# RPM HOWTO

Donnie Barnes, djb@redhat.com, перевод Alex Ott ott@phtd.tpu.edu.ru v2.0, 8 апреля 1997

## Содержание



**Примечание переводчика:** Шлите мне любый комментарии и замечания, даже небольшие.

## 1 Введение

RPM это **R**ed Hat **P**ackage **M**anager (Менеджер пакетов RedHat). Хотя он содержит Red Hat в своем имени, он полностью предназначен работать как открытая пакетная система доступная для использования кем угодно. Она позволяет пользователям брать исходный код для нового программного обеспечения и упаковывать его в форме исходного и двоичного кода, так что двоичные файлы могут быть легко установлены и отслежены, а исходный код легко построен. Эта система также сопровождает базу данных всех пакетов и их файлов, что может быть использовано для проверки пакетов и запроса информации о файлах и/или пакетах.

Red Hat Software поощряет создателей других дистрибутивов чтобы они взглянули на RPM и использовали его для своих собственных дистрибутивов. RPM является довольно гибкой системой и легкой в использовании, хотя он обеспечивает основу для очень сложных систем. Он также полностью открыт и доступен, и мы будем увеличивать список найденных ошибок и исправлений. Разрешение дано на свободное использование и распространение RPM без оплаты под действием GPL.

Более полная документация о RPM доступна в книге Ed Bailey, *Maximum RPM*. Эта книга доступна для скачивания или покупки с *www.redhat.com* <http://www.redhat.com>.

## 2 Обзор

Первым делом разрешите мне изложить философию RPM. Одна из целей проектирования была в том чтобы позволить использовать "безупречные" исходные тексты. В RPP (нашей предыдущей системе пакетов от которой RPM не унаследовала *ничего*) исходные тексты были "исправлены", чтобы мы могли скомпилировать их. Теоретически кто-либо должен был установить исходные тексты с помощью RPP и собрать их без проблем. Но исходные тексты были не оригинальными и не

было ссылок на то какие изменения мы сделали, чтобы заставить их работать. Некоторые люди загружали безупречные исходные тексты раздельно. В RPM мы имеем безупречные исходные тексты вместе с заплатками, которые используются для компиляции. Мы рассматриваем это как большое достижение. Для некоторых людей. если пришла новая версия программы, вам не нужно начинать с самого начала чтобы заставить ее скомпилироваться под RHL. Вы можете посмотреть на заплатку (patch), чтобы увидеть что вам необходимо сделать. Все значения по умолчанию для компиляции легко видны при этом способе.
RPM также спроектирован, чтобы иметь мощную систему запроса. Вы можете искать во всей вашей базе данных информацию о пакете или о просто определенных файлах. Вы также можете легко найти какому пакету принадлежит файл и и откуда появился. Сами файлы RPM являются сжатыми архивами, но вы можете запрашивать отдельный пакет очень легко и быстро, потому-что к пакету добавлен дополнительный двоичный заголовок со всем что вам может быть необходимо знать в несжатой форме. Это позволяет производить *быстрые* запросы.
Другое мощное свойство — это проверка пакетов. Если вы беспокоитесь о том, что вы удалили важный файл из некоторого пакета, просто проверьте это. Вы будете оповещены об любых аномалиях. С этой точки вы можете переустановить пакет, если это необходимо. Любые конфигурационные файлы будут сохранены.
Мы хотим отблагодарить людей из дистрибутивной группы BOGUS за множество их идей и концепций, которые включены в RPM. Хотя RPM был полностью написан Red Hat Software, его операции основаны на коде, написанном BOGUS (PM и PMS).

# 3 Основная информация

## 3.1 Получение RPM

Лучший способ получить RPM это установить Red Hat Linux. Если вы не хотите делать это, то вы все равно сможете получить и использовать RPM. Он мажет быть получен с *ftp.redhat.com* <ftp://ftp.redhat.com/pub/redhat/code/rpm>.

## 3.2 Требования RPM

Основное требование для запуска RPM это наличие cpio 2.4.2 или старше. Хотя эта система предназначена для использования в Linux, она может быть спокойно перенесена на другие Unix-системы. Она была скомпилирована на SunOS, Solaris, AIX, Irix, AmigaOS, и других. Будьте предупреждены, двоичные пакеты, сгенерированные на разных типах систем Unix не являются совместимыми.
Это минимальные требования для установки RPM. Для построения RPM из исходных текстов, вам также необходимо все обычно требуемое для построения пакета, подобно `gcc`, `make`, и т.п.

# 4 Использование RPM

В простейшей форме RPM может быть использован для установки пакетов:

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

Следующая простая команда используется для удаления пакета:

```
rpm -e foobar
```

Одна из более сложных, но *очень* полезных команд позволяет вам устанавливать пакеты через FTP. Если вы подключены к сети и хотите установить новый пакет, все что вам нужно — это указать файл с правильным URL, примерно так:

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

Заметим, что RPM запросит и/или установит через FTP.
Хотя это простые команды, RPM может быть использован множеством способов, как это видно из сообщения об использовании (`Usage`):

```
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}
```

Вы можете найти больше информации о том что какая опция обозначает на справочной странице RPM.

# 5 Что я могу *по-настоящему* делать с RPM?

RPM это очень полезная утилита, и как вы видите имеет различные опции. Лучший способ прочувствовать их это посмотреть на несколько примеров. Я привел простые примеры установки/удаления выше, так что здесь будет несколько больше примеров:

- Допустим вы случайно удалили некоторые файлы, но не уверены в том что удалили. Если вы хотите проверить всю систему и просмотреть что может отсутствовать, вы должны сделать:

```
rpm -Va
```

- Допустим вы нашли файл, который вы не можете опознать. Для того чтобы найти к какому пакету он относится, вы должны сделать:

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

  Вывод должен быть:

```
xjewel-1.6-1
```

- Вы нашли новый пакет RPM, но не знаете для чего он. Для того чтобы найти некоторую информацию о нем, сделайте:

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

  Вывод должен быть:

```
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.
```

- Теперь вы хотите посмотреть какие файлы установит этот пакет. Вы должны сделать:

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

  Вывод будет:

```
/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
```

Это только несколько примеров. Более творческие примеры могут придуманы легко, если вы подружитесь с RPM.

# 6 Построение пакетов RPM

Построить пакеты RPM довольно легко, особенно если вы можете получить программное обеспечение, которые вы пытаетесь упаковать чтобы построить для себя.
Основные процедуры чтобы построить пакет RPM следующие:

- Убедитесь, что файл `/etc/rpmrc` установлен на вашей системе.
- Получите исходный код из которого вы хотите построить пакет RPM на вашей системе.
- Сделайте заплатки к любым изменениям, которые вы сделали чтобы исходный код построился правильно.
- Создайте spec-файл для пакета.
- Убедитесь, что все на нужном месте.
- Постройте пакет используя RPM.

При нормальных условиях, RPM построит и двоичный пакет и пакет с исходным кодом.

## 6.1 Файл rpmrc

Настройка RPM доступна через файл `/etc/rpmrc`. Пример выглядит подобно:

```
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
```

Строка `require_vendor` заставляет RPM найти строку производителя. Она может быть из файла `/etc/rpmrc` или из заголовка самого spec-файла. Что выключить эту опцию, смените число на `0`. Тоже самое является правдой для строк `require_distribution` и `require_group`.
Следующая строка это строка `distribution`. Вы можете определить ее здесь или позже в заголовке spec-файла. При построении пакета для особого дистрибутива это хорошая идея убедиться что строка правильна, даже хотя она требуется. Строка `vendor` обозначает тоже самое, но может быть чем угодно (например, Joe's Software and Rock Music Emporium).
RPM также сейчас поддержку для построения пакетов для множественных архитектур. Файл `rpmrc` может содержать переменную "optflags" для построения вещей, которые требуют специфических для данной архитектуры флагов для построения. Смотрите следующие разделы для описания как использовать эту переменную.
В добавление к вышеприведенным макросам, существует еще несколько. Вы можете использовать:

```
rpm --showrc
```

чтобы увидеть какие значения установлены у вас и какие флаги доступны.

## 6.2 Spec-файл

Мы начнем с обсуждения spec-файла. Spec-файл требуется для построения пакета. Spec-файл это описание программного обеспечения вместе с инструкциями как построить пакет и списком файлов для всех устанавливаемых файлов.
Вы можете захотеть назвать ваш spec-файл согласно стандартному соглашению. Имя должно быть следующим: имя пакета-тире-номер версии-тире-номер выпуска (релиз)-точка-spec.
Здесь приведен маленький spec-файл (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 Заголовок

Заголовок имеет несколько стандартных полей, которые вам необходимо заполнить. Также существует несколько предостережений. Поля должны быть заполнены как показано:

- `Summary`: Это однострочное описание пакета.
- `Name`: Это должна быт строка имени из имени файла rpm, котрое вы планируете использовать.
- `Version`: Это должна быть строка версии из имени файла rpm, которое вы планируете использовать.
- `Release`: Это номер выпуска для пакета с той же самой версией (например, если мы сделали пакет и обнаружили, что он незначительно неисправный и нам необходимо сделать его заново, то следующий пакет будет номер выпуска 2).
- `Icon`: Это имя файла иконки, которое будет использоваться другими высокоуровневыми утилитами установки (подобными "glint" из Red Hat). Она должна быть в формате gif и располагаться в директории SOURCES.

- `Source:` Эта строка указывает на расположение "ДОМА"файла исходных текстов. Она используется если вы хотите получить исходные тексты снова и проверить новые версии. Предостережение: Имя файла в этой строке ДОЛЖНО соответствовать имени файла который вы имеете на своей собственной системе (например не изменяйте имя загруженного файла исходных текстов). Вы можете также указать больше чем один файл исходных текстов используя следующие строки:

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

  Эти файлы должны находиться в директории `SOURCES`. (Структура директорий обсуждается далее в разделе "Дерево директорий исходных текстов").

- `Patch:` Это место где вы можете найти заплатки, если вы захотите загрузить их снова. Предостережение: Имя файла должно соответствовать имени файла которое вы использовали когда делали вашу заплатку. Вы можете также заметить, что вы можете иметь много файлов заплаток также как вы можете иметь много файлов исходных текстов. У вас должно быть что-то подобное:

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

  Эти файлы должны быть в директории `SOURCES`.

- `Copyright:` Эта строка говорит с какими авторскими правами идет пакет. Вы можете использовать что-то подобное GPL, BSD, MIT, public domain, distributable, или commercial.

- `BuildRoot:` Эта строка позволяет вам указать директорию как “корневую” для построения и установки нового пакета. Вы можете использовать это для тестирования вашего пакета до установки его на вашей машине.

- `Group:` Эта строка используется чтобы указать высокоуровневым программам установки (таким как “glint” Red Hat) где разместить эту отдельную программу в их иерархических структурах. Дерево груп в настоящее время выглядит примерно так:

```
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` В действительности это не часть заголовка, но этот раздел должен быть описан вместе с остальными частями заголовка. Вам нужен один таг описания на один пакет и/или подпакет. Это многостроковое поле, которое должно использоваться чтобы дать достаточно полное описание пакета.

## 6.4 Раздел Prep

Это второй раздел в spec-файле. Он используется чтобы сделать исходные тексты готовыми к построению. Здесь вам необходимо сделать все что угодно чтобы сделать исправления в исходных текстах и сделать настройку подобную той, которую необходимо сделать чтобы выполнить `make`.
Одно замечание: Каждый из этих разделов в действительности просто место для выполнения скриптов оболочки. Вы должны просто сделать `sh`-скрипт и поместить его после тага `%prep` для распаковки и исправления ваших исходных текстов. Однако мы добавили макросы чтобы помочь вам сделать это.
Первый из этих макросов это макрос `%setup`. В своей простейшей форме (без командной строки), он просто распаковывает исходные тексты и делает `cd` в директорию исходных текстов. Он также принимает следующие опции:

- `-n name` установит имя директории где будет производиться построение пакета в `name`. Значение по умолчанию равно `$NAME-$VERSION`. Другие возможные значения включают `$NAME`,

`${NAME}${VERSION}`, или что использует главный файл архива. (Заметим, что эти переменные с "$" *не* являются настоящими переменными доступными внутри spec-файла. Они просто используются здесь вместо имен примеров. Вам необходимо использовать настоящие имена и версии в вашем пакете, а не эти переменные).

- `-c` создаст указанную директорию *go* выполнения распаковки архивов.
- `-b #` будет выполнять распаковку Source*#* *go* выполнения cd в директорию (и это делает нечувствительной к опции `-c` так что не делайте ее). Это полезно только в случае множества файлов исходных текстов.
- `-a #` будет выполнять распаковку Source*#* *после* перехода в директорию.
- `-T` Эта опция отменяет действия по умолчанию при распаковке исходных текстов и требует опций `-b 0` или `-a 0` чтобы произвести разархивацию главного файла исходных текстов. Вам нужно это в случае наличия дополнительных файлов исходных текстов.
- `-D` *Не* удалять директорию до распаковки. Это полезно только когда вы имеете больше одного макроса setup. Эта опция должна использоваться *только* в макросах setup *после* первого (но никогда не быть в первом макросе).

Следующий из имеющихся макросов это макрос `%patch`. Этот макрос помогает автоматизировать процесс наложения заплаток на исходные тексты. Макрос имеет несколько опций, перечисленных ниже:

- `#` будет прикладывать Patch*#* как файл заплатки.
- `-p #` указывает количество отбрасываемых директорий для команды patch(1).
- `-P` Действие по умолчанию — наложение `Patch` (или `Patch0`). Этот флаг запрещает действие по умолчанию и будет требовать `0` чтобы распаковать главный файл с исходными текстами. Эта опция полезна во второй (и последующих) макросах `%patch`, которые требуют номера отличного от номера в первом макросе.
- Вы также можете выполнять `%patch#` вместо выполнения команды: `%patch # -P`

Это все макросы которые вам необходимы. После того как вы все сделаете правильно, вы также можете сделать любую другую настройку, которая необходима, используя скрипты на `sh`. Все что вы включите до макроса `%build` (обсуждаемого в следующем разделе) выполняется через `sh`. Посмотрите в вышеприведенном разделе для того чтобы увидеть какие вещи вы можете сделать если захотите.

## 6.5 Раздел Build

Для этого раздела нет никаких макросов. Вы должны просто поместить здесь любые команды которые вам необходимо выполнить для построения программного обеспечения после того как вы распаковали исходные тексты, изменили их с помощью заплаток и вошли в директорию. Это просто другой набор команд передаваемых `sh`, так что любые команды `sh` могут быть здесь (включая комментарии) **Ваша текущая директория устанавливается в каждом из этих разделов в корневую директорию для исходных текстов**, так что помните это. Вы можете переходить в поддиректории если это необходимо.

## 6.6 Раздел Install

В этом разделе также нет никаких макросов. Вам просто необходимо поместить команды необходимые для установки. Если для вашего пакета существует команда `make install`, то просто поместите ее здесь. Если ее нет, то вы можете сделать заплатку для makefile, чтобы выполнялась команда `make install` и просто делать здесь `make install`, или вы можете вручную устанавливать пакет с помощью команд `sh`. Вы можете считать свою текущую директорию как корневую директорию для исходных текстов пакета.

## 6.7 Опциональные скрипты выполняемые до и после установки/удаления пакета

Вы можете поместить скрипты которые запустятся до и после инсталляции и деинсталляции двоичного пакета. Основная причина для этого — это выполнение вещей подобных запуску `ldconfig` после установки или удаления пакета, который содержит разделяемые библиотеки. Макросы для каждого из скриптов приведены как показано:

- `%pre` макрос для выполнения предустановочного скрипта.
- `%post` макрос для выполнения послеустановочного скрипта.
- `%preun` макрос для выполнения скрипта перед удалением пакета.
- `%postun` макрос для скрипта выполняемого после удаления пакета.

Содержимым разделов должны быть любые `sh` скрипты, хотя вам *не* нужно определять строку `#!/bin/sh`.

## 6.8 Раздел Files

Это раздел где вы *должны* перечислить файлы для двоичного пакета. У RPM нет способа узнать какие двоичные файлы установлены как результат выполнения `make install`. *НЕ* существует способа сделать это. Некоторые предлагают выполнить команду `find` до и после установки пакета. На многопользовательской системе это неприемлемо так как другие файлы могут быть созданы в течении процесса построения пакета, которые не имеют ничего общего с самим пакетом.
Есть несколько доступных макросов для выполнения специальных действий. Они перечислены и описаны здесь:

- `%doc` используется для обозначения документации в исходных текстах пакета, которую вы хотите установить при установке. Документы будут установлены в директорию `/usr/doc/$NAME-$VERSION-$RELEASE`. Вы можете перечислить много документов в командной строке этого макроса или вы можете перечислить все отдельно, используя этот макрос для каждого документа.
- `%config` используется для обозначения конфигурационных файлов в пакете. Этот список включает файлы подобные sendmail.cf, passwd, и т.п. Если вы позже удаляете пакет содержащий конфигурационные файлы, все неизмененные файлы будут удалены и все измененные будут переименованы со старыми названиями с добавлением `.rpmsave` к имени файла. Вы можете перечислять много файлов в этом макросе.
- `%dir` обозначает единичную директорию в списке файлов включенную как директория которой владеет пакет. По умолчанию, если вы укажете имя директории *БЕЗ* макроса `%dir`, то *ВСЕ* в этой директории будет включено в список файлов и позже установлено как часть пакета.

- `%files -f <filename>` позволит вам перечислить ваши файлы в некотором файле внутри директории построения исходных текстов. Это просто великолепно в случае когда у вас пакет, который может построить свой собственный список файлов. Затем вы просто включаете этот список файлов здесь и вы не должны специально перечислять файлы.

Наибольшое предостережение в списке файлов это перечисление директорий. Если вы случайно укажете `/usr/bin`, то ваш двоичный пакет будет содержать *все* файлы в директории `/usr/bin` на вашей системе.

## 6.9 Построение пакета

### 6.9.1 Дерево директорий исходных текстов

Первая вещь которая вам необходима — это правильно настроенное дерево построения. Это настраивается используя файл `/etc/rpmrc`. Большинство людей просто используют директорию `/usr/src`.
Вам надо создать следующие директории чтобы создать дерево построения:

- `BUILD` это директория где происходят все построения с помощью RPM. Вы не должны проводить ваше тестовое построение где то в особом месте, но это место где RPM будет проводить построение пакета.
- `SOURCES` это директория где вы должны поместить ваши оригинальные архивные файлы и ваши заплатки. Это место где RPM будет искать их по умолчанию.
- `SPECS` это директория где должны находиться все spec-файлы.
- `RPMS` это место где RPM поместит все двоичные пакеты RPM после их построения.
- `SRPMS` место где будут помещены пакеты RPM с исходными текстами.

### 6.9.2 Тестовое построение пакета

Первая вещь которую вы вероятно захотите сделать — это построить исходные тексты без использования RPM. Чтобы сделать это распакуйте исходные тексты и измените имя директории на $NAME.orig. Затем еще раз распакуйте исходные тексты. Используйте эти исходные тексты для построения. Перейдите в директорию исходных текстов и следуйте инструкциям по их построению. Если вы что-то редактировали вам необходимо сделать заплатку. После того как вы построили исходные тексты, очистите директорию исходных текстов. Убедитесь что вы удалили все файлы созданные скриптом `configure`. Затем перейдите из директории исходных текстов в директорию являющуюся для них родительской. Затем сделайте что-то подобное:

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

Это создаст для вас заплатку, которую вы сможете использовать в вашем spec-файле. Заметим что "linux", который вы видите в имени заплатки это просто идентификатор. Вы можете захотеть использовать что-нибудь более описательное как "config" или "bugs" для описания *почему* вы сделали эту заплатку. Также хорошая идея посмотреть в файл заплатки, который вы создали, до его использования чтобы убедиться что бинарные файлы случайно не включены.

### 6.9.3 Создание списка файлов

Сейчас у вас есть исходные тексты, которые будут строиться и вы знаете как построить и установить их. Посмотрите в вывод установочной последовательности и постройте на его основе список ваших файлов для использования в spec-файле. Мы обычно создаем spec-файл параллельно со всеми описанными шагами. Вы можете сначала создать его и заполнить самые легкие его части, а затем затем заполнять его по мере прохождения всех этапов.

### 6.9.4 Построение пакета с помощью RPM

Когда вы имеете spec-файл, вы готовы попытаться и построить ваш пакет. Наиболее полезный способ сделать это — использовать команду похожую на следующую:

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

Также существуют другие опции полезные с переключателем `-b`:

- `p` обозначает просто запуск раздела `prep` spec-файла.
- `l` это проверка списка, который делает некоторые проверки раздела `%files`.
- `c` выполняет раздел prep и компиляцию. Это полезно в случае, когда вы не уверены будут ли ваши исходные тексты построены. Это выглядит бесполезно, потому-что вы можете захотеть просто самому поиграть с исходными текстами до их построения и затем начать использовать RPM, но когда вы привыкнете использовать RPM вы найдете случаи когда этот ключ используется.
- `i` выполняет prep, компиляцию и установку.
- `b` выполняет prep, компиляцию, установку, и построения двоичного пакета.
- `a` строит все (и двоичный пакет и пакет с исходными текстами).

Существует несколько модификаторов к переключателю `-b`. Это:

- `--short-circuit` будет пропускать действия до указанной стадии (может использоваться с ключами c и i).
- `--clean` удаляет дерево построения когда все сделано.
- `--keep-temps` будет сохранять все временные файлы и скрипты которые созданы в /tmp. Вы можете в действительности посмотреть какие файлы созданы в директории /tmp используя опцию `-v`.
- `--test` не выполняет никакую реальную стадию, но делает keep-temp.

## 6.10 Тестирование пакета

После того как вы построили двоичный пакет и пакет с исходным кодом, вам необходимо проверить их. Самый легкий и наилучший способ — это использовать для тестирования использовать другою машину, а не ту на которой вы создавали пакет. После всего вам только лишь надо выполнить несколько команд `make install` на вашей машине, так что все должно быть установлено.
Вы можете выполнить `rpm -u packagename` для тестирования пакета, но это может быть обманывающим потому-что в процессе построения пакета вы делали `make install`. Если вы пропустите что-нибудь в своем списке файлов, это не будет удалено при инсталляции. Если вы затем будете переставлять двоичный пакет, то все на вашей системе будет в норме, но сам пакет не будет полным. Будьте уверенными и помните, что если вы делали `rpm -ba package`, то большинство людей будут устанавливать ваш пакет выполняя лишь команду `rpm -i package`. Будьте уверены, что вы не делаете ничего в разделах `build` или `install`, что должно будет быть сделано при установке только двоичного пакета.

### 6.11 Что делать с вашим новым пакетом RPMs

После того как вы сделали свой собственный пакет RPM из чего-либо (предполагая что этого еще нет в виде RPM) вы можете предлагать свою работу другим людям (также предполагая, что ваш RPM свободно распространяемый). Чтобы сделать это вы можете захотеть загрузить пакет на сервер *ftp.redhat.com* <ftp://ftp.redhat.com>.

### 6.12 Что теперь?

Пожалуйста смотрите выше разделы "Тестирование пакета"и "Что делать с вашим новым пакетом RPM". Мы хотим сделать доступными все пакеты RPM которые мы можем получить и хотим чтобы это были хорошо сделанные пакеты. Пожалуйста найдите время для хорошего тестирования пакетов и найдите время для их загрузки для общей выгоды. Также *пожалуйста* будьте уверены, что вы загружаете только *свободно распространяемое программное обеспечение*. Коммерческое программное обеспечение и shareware *не* должны быть загружены до тех пор пока не будут иметь авторские права, которые определенно констатируют что это разрешено. Это включает программное обеспечение Netscape, ssh, pgp, и т.п.

## 7 Построение RPM для нескольких архитектур

Сейчас RPM может использоваться для построения пакетов для Intel i386, Digital Alpha с работающим Linux и the Sparc. Также было сообщено, что RPM работает на SGI и рабочих станциях HP. Существует несколько свойств, которые делают построение пакетов не всех платформах легким. Первое из этих свойств это директива “optflags" в файле `/etc/rpmrc`. Она может быть использована для установки используемых для построения программного обеспечения флагов в значения соответствующие определенной архитектуре. Другое свойство это макрос “arch" в spec-файле. Оно может быть использована чтобы делать разные вещи в зависимости от архитектуры на которой производится посторонние. Еще одно свойство это директива “Exclude" в заголовке.

### 7.1 Простой spec-файл

Следующая информация это часть spec-файла для пакета “fileutils". Он настроен для построения и на Alpha и на Intel платформах.

```
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 Директива Optflags

В этом примере вы видите как директива "optflags" используется из файла `/etc/rpmrc`. В зависимости от того на какой архитектуре вы производите построение, соответствующее значение дается переменной `RPM_OPT_FLAGS`. Вы должны изменить Makefile вашего пакета для использования этой переменной вместо директив, которые вы могли бы использовать (подобно директивам `-m486` и `-O2`). Вы можете лучше почувствовать что надо сделать если загрузите этот пакет с исходными текстами, распакуйте исходные тексты и посмотрите на Makefile. Затем посмотрите на заплатку для Makefile и вы увидите какие изменения должны быть сделаны.

## 7.3 Макросы

Макрос `%ifarch` очень важен. Очень часто вам необходимо сделать одну или несколько заплаток, специфических только для одной архитектуры. В этом случае RPM позволит вам приложить эти заплатки только на этой архитектуре.
В вышеприведенном примере, fileutils имеют заплатку для 64-битовых машин. Очевидно, что она должна быть приложена только на Alpha. Так что ма добавим макрос `%ifarch` вокруг применения 64-битовой заплатки как приведено:

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

Это будет обеспечивать, что заплатка не будет приложена на любой архитектуре за исключением alpha.

## 7.4 Исключение архитектур из пакетов

Для того чтобы вы могли сопровождать пакеты с исходным текстом в одной директории для всех платформ мы реализовали возможность "исключения" построения пакетов на определенных архитектурах. Так что вы все равно можете делать такие вещи как:

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

и иметь правильно построенные пакеты. Если вы еще не перенесли приложение на определенную платформу, все что вам надо сделать это добавить примерно следующую строку:

```
ExcludeArch: axp
```

к заголовку spec-файла пакета с исходными текстами. Затем заново постройте пакет на платформе на которой он может строится. Вы будете иметь пакет с исходными текстами, который может строиться на платформе Intel и может быть легко пропущен на платформе Alpha.

### 7.5 Окончание

Использование RPM для создания многоплатформенных пакетов обычно более легко сделать, чем заставить сам пакет быть построенным в обоих местах. Как всегда наилучшая помощь когда вы застряли это посмотреть как сделан похожий пакет.

## 8 Замечание об авторских правах