalam sebuah daftar file untuk disertakan sebagaimana dimilki oleh sebuah paket. Standarnya, bila anda melihat sebuah nama direktori TANPA sebuah makro %dir, SEGALA SESUATU dalam direktori itu disertakan dalam daftar file dan kemudian dipasang (*installed*) sebagai bagian dari paket itu.
- `%files -f <filename>` akan mengizinkan anda untuk melihat file anda dalam beberapa file arbitrary di dalam direktori pembangunan *source*.Ini menyenangkan dalam kasus di mana anda mempunyai sebuah paket yang dapat membangun daftar file miliknya. Anda kemudian hanyalah menyertakan daftar file di sini dan anda tidak harus mendaftarkan file secara spesifik.

Keberatan terbesar dalam daftar file adalah daftar direktori-direktori. Bila anda mendaftarkan `/usr/bin` secara tak sengaja, paket biner anda akan berisi setiap file dalam `/usr/bin` di sistem anda.

6.9 Cara membangunnya !

6.9.1 'Source' Pohon Direktori

Pertama kali yang anda butuhkan adalah *build tree* yang terkonfigurasi dengan benar. Ini dapat dikonfigurasi menggunakan file `/etc/rpmrc`. Sebagian besar orang akan hanya menggunakan /usr/src.

Anda mungkin memerlukan membuat direktori selanjutnya untuk membuat sebuah *build tree* :

- `BUILD` adalah direktori di mana semua proses pembangunan terjadi oleh RPM. Anda tak harus mengerjakan test anda untuk dibangun di mana saja pada khususnya, tetapi ini adalah di mana RPM akan mengerjakannya sendiri.
- `SOURCES` adalah direktori di mana anda seharusnya meletakkan file tar *source* asli anda dan *patch* anda. Ini adalah tempat di mana RPM akan mencari dalam keadaan standar.
- `SPECS` adalah direktori di mana semua file yang spesifikasi harus berada.
- `RPMS` adalah tempat di mana RPM akan meletakkan semua file biner RPM saat proses pembangunan.
- `SRPMS` adalah tempat di mana semua *source* RPM akan diletakkan.

6.9.2 Mengetes pembangunan

Pertama kali yang anda mungkin akan inginkan adalah mengambil *source* untuk membangun secara bersih tanpa RPM. Untuk mengerjakan ini, lakukan 'unpack' pada *source*, dan rubah nama direktori ke`$NAME.orig`. Lalu lakukan 'unpack' pada *source* lagi. Gunakan *source* ini untuk membangunnya. Pergilah ke dalam direktori *source* dan ikuti instruksi untuk membangunnya. Bila anda harus menyunting sesuatu, anda akan membutuhkan sebuah *patch*. Sekali anda memilihnya untuk dibangun, bersihkan direktori *source*nya. Pastikan dan hapus file apa pun yang diambil dari sebuah script `configure`. Lalu lakukan `cd` kembali, keluar dari direktori *source* ke induknya. Kemudian anda akan mengerjakan sesuatu seperti :

```
diff -uNr dirname.orig dirname >../SOURCES/dirname-linux.patch
```

Ini akan membuat sebuah *patch* untuk anda yang anda gunakan dalam file spesifikasi anda. Catatan bahwa “linux” yang anda lihat dalam nama *patch* adalah hanya sebuah identifier. Anda mungkin ingin menggunakan sesuatu yang lebih bisa menggambarkan seperti "config" atau *bugs* untuk menggambarkan mengapa anda harus membuat sebuah *patch*.

Sebuah ide yang bagus juga untuk melihat file *patch* yang anda buat sebelum menggunakannya untuk memastikan tak ada file biner yang terselip secara tak sengaja.

6.9.3 Membuat Daftar File

Sekarang, anda telah mempunyai *source* yang akan dibangun dan tahu bagaimana mengerjakannya, membangunnya, dan memasangnya. Lihatlah keluaran dari urutan pemasangan dan membangun daftar file anda dari hal itu untuk digunakan dalam file spesifikasi. Kami pada umumnya membangun file spesifikasi secara paralel dengan semua langkah ini. Anda dapat membuat salah satu inisial dan mengisinya ke dalam bagian yang mudah, dan lalu mengisikan ke dalam langkah lain sebagaimana anda lakukan.

6.9.4 Membangun sebuah Paket dengan RPM

Sekali anda punya sebuah file spesifikasi, anda siap untuk mencoba dan membangun paket anda. Cara yang paling bermanfaat untuk mengerjakan ini adalah dengan sebuah perintah seperti :

```
rpm -ba foobar-1.0.spec
```

Ada pilihan lain yang berguna dengan switch `-b` seperti :

- `p` berarti kerjakan bagian persiapan dari file spesifikasi.
- `l` adalah sebuah cek daftar yang mengerjakan beberapa pengecekan atas `%files`.
- `c` mengerjakan prep dan compile. Ini berguna manakala anda tak yakin kalau-kalau *source* akan dibangun seluruhnya. Kelihatannya tak berguna karena anda mungkin ingin 'memainkan' *source*nya sendiri sampai ia dibangun dan kemudian memulai menggunakan RPM, tetapi sekali anda menjadi terbiasa menggunakan RPM, anda akan menemukan kemudahan dalam menggunakannya.
- `i` mengerjakan persiapan, kompilasi dan instalasi.
- `b` persiapan, kompilasi, instalasi dan membangun sebuah paket biner saja.
- `a` membangun seluruhnya (baik *source* dan paket binernya).

Ada beberapa modifier dalam switch `-b`, yaitu :

- `-short-circuit` akan melompat langsung ke tahap yang telah ditentukan (hanya dapat digunakan dengan c dan i).
- `-clean` menghapus bangunan pohon saat usai.
- `-keep-temps` akan mengamankan semua file temporer dan script yang dibuat dalam /tmp. Anda sesungguhnya dapat melihat file-file apa yang dibuat dalam /tmp menggunakan option `-v`.
- `-test` tidak menjalankan tingkat yang sebenarnya, tetapi tetap mengamankan -temp.

6.10 Pengetesan

Sekali anda telah memiliki sebuah rpm *source* dan biner untuk paket anda, anda perlu mengetesnya. Cara paling mudah dan terbaik adalah menggunakan mesin yang benar-benar berbeda dari seseorang di mana anda membangun di atasnya dalam rangka mengetes. Sesudah itu, anda hanya harus mengerjakan make install di mesin anda sendiri, sehingga seharusnya ia sudah terpasang dengan cukup baik.

Anda dapat mengetik `rpm -u namapaket` atas sebuah paket untuk mengetesnya, tetapi cara itu tidak dapat dipercaya, karena dalam membangun sebuah paket, anda telah mengerjakan `make install`. Bila anda meninggalkan sesuatu di luar daftar file anda, ia tidak akan di-*uninstall*. Anda akan kemudian meng*install* ulang paket biner dan sistem anda akan menjadi lengkap lagi, tetapi rpm anda masih belum lengkap. Pastikan dan ingat-ingat bahwa hanya karena anda melakukan `rpm -ba namapaket`, kebanyakan orang memasang paket anda akan hanya mengerjakan `rpm -i namapaket`. Patikan anda tidak mengerjakan apa pun dalam bagian pemasangan atau *install* yang akan diperlukan untuk dikerjakan saat binernya terpasang secara otomatis.

6.11 Apa dapat yang dikerjakan dengan RPM baru anda

Sekali anda telah membuat RPM anda sendiri atas sesuatu (diasumsikan sesuatu yang belum pernah di-RPM-kan), anda dapat melakukan kontribusi pekerjaan anda ke yang lain (juga diasumsikan anda me-RPM-kan sesuatu yang didistribusikan dengan bebas). Untuk mengerjakan itu, anda ingin melakukan upload paketnya ke *ftp://ftp.redhat.com* `<ftp://ftp.redhat.com>`

6.12 Sekarang Mau Apa Lagi?

Silakan lihat bagian di atas dalam Pengetesan dan Apa yang dikerjakan dengan RPM baru anda. Kami ingin semua ketersediaan RPM dapat kami ambil, dan kami ingin 'mereka' menjadi RPM yang baik. Silakan ambil waktu untuk mengetes mereka sebaik mungkin, dan kemudian ambil waktu untuk meng-uploadnya untuk keuntungan semua orang. Juga, silakan memastikan anda hanya melakukan upload perngkat lunak yang tersedia dengan gratis. Perangkat lunak komersial dan shareware tak seharusnya di-upload kecuali mereka punya hak cipta dan dinyatakan bahwa hal ini diizinkan. Ini termasuk perangkat lunak dari Netscape, ssh, pgp, dll.

7 Pembangunan RPM multi-arsitektur

Sekarang RPM dapat digunakan untuk membangun paket untuk Intel i386, Digital Alpha yang menjalankan Linux, dan Sparc. RPM juga dilaporkan bekerja dalam SGI dan workstation HP dengan baik. Ada beberapa kemampuan yang membuat pembangunan dalam *platform-platform* tersebut menjadi mudah, Pertama adalah directive "optflags" yang ada dalam `/etc/rpmrc`. Ini dapat digunakan untuk mengeset flag yang digunakan saat membangun perangkat lunak ke arsitektur yang spesifik. Kemampuan lain adalah makro "arch" dalam file spesifikasi. Mereka dapat digunakan untuk mengerjakan sesuatu yang berbeda berdasarkan atas arsitektur dimana anda membangunnya. Kemampuan yang lain lagi adalah directive "Exclude" dalam *header* / kepalanya.

7.1 Contoh file spesikasi

Bagian di bawah ini adalah file spesifikasi dari paket "fileutils". Ini adalah setup untuk dibangun di atas Alpha dan Intel.

```
Summary: GNU File Utilities
Name: fileutils
Version: 3.16
Release: 1
Copyright: GPL
Group: Utilities/File
Source0: prep.ai.mit.edu:/pub/gnu/fileutils-3.16.tar.gz
Source1: DIR_COLORS
Patch: fileutils-3.16-mktime.patch

%description
These are the GNU file management utilities.  It includes programs
to copy, move, list, etc, files.

The ls program in this package now incorporates color ls!

%prep
%setup

%ifarch alpha
%patch -p1
autoconf
%endif
%build
configure --prefix=/usr --exec-prefix=/
make CFLAGS="$RPM_OPT_FLAGS" LDFLAGS=-s

%install
rm -f /usr/info/fileutils*
make install
gzip -9nf /usr/info/fileutils*

.
.
.
```

7.2 Optflags

Dalam contoh tersebut, anda melihat bagaimana directive "optflags" digunakan dalam `/etc/rpmrc`. Bergantung kepada arsitektur mana anda membangunnya, harga yang wajar diberikan kepada RPM_OPT_FLAGS. Anda harus menambal Makefile untuk paket anda untuk menggunakan variabel ini di tempat directive yang normal yang mungkin anda gunakan seperti -m486 dan -O2. Anda dapat merasakan perasaan yang nyaman untuk kebutuhan apa yang harus dikerjakan dengan meng*install* paket *source* ini dan kemudian melakukan unpack *source* dan menguji Makefile. Setelah itu lihatlah *patch* untuk Makefile dan lihat perubahan apa yang harus dibuat.

7.3 Makro

Makro `%ifarch` sangat penting secara keseluruhan. Banyak waktu akan diperlukan untuk membuat sebuah *patch* atau dua yang spesifik ke satu arsitektur saja. dalam kasus ini, RPM akan mengizinkan anda untuk menerapkan *patch* itu untuk satu arsitektur saja.

Dalam contoh di atas fileutils mempunyai *patch* untuk mesin 64 bit. Dengan jelas, ini seharusnya diterapkan untuk Alpha saat ini. Jadi, kami menambahkan sebuah makro %ifarch di sekitar *patch* 64 bit seperti ini :

```
%ifarch axp
%patch1 -p1
%endif
```

- Hal tersebut akan memastikan bahwa *patch* itu tidak diterapkan untuk semua arsitektur, kecuali alpha.

7.4 Membuat perkecualian Arsitektur dari Paket

Anda dapat memelihara *source* RPM dalam satu direktori untuk semua paltform, kami telah mengimplementasikan kemampuan untuk "mengecualikan" paket dari proses pembangunan dalam arsitektur tertentu. Dengan ini anda masih dapat mengerjakan sesuatu seperti

```
rpm --rebuild /usr/src/SRPMS/*.rpm
```

dan mendapatkan paket yang benar sudah dibangun. Bila anda belum membuat porting sebuah aplikasi dalam *platform* tertentu, semua yang harus anda lakukan adalah menambahkan sebuah baris, seperti :

```
ExcludeArch: axp
```

ke dalam kepala dari file spesifikasi dari paket *source*. Kemudian bangun ulang paketnya di atas *platform* dimana ia telah dibangun di atasnya. Anda akan mempunyai paket *source* yang dibangun di atas Intel dan dapat dengan mudah berpindah ke atas Alpha.

7.5 Penyelesaian Akhir

Menggunakan RPM untuk membuat paket multi-arsitektur pada umumnya lebih mudah dikerjakan daripada mendapatkan paket itu sendiri dibangun dalam kedua tempat. Seperti lebih dari paket yang dibangun, ini akan menjadi mudah, bagaimana pun juga. Sebagaimana lazimnya, bantuan yang terbaik saat anda mulai bingung dalam membangun sebuah RPM adalah melihat paket *source* yang mirip.

8 Pemberitahuan Hak Cipta

Dokumen ini dan isinya dilindungi dengan undang-undang Hak Cipta. Distribusi ulang dari dokumen ini diizinkan sejauh isinya tetap lengkap dan tidak diganti. Dengan kata lain, anda boleh memformat ulang dan mencetak ulang atau mendistribusikan ulang saja.