# RPM HOWTO (RPM at Idle)

作者: Donnie Barnes, djb@redhat.com
譯者: 胡崇偉 marr@cc.nccu.edu.tw

v2.0, 8 April 1997

如何使用 Red Hat 包裝管理程式。

## 目錄

# 1 Introduction

RPM 指的是 **R**ed Hat **P**ackage **M**anager 之縮寫， 雖然名稱上包含著 Red Hat 的字號， 但其原始設計理念是採開收式的， 用意在於讓這個程式能提供給所有人使用。 透過 rpm 的管理， 使用者可以將某個新軟體的 source code， 重新整理包裝成另一種 source 與 binary 的檔案型式， 如此一來， 我們可以輕鬆地由 binary 型式的檔案， 進行安裝與套件追蹤管理的工作， 而 source 型式的檔案， 也很方便地能夠再被重新整理包裝。 同時， rpm 管理著一份資料庫， 裡頭包含了所有的程式套件的檔案資料， 透過這份資料庫， 我們可以進行程式套件之確認與查詢工作。

Red Hat 公司鼓勵其他協力廠商， 一同發展及使用 RPM 於它們的 distribution 當中。 RPM 相當具有彈性， 其操作非常簡單， 提供系統擴充發展的基礎， 另一方面， 它的發展完全公開， 而且可免費取得， 使用者可以回覆任何程式碼上的錯誤與修正內容。 只要在符合 GPL 的宣告條件下， 大家可以自由地使用及傳播 RPM 程式。

如果您想獲得有關 RPM 更詳細的文件説明， 可以參考 Ed Bailey 所撰之 *Maximum RPM* 一書。 這本書可由 *www.redhat.com* `<http://www.redhat.com>` 下載或購買到。

# 2 Overview

首先， 讓我們先來了解 RPM 背後的設計哲學， 其設計的目的之一， 便是允許使用者能夠使用「純淨」的 source 檔案。 關於此點， 我們必須回溯至 RPP 程式， (這是一個早期所使用的程式套件管理系統， 不過， 後來 RPM 倒沒有由其衍生)， 它所採用的 source 檔案， 都是經過額外的「破解處理」。 理論上， 使用者可以順暢無誤地， 從一份 RPP 原始檔， 完成 install 與 `make` 的動作。 但是這樣處理過的 source 檔案， 已是面貌全非， 使用者可能無從得知， 檔案裡到底哪些地方被更改過， 通常必須額外取回未經處理過的 source 檔案。 而 RPM 的做法， 便是讓使用者取得「未經處理過的 source 檔案」， 同時再附上一份「patch 檔案」， 可藉之完成程式編譯的工作。 我們很容易發現， 這樣的做法帶來不少好處。 怎麼説呢？ 舉個狀況爲例， 在 Red Hat Linux 下， 如果某個新版的程式問世了， 您可能沒有必要再重頭開始做 compile 動作， 先觀察「patch 檔案」的內容， 看看是否有哪些部份才是您需要做的。 如此一來， 所有內含的程式預設值便很容易看出來。

另一方面， RPM 也具有強大的查詢功能， 您可以針對整個程式套件的資料庫， 或是某些特定的檔案， 進行查詢的動作， 也可以輕鬆地查出， 某個檔案是屬於哪個程式套件， 或是它打從哪裡來的。 RPM 檔案本身是經過壓縮的， 但您還是可以很容易而快速地， 查詢每個程式套件的內容， 因爲在程

式套件裡， 已經加入一段特殊的 binary header， 它記錄了全部查詢時所需的內容， 這樣的做法， 使得查詢功能變得相當快速。

另一項強大的功能， 便是程式套件的完整確認。 如果您擔心誤砍某個檔案， 而它正是某個程式套件的重要部份， 那麼做個「確認動作」就行了。 如有任何不正常的地方， 便會通知您， 這時候， 您可以視情況重新安裝該程式套件， 而且所有之前的設定檔， 都會完好地被保存下來。

我們非常感謝來自 BOGUS distribution 的高手們， 他們提供了許多意見構想， 後來也都被採用於 RPM 當中。 雖然 RPM 完全是由 Red Hat Software 所撰寫， 但其操作模式是根據 BOGUS (PM 與 PMS) 所寫的程式碼。

# 3 General Information

## 3.1 Acquiring RPM

最好且最簡單的方法， 當然就是安裝 Red Hat Linux 囉， 如果您抵死不從， 那麼依然可以從 FTP 站， 抓取原始程式回來安裝。 程式檔案的位置是 *ftp.redhat.com* `<ftp://ftp.redhat.com/pub/redhat/code/rpm>` 。

## 3.2 RPM Requirements

順利執行 RPM 的最大條件， 就是「要有 2.4.2 版以上的 cpio 程式」。 RPM 主要是設計給 Linux 使用的， 但對於其他 Unix 系統的相容性也相當不錯， 事實上， 它已經順利在 SunOS、 Solaris、 AIX、 Irix、 AmigaOS 及其他系統上完成 compile 動作。 值得注意的是， 由不同 Unix 作業平台所編譯出來的 binary 程式套件， 是不能彼此相容的。

上述只是安裝 RPM 的最低需求， 如果想要由 source 檔案建造起， 那麼您還需要 `gcc`， `make` 之類的編譯工具程式， 這和建造一般程式套件並無二致。

# 4 Using RPM

RPM 可以用於安裝程式套件上， 最簡單的操作格式如下：

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

另一個簡單的操作範例， 是用於解除某一個程式套件的安裝：

```
rpm -e foobar
```

接下來的指令範例較為複雜， 但它非常有用， 可以讓您透過 FTP 來安裝程式套件。 如果您能夠連上網路， 想安裝某個新的程式套件時， 所需下達的指令， 便是在檔案前加上適當的 URL 格式， 例如：

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

請特別注意， RPM 目前已允許透過 FTP 來進行查詢或安裝。

上述都是些簡單的指令， rpm 還有許多不同的選項方式， 這些可由 `Usage` 的說明中看出：

```
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}
```

您可以在 RPM 的 man page 裡，找到這些選項的詳細說明。

# 5 Now what can I *really* do with RPM?

RPM 是個非常有用的工具，而且如您所知的，它具有許多選項功能。讓您了解這些功能的最好方式，便是實際介紹幾個範例，前面我們已經提過安裝與解除安裝的簡單例子，底下是一些額外的範例：

- 假設說，您不小心誤刪了幾個檔案，但不確定到底是哪些檔案，您想對整個系統進行確認動作，以了解哪些部份可能已經遺損，您可以這樣做：

  ```
  rpm -Va
  ```

- 假設說，您碰到一個認不出來的檔案，想要知道它是屬於哪一個程式套件的話，您可以這樣做：

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

  其結果會得到：

  ```
  xjewel-1.6-1
  ```

- 如果您找到一個新的 RPM 檔案，卻不清楚它的內容，想要獲得相關的資訊，可以這樣做：

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

  其結果會得到：

  ```
  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.
  ```

- 如果您想了解某個 RPM 檔案，會在系統裡安裝哪些檔案，您可以這樣做：

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

  其結果會是：

  ```
  /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
```

上述只是一些範例， 當您對 RPM 愈加熟悉時， 應該能想出其他更多的創意組合。

# 6 Building RPMs

如果您可以自個兒取得所需的軟體，那麼建造 RPM 檔案也是非常簡單的。

建造 RPM 檔案的基本步驟如下：

- 確定您的 /etc/rpmrc 已經符合系統所需， 完成設定。
- 取回所要建造的 source code。
- 進行所需的 source 檔案 patch 動作， 以使得我們能順利地 build 整個程式。
- 爲程式套件撰寫一份 spec 檔案。
- 確認每個檔案都在正確的目錄位置。
- 使用 RPM 來 build 整個程式套件。

如果一切操作正確， RPM 便能順利 build 完成 binary 與 source 程式套件。

## 6.1 The rpmrc File

目前爲止， RPM 系統唯一的設定檔， 是透過 /etc/rpmrc 檔案來管理。 其內容範例如下：

```
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
```

檔案中的 require_vendor 這一行敘述， 用以控制 RPM 是否須要找尋 vendor 那一行敘述， 而 verdor 的資訊可能來自 /etc/rpmrc 或是 spec 檔案的 header 處。 如果您把上述的號碼改爲 0， 便能把這項尋找功能關閉。 這樣的設定方式， 同樣適用於 require_distribution 與 require_group 的敘述上。

接下來， 我們看到 distribution 這一行， 您可以在此設定， 或是日後在 spec 檔案的 header 處設定。 當我們在某個 distribution 上 build 程式套件時， 就算不需要查詢設定， 此行內容的設定正確， 也是能夠帶來許多便利。 vendor 那一行的作用， 和上述的 distribution 非常相似， 但其內容並不限定 ( 例如是 Joe's Software 或 Rock Music Emporium )。

RPM 目前支援「多平台架構」的程式套件 build 功能， 我們可以在 rpmrc 檔案裡指定 "optflags" 變數， 當進行程式套件 build 動作時， 便可依據所需的平台類型， 應用特定的變數內容。 我們將會在接下去的章節裡， 說明如何使用這些變數。

除了上述的 macro 設定外， 還有許多其他的設定方式， 您可以使用：

```
rpm --showrc
```

來查看系統的 tag 與可供使用的 flag 有哪些。

## 6.2 The Spec File

在此我們將討論 spec 檔案的設定。 build 一個程式套件時， 我們需要使用到 spec 檔案， 其內容爲該程式套件的說明， 額外還包括一些指令， 用以指示整個 build 的過程， 還有一份檔案列表， 用以表示程式套件中的檔案， 分別被安裝到哪裡。

spec 檔案的命名方式， 最好是遵循標準的慣例， 其格式應該爲 package name-dash-version number-dash-release number-dot-spec。

這裡我們舉一個小型的 spec 檔案爲例 (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 The Header

檔案 header 的部份，有幾個特定的欄位內容，您必須加以設定完成，另外還有幾點注意事項。您必須設定完成的欄位內容如下：

- `Summary:` 以一行長度的描述，來說明程式套件的內容。
- `Name:` 這個檔案名稱必須與您準備使用的 rpm 檔名一致。
- `Version:` 這個版本名稱必須與您準備使用的 rpm 檔名一致。
- `Release:` 這個發行序號必須與您準備使用的 rpm 檔名一致。(也就是說，如果我們完成一個程式套件，但事後發現它有些小小的問題，必須重新 build 一次，此時新的程式套件，其發行序號便是 2 號)。
- `Icon:` 如果您使用了其他高階的安裝工具程式 (像是 Red Hat 的 “glint” 程式)，那麼這裡可以指定其相對應的圖示檔，它必須是一個 gif 檔案，而且必須位於 SOURCES 目錄。
- `Source:` 這裡指定了那些「未經處理過的 source 檔案」的 HOME 目錄，當您想要重新取得 source 檔案，或是檢察是否有新版本時，就需要用到這個設定。注意事項：這裡的檔名，「務必」與您系統上的檔案名稱一致 (也就是說，下載 source 檔案後，不要去變更其檔案名稱)。同時，您可以指定一個以上的 source 檔案，方式如下：

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

  這些檔案都會到 `SOURCES` 目錄底下 (相關的目錄結構說明，會在後面的 "The Source Directory Tree" 章節裡加以討論)。

- `Patch:` 如果您需要重新下載 patch 檔案，那麼其目錄位置設定於此。注意事項：這裡的檔名，必須與您系統上使用的 patch 檔案名稱相符，另一方面，和前述多個 source 檔案的設定一樣，您也可以指定多個 patch 檔案名稱。其格式範例如下：

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

  這些檔案都會到 `SOURCES` 目錄底下。

- `Copyright:` 這裡的設定，用以說明程式套件採用何種版權聲明。像 GPL、BSD、MIT、public domain、distributable、commercial 等，都是您可以指定的。

- BuildRoot: 您可以指定一個目錄，它會被當作是 build 與 install 新程式套件的 "root" 目錄，如果您想在實際安裝之前，先行測試程式內容，可以善用此項設定。
- Group: 這裡的設定，用於高階的安裝工具程式（例如 Red Hat 的 "glint" 程式）當中，用以說明程式所屬的群組位置。目前的群組架構，大致如下所述：

```
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` 這個並非真的是 header 項目，但您應該連同上述的項目一同填寫，每個「程式套件」或「子程式套件」，都應該有一個 description tag。這裡允許您輸入多行內容，使得程式套件能夠具有一份完整詳盡的說明。

## 6.4 Prep

這裡是 spec 檔案的另一個段落章節，用以設定讓 source 檔案就緒，以供下一步的 build 動作。平常我們必須經過 setup，才能實際進行 `make` 動作，因此在本段落章節中，我們將視需要進行 source 檔案的 patch 與 setup。

有件事值得注意的：接下來的段落設定，實際上只是指明某段 shell scripts 的位置，您可以將 shell scripts 的內容，另外以 `sh` script 的方式加以存檔，並將 script 程式名稱置於 `%prep` tag 之後，用以執行 source 檔案的 unpack 與 patch 動作。當然，以原有之 macro 型式來做，應該是方便許多的。

第一個要說明的 macro 是 `%setup`。如果我們採用其最簡單的格式（即不加任何命令列參數的情況），它會單純地將 source 檔案加以 unpack，並 `cd` 進入 source 檔案的目錄。除此之外，您還可以使用下列的選項：

- `-n name` 這個選項設定會把 build 的目錄設為 name，原預設值為 `$NAME-$VERSION`，其他可能目錄名稱包括 `$NAME`，`${NAME}${VERSION}`，或是 tar 檔案本身所用的目錄。（請注意到，上述的 “$” 變數，並不是 spec 檔案裡的變數，它們在這裡只是用來代表一個範例名稱，您必須在程式套件裡，使用實際的檔案名稱與版本名稱）。
- `-c` 在進行 untar 動作之前，會先建立一個目錄，並 cd 進入該目錄。
- `-b #` 在 cd 進入該目錄之前，會先將 Source# 進行 untar 動作（這個選項與 `-c` 選項並存時，是沒有意義的，所以應避免同時使用），而且這個選項，僅適用於多個 source 檔案的場合。
- `-a #` 在 cd 進入該目錄之後，再將 Source# 進行 untar 動作。
- `-T` 這個選項會蓋過原本預設的 untar 動作，同時需要一個 `-b 0` 或是 `-a 0` 的選項來配合，當您使用到兩個以上的 source 檔案時，便需要此項功能。
- `-D` 在解開程式套件之前，不要刪除該目錄。這個選項僅適用於使用兩個以上 setup macro 之場合，而且只能用於第一個 setup macro 之後（千萬別用於第一個之內）。

接下來要說明的 macro 是 `%patch`。它是用來協助自動處理 source 檔案更新的動作，其相關的選項很多，列表說明如下：

- `#` 會引用 Patch# 當作其所需之更新檔。

- -p # 有時 patch(1) 指令需配合指定 strip 的目錄數目， 你可以在此選項中加以設定。
- -P 其預設之動作是引用 Patch ( 或 Patch0 )， 當您使用兩個以上的 %patch macro， 並且需要與第一個 macro 不同之 patch number 時， 這個選項顯得相當有用。 它會蓋過原本的預設動作， 並且需要配合一個 0， 以使得主要的 source 檔案得以進行 untar 動作。
- 您也可以使用 %patch#， 而不必使用這樣的指令： %patch # -P

這些應該就是全部所需要知道之 macro 說明， 了解它們之後， 您也可以透過 sh 之 script 格式， 設定不同的 setup 方法， 在 %build macro ( 在下一章節中會提及 ) 之前， 您所設定之所有選項， 都是經由 sh 來執行， 您可以再參考一下前述的範例， 或許可以適用於您的需要。

## 6.5 Build

這個章節裡， 所述及的並非眞正的 macro。 當您把 source 檔案 untar， 並且 patch 完成， cd 進入目錄之後， 開始準備 build 動作時， 便是在這裡設定那些控制 build 動作的指令。 而這些指令都還是傳給 sh， 所以任何 sh 之指令， 都可以在此指定 ( 包括 comments 在內 )。 在 **spec** 檔案裡的每一段落章節設定中， 您的目前所在目錄位置， 都會被重新設定爲 **source directory** 的最上層， 所以請牢記在心， 必要時， 您可以 cd 進入相關的子目錄。

## 6.6 Install

這裡所設定的， 同樣也不是 macro， 基本上， 您只須要在此設定一些 install 所需之指令。 如果您打算在程式套件裡， 提供完整的 make install 指令設定， 那麼請在本段落設定中完成， 不然， 您也可以更改 makefile 檔案裡， 有關 make install 的部份， 然後僅在本段落設定中指定 make install。 或是， 也可以把整個 install 的指令交給 sh 來做。 記住， 您的目前所在目錄位置， 應該已被重新設定爲 source directory 的最上層。

## 6.7 Optional pre and post Install/Uninstall Scripts

程式套件在安裝與解除安裝之前後，您可以指定 script， 使其視情況加以執行。 進行此項動作的主要原因之一， 便是遇到如下的場合， 譬如說， 我們在安裝或解除安裝一些含有 shared library 的程式套件時， 需要執行 ldconfig。 各式 script 所需之 macro 名稱如下：

- %pre 執行 pre-install scripts 的 macro。
- %post 執行 post-install scripts 的 macro。
- %preun 執行 pre-uninstall scripts 的 macro。
- %postun 執行 post-uninstall scripts 的 macro。

這些段落設定的內容， 可以是任何 sh 型式之 script， 不過， 您無須指定關鍵敘述 #!/bin/sh。

## 6.8 Files

本段落設定當中， 您必須列出程式套件內之所屬檔案名稱。 RPM 本身並無從得知， 執行 make install 之後， 到底有哪些 binary 檔案被安裝進去， 目前並無他法可以直接解決此問題。 有些人建議在 install 程式套件的前後， 使用 find 指令來處理， 不過在一個多使用者的系統下， 這應該是不可行的， 因為在程式套件 build 的過程中， 可能有其他與程式套件本身無關的檔案被產生。

另外還有一些 macro， 它們可用來做一些特殊的工作， 茲將其列述於下：

- %doc 用以標示在 install 程式套件時， 您所想要安裝之 source 檔案裡的說明文件是哪些， 而這些說明文件會被安裝在 /usr/doc/$NAME-$VERSION-$RELEASE。 您可以在命令列上， 以此 macro 同時指定好幾個說明文件名稱， 或是以此 macro， 各別為它們一一完成指定。
- %config 用以標示程式套件裡的設定檔案是哪些， 這包括像 sendmail.cf、 passwd 之類的檔案。 如果您事後 uninstall 了某個程式套件， 而且它含有設定檔案， 那麼所有沒有改變的檔案會被移除， 而所有已經改變的檔案， 會在原檔案名稱之後， 加上 .rpmsave 的延伸名稱。 同樣的， 您可以在此同時列出多個檔案名稱。
- %dir 在檔案列表中， 標明某一個特定的目錄， 用以說明該目錄為某程式套件所擁有。 如果您在檔案列表中指定一個目錄名稱， 但卻沒有在前面加上%dir macro， 那麼該目錄內， 所有檔案目錄都會被包含在檔案列表當中， 並在稍後， 被當作是程式套件的一部份而全被安裝進去。
- %files -f <filename> 允許您引用 source 之 build 目錄裡， 某份檔案清單的內容。 當您遇到某一個程式套件， 它可以建立自己的檔案清單時， 這個選項便顯得相當不錯， 您只須要引用該份檔案清單， 不必再自行辛苦地列出。

在檔案列表裡， 有個最大的注意事項， 便是目錄的設定。 如果您不小心將 /usr/bin 列入， 那麼您的程式套件， 將會包括系統裡 /usr/bin 底下的所有檔案。

## 6.9 Building It

### 6.9.1 The Source Directory Tree

第一件事， 您必須選定一個適當的 build tree， 此項設定可在 /etc/rpmrc 檔案裡完成， 而大多數人會直接使用 /usr/src。

您可能還需要建立下列的目錄， 使得 build tree 的設定能夠完成：

- BUILD 此目錄便是 RPM 進行所有 build 動作的工作目錄。 您不必特別地在哪個目錄進行測試性的 build 動作。
- SOURCES 此目錄存放著您的「起始」 (original) 原始 tar 檔案、 與相關的 patch 檔案， RPM 預設會尋找本目錄。
- SPECS 此目錄存放所有之 spec 檔案。
- RPMS 此目錄存放所有 build 產生之 binary 格式 RPM 檔案。
- SRPMS 此目錄存放所有 source 格式 RPM 檔案。

### 6.9.2 Test Building

首先， 您大概會想要取回 source 檔案， 在沒有使用 RPM 的情況下， 進行一次「純淨的」 build 動作。 其步驟便是， 解開 source 檔案， 將該目錄名稱改為 $NAME.org， 然後再次解開 source 檔案， 我們需要使用此一 source 來進行 build 動作。 進入此一 source 目錄， 按照指示來進行 build， 如果您必須編輯任何東西， 您會需要一份 patch， 一旦您完成 build 工作， 便可清除 source 目錄裡的內容。 請確定將 `configure` script 裡， 所有產生之檔案加以清除， 然後再 `cd` 回到 source 目錄之上層， 接著便可執行這樣的動作：

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

上述指令會產生一份 patch， 您在 spec 檔案可以使用到它。 注意到上面的 "linux" 名稱， 它只是一個提示作用， 或許您可以使用其他名稱， 諸如 "config" 或 "bugs" 之類的提示名稱， 用以說明您何以製作此一 patch 檔案。 同時， 您最好也在使用 patch 檔案之前， 先觀察裡頭的內容， 確定是否無意間包含了其他的 binary 檔案。

### 6.9.3 Generating the File List

現在， 您已經有了一份可以拿來 build 的 source 檔案， 而且您也知道如何完成其相關的動作， 如 build 與 install 等。 觀察 install 時， 依序產生的結果， 我們將由此結果， 在 spec 檔案中建立一份檔案列表。 通常我們會在進行上述步驟的同時， 一起建立 spec 檔案， 您可以先完成檔案的起始部分， 和幾個簡單的部分， 然後再把其他部分的步驟加以完成。

### 6.9.4 Building the Package with RPM

一旦您有了一份 spec 檔案， 一切便已就緒， 您可以準備 build 的測試動作。 最好的方式， 就是使用類似下列的指令：

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

配合 `-b` 選項， 我們還可以使用其他有用的選項：

- `p` 指的是只執行 spec 檔案之 `prep` 部份。
- `l` 針對 `%files` 進行檔案列表的比對檢查動作。
- `c` 執行 prep 與 compile 的動作。 當您不確定 source 檔案是否能夠 build 時， 此選項顯得相當有用。 或許您會覺得它並沒有多大用處， 因為您可能想要繼續修改 source 檔案本身， 直到 build 後再開始使用 RPM， 但一旦您熟悉 RPM 的用法後， 會發現它的便利之處。
- `i` 執行 prep、 compile、 與 install。
- `b` 執行 prep、 compile、 install、 並只 build 出一份 binary 程式套件。
- `a` 執行所有的 build 動作 ( 包括 source 與 binary 的程式套件 )。

配合 `-b` 選項， 另外還有一些細項選項可供使用， 它們分別是：

- `-short-circuit` 會直接跳至某個特定的階段 ( 僅可配合 c 與 i 選項來使用 )。

- -clean 當執行完畢時，移除相關的 build tree。
- -keep-temps 會把所有的 temp 與 scripts 檔案，都保留在 /tmp 目錄，您可以使用 -v 選項，實際觀察有哪些檔案被產生。
- -test 並不實際執行任何階段的動作，但有執行 keep-temp 的部份。

### 6.10 Testing It

一旦有了 source 與 binary 的 rpm 檔案，您必須進行測試工作。最簡單且最好的方式，就是使用另一台機器來測試，也就是進行 build 動作之外的機器。畢竟，您剛在您的機器上，完成一大堆的 make install 動作，若在原機器上做測試，當然會顯得相當順利囉。

您可以執行 rpm -u packagename 來進行測試，但這樣做還是有造假的可能，因爲在 build 的過程中，您做了 make install 的動作，如果您在檔案列表中遺漏了某些東西，那麼它不會被解除安裝，然後您再 reinstall 這份 binary 程式套件，便會發現整個系統還是完整而運作正常的，但實際上的 rpm 檔案還是有問題。因此請特別記住，由於您執行的是 rpm -ba package，而大多數的人，會只以 rpm -i package 方式，來安裝您的程式套件。當 binary 檔案獨自被安裝時，您必須確定在 build 或 install 的段落設定中，並沒有相關的部份對其有影響。

### 6.11 What to do with your new RPMs

一旦您作出了一份自己的 RPM 檔案（假定這份檔案，之前並未以 RPM 方式製作過），您可以將您的作品貢獻給別人（此時假定您製作的 RPM 檔案，是可以自由傳佈的）。您可以考慮把它上傳至 *ftp.redhat.com* <ftp://ftp.redhat.com>。

### 6.12 What Now?

請回顧上述的章節，在「Testing」和「What to do with new RPMs」裡，我們希望所有的 RPM 檔案都能被提供出來，而且我們希望它們都會是好的 RPM 檔案。因此，請多花一點時間好好地測試它們，然後再花點時間將它們上傳，以造福普羅大眾。同時，請確定您只上傳可供自由傳佈的軟體。商業軟體與共享軟體是不應該被上傳的，除非它們有份許可聲明在上面。這樣的軟體，包括有 Netscape software、ssh、pgp 等。

## 7 Multi-architectural RPM Building

RPM 目前可用於 build 各式作業平台之程式套件，如 Intel i386、安裝 Linux 之 Digital Alpha、與 Sparc 機器，而且有人測試報告，說明它也可以在 SGI 與 HP 工作站上使用。RPM 有許多特色，使得它能夠輕鬆地支援在各式作業平台上 build 程式套件，第一個特色便是在 /etc/rpmrc 檔案裡的 “optflags” 設定項目，當我們在 build 軟體套件時，它可用於指定平台架構的相關設定值。另一個特色是 spec 檔案裡的 “arch” macro 項目，根據所要進行 build 的平台架構，您可以指定它們執行不同的工作。另一個特色是在檔案 header 處的 “Exclude” 設定項目。

## 7.1 Sample spec File

下列以 “fileutils” 程式套件爲例， 其 spec 檔案的部份內容， 可用來在 Alpha 與 Intel 平台架構上， 進行 build 的工作。

```
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

在此一範例中， 您可以發現 “optflags” 設定項目， 是如何由 `/etc/rpmrc` 檔案中發揮作用。 首先決定您所進行 build 動作的機器， 是哪一種平台架構， 請將其正確之設定值， 指定到 `RPM_OPT_FLAGS` 。 您必須針對程式套件之 Makefile 檔案， 進行 patch 的動作， 將一些通用的變數設定值， 加以改變以適用於您的環境 ( 如 `-m486` 與 `-O2` )。 您可以 install 這份 source 套件， 接著解開 source 並檢查其 Makefile 檔案， 以進一步找出更好的設定方式， 然後觀察 Makefile 的 patch 內容， 以了解我們必須做哪些改變。

### 7.3 Macros

`%ifarch` 此一 macro 設定項目是非常重要的， 在許多場合， 您須要只針對某一特定平台架構， 進行一兩個 patch 工作， 這種情況下， RPM 能夠讓您僅針對某一平台架構， 採用必要的 patch。

在上述範例中， fileutils 有個提供給 64 bit 機器使用的 patch， 很明顯地， 這項 patch 僅需要由 Alpha 機器所採用， 因此， 我們針對 64 bit 的 patch 部份， 增加一個 `%ifarch` 的 macro 設定項目， 其範例如下：

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

這樣能保證， 除了 Alpha 機器之外， 其他平台架構的機器並不會採用此 patch。

### 7.4 Excluding Architectures from Packages

您可以在一個目錄內， 同時管理各式作業平台的 RPM 原始檔案， 這項功能使得我們在製作程式套件的時候， 不必限定於某些作業平台上。 因此您可以這樣下指令：

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

使得正確的程式套件能夠被 build。 如果您尚未移植某個應用程式至某一特定的作業平台， 可以加上這樣的一行內容：

```
ExcludeArch: axp
```

到 source 程式套件當中， spec 檔案的 header 處， 然後在原本進行 build 的作業平台上， 再做一次 rebuild 動作。 此時， 您就有了一份在 Intel 機器上 build 完成的 source 程式套件， 同時它也很容易便能移植至 Alpha 機器上。

### 7.5 Finishing Up

使用 RPM 來製作「多作業平台」的程式套件， 通常會比各別在該平台製作來得容易， 而且隨著您製作越多的程式套件， 其效應會更加明顯， 因爲， 我們可以參考其他類似的原始程式套件內容， 得以處理製作時遇到的問題。

## 8 Copyright Notice