rpm

COLLABORATORS

	TITLE : rpm	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 15, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 rpm **1**

1.1 rpm.guide . . . 1
1.2 rpm.guide/Introduction . . . 1
1.3 rpm.guide/Overview . . . 2
1.4 rpm.guide/General Information . . . 3
1.5 rpm.guide/Acquiring RPM . . . 3
1.6 rpm.guide/Acquiring RPM for your Linux system- . . . 3
1.7 rpm.guide/Acquiring RPM for your AmigaOS system- . . . 4
1.8 rpm.guide/RPM Requirements . . . 4
1.9 rpm.guide/Installing RPM . . . 5
1.10 rpm.guide/Installing RPM on Linux . . . 5
1.11 rpm.guide/Installing RPM on AmigaOS . . . 5
1.12 rpm.guide/Using RPM . . . 6
1.13 rpm.guide/Now what can I `really' do with RPM? . . . 9
1.14 rpm.guide/Building RPMs . . . 10
1.15 rpm.guide/The rpmrc File . . . 11
1.16 rpm.guide/The Spec File . . . 12
1.17 rpm.guide/The Header . . . 13
1.18 rpm.guide/Prep . . . 15
1.19 rpm.guide/Build . . . 16
1.20 rpm.guide/Install . . . 17
1.21 rpm.guide/Optional pre and post Install-Uninstall Scripts . . . 17
1.22 rpm.guide/Files . . . 17
1.23 rpm.guide/Building It . . . 18
1.24 rpm.guide/The Source Directory Tree . . . 18
1.25 rpm.guide/Test Building . . . 19
1.26 rpm.guide/Generating the File List . . . 19
1.27 rpm.guide/Building the Package with RPM . . . 20
1.28 rpm.guide/Testing It . . . 21
1.29 rpm.guide/What to do with your new RPMs . . . 21

1.30 rpm.guide/Advanced RPM Building . . . 21
1.31 rpm.guide/How to Get Started . . . 22
1.32 rpm.guide/Sub-Packages . . . 24
1.33 rpm.guide/Adv The Header . . . 24
1.34 rpm.guide/Adv Prep . . . 24
1.35 rpm.guide/Adv Build . . . 24
1.36 rpm.guide/Adv Install . . . 25
1.37 rpm.guide/Adv Optional pre and post Install-Uninstall Scripts . . . 25
1.38 rpm.guide/Adv Files . . . 25
1.39 rpm.guide/What Now? . . . 26
1.40 rpm.guide/Multi-architectural RPM Building . . . 26
1.41 rpm.guide/Sample spec File . . . 26
1.42 rpm.guide/Optflags . . . 27
1.43 rpm.guide/Macros . . . 28
1.44 rpm.guide/Excluding Architectures from Packages . . . 28
1.45 rpm.guide/Finishing Up . . . 28
1.46 rpm.guide/Copyright Notice . . . 29

Chapter 1

rpm

1.1 rpm.guide

```
                RPM
***

     Kristof Depraetere, 'Kristof.Depraetere@rug.ac.be'
     V1, August 15, 1996

     Based on:
     RPM HOWTO by Donnie Barnes, djb@redhat.com
     Copyright 1995, Red Hat Software


                 Introduction

                 Overview

                 General Information

                 Installing RPM

                 Using RPM

                 Now what can I 'really' do with RPM?

                 Building RPMs

                 Advanced RPM Building

                 Multi-architectural RPM Building

                 Copyright Notice
```

1.2 rpm.guide/Introduction

Introduction
****

RPM is the ‘R’ed Hat ‘P’ackage ‘M’anager. While it does contain Red Hat in the name, it is completely intended to be an open packaging system available for anyone to use. It allows users to take source code for new software and package it into source and binary form such that binaries can be easily installed and tracked and source can be rebuilt easily. It also maintains a database of all packages and their files that can be used for verifying packages and querying for information about files and/or packages.

Red Hat Software encourages other distribution vendors to take the time to look at RPM and use it for their own distributions. RPM is quite flexible and easy to use, though it provides the base for a very extensive system. It is also completely open and available, though we would appreciate bug reports and fixes. Permission is granted to use and distribute RPM royalty free under the GPL.

RPM can even provide an excellent method to upgrade an existing system. The database won’t be as up to date as a machine that was completely installed with RPM, but it will still contain anything installed with RPM. It can also be used to package commercial software.

1.3 rpm.guide/Overview

Overview
****

First, let me state some of the philosophy behind RPM. One design goal was to allow the use of "pristine" sources. With RPP (our former packaging system of which ‘none’ of RPM is derived), our source packages were the "hacked" sources that we built from. Theoretically, one could install a source RPP and then ‘make’ it with no problems. But the sources were not the original ones, and there was no reference as to what changes we had to make to get it to build. One had to download the pristine sources separately. With RPM, you have the pristine sources along with a patch that we used to compile from. We see this as a big advantage. Why? Several reasons. For one, if a new version of a program comes out, you don’t necessarily have to start from scratch to get it to compile under RHL. You can look at the patch to see what you ‘might’ need to do. All the compile-in defaults are easily visible this way.

RPM is also designed to have powerful querying options. You can do searches through your entire database for packages or just certain files. You can also easily find out what package a file belongs to and where it came from. The RPM files themselves are compressed archives, but you can query individual packages easily and ‘quickly’ because of a custom binary header added to the package with everything you could possibly need to know contained in uncompressed form. This allows for ‘fast’ querying.

Another powerful feature is the ability to verify packages. If you are worried that you deleted an important file for some package, just verify it. You will be notified of any anomalies. At that point, you can reinstall the package if necessary. Any config files that you had are preserved as well.

We would like to thank the folks from the BOGUS distribution for many of their ideas and concepts that are included in RPM. While RPM was completely written by Red Hat Software, its operation is based on code written by BOGUS (PM and PMS).

1.4 rpm.guide/General Information

```
                General Information
*******************
```

Acquiring RPM

RPM Requirements

1.5 rpm.guide/Acquiring RPM

```
                Acquiring RPM
=============
```

Acquiring RPM for your Linux system-

Acquiring RPM for your AmigaOS system-

1.6 rpm.guide/Acquiring RPM for your Linux system-

```
Acquiring RPM for your Linux system-
------------------------------------
```

The best way to get RPM is to install Red Hat Linux. If you don't want to do that, you can still get and use RPM. It can be acquired from any Official Red Hat Mirror. Some of those are:

FTP Site	Directory
ftp.pht.com	/pub/linux/redhat
sunsite.unc.edu	/pub/Linux/distributions/redhat
sunsite.doc.ic.ac.uk	/packages/linux/sunsite.unc-mirror/distributions/redhat/redhat-2.0

ftp.cms.uncwil.edu	/linux/redhat
ftp.wilmington.net	/linux/redhat
ftp.caldera.com	/pub/mirrors/redhat
ftp.lasermoon.co.uk	/pub/distributions/RedHat
ftp.cc.gatech.edu	/pub/linux/distributions/redhat
uiarchive.cso.uiuc.edu	/pub/systems/linux/distributions/redhat
ftp.ibp.fr	/pub/linux/distributions/redhat
ftp.gwdg.de	/pub/linux/install/redhat
ftp.uoknor.edu	/linux/redhat
ftp.msu.ru	/pub/Linux/RedHat
linux.ucs.indiana.edu	/pub/linux/redhat
ftp.cvut.cz	/pub/linux/redhat
ftp.ton.tut.fi	/pub/Linux/RedHat
ftp.funet.fi	/pub/Linux/images/RedHat

We are unsure at this point where to find it past there, but it will most likely just be a directory called RPM. We will make a tar file available with a README containing all the install instructions you should need.

1.7 rpm.guide/Acquiring RPM for your AmigaOS system-

Acquiring RPM for your AmigaOS system-

You can find an AmigaOS binary distribution of RPM on these sites:

FTP Site	Directory
ftp.ninemoons.com	/pub/ade/
ftp.grolier.fr	/pub/amiga/ade/

1.8 rpm.guide/RPM Requirements

RPM Requirements

You must have a working copy of ‘cpio’, ‘tar’ and ‘gunzip’, which most Linux distributions have now. While this system is intended for use with Linux, it may very well be portable to other Unix systems who meet the above conditions. It also works on the Amiga with AmigaOS and IXEmul. Be warned, the binary packages generated on a different type of Unix system will not be compatible.

Those are the minimal requirements to install RPMs. To build RPMs from source, you also need everything normally required to build a package, like ‘gcc’, ‘make’, etc.

1.9 rpm.guide/Installing RPM

```
                Installing RPM
**************


                 Installing RPM on Linux

                 Installing RPM on AmigaOS
```

1.10 rpm.guide/Installing RPM on Linux

```
Installing RPM on Linux
=======================

   Before you start make sure that you have an Linux version of RPM. If
you got it, follow these steps carefully:

   * cd /

   * tar -zxvf rpm-2.2.2-bin.tar.gz

   * makedir /var/lib/rpm

   * edit the 'rpmrc' file in the 'etc:' directory. Change the 'topdir'
     directory to reflect your setup.

   * create these directories in the 'topdir' directory:
        * SOURCES

        * SPECS

        * SRPMS

        * RPMS/i386

   * now initialize the RPM database with the command:
                  rpm --initdb

   Everything is now setup for using RPM.
```

1.11 rpm.guide/Installing RPM on AmigaOS

```
Installing RPM on AmigaOS
=========================

   Before you start make sure that you have an AmigaOS version of RPM.
If you got it, follow these steps carefully:
```

```
   * cd <your-build-directory>

   * tar -xzvf rpm-2.2.4-src.tar.gz

   * cd rpm-2.2.4

   * configure --prefix=/ade

   * make

   * make install

   Now it's time to edit the file "ade:lib/rpmrc" to correspond to your
system setup. Change the following entries:
   * dppath: /ade/lib/rpm

   * topdir: /place/where/you/want/to/store/RPM/packages

   Also edit the file "etc:rpmrc". Replace the <topdir> entry with a
<builddir> entry (note UNIX style path names):
   * builddir: /place/where/RPM/will/build/packages

   It is recommended to let <topdir> and <builddir> point to two
different partitions.

   Now create the following directories in <topdir>:
   * RPMS

   * RPMS/m68k

   * SOURCES

   * SRPMS

   * SPECS

   Create the "rpm" directory in "ade:lib"

   Now it's time to initialise the rpm database with:
             rpm --initdb

   Everything is now setup for using RPM.
```

1.12 rpm.guide/Using RPM

```
Using RPM
*********

   In its simplest form, RPM can be used to install packages:
             rpm -i foobar-1.0-1.i386.rpm

   The next simplest command is to uninstall a package:
             rpm -u foobar
```

One of the more complex but 'highly' useful commands allows you to install packages via FTP. If you are connected to the net and want to install a new package, all you need to do is specify the file with a valid URL, like so:

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

Please note, however, that the current version of RPM will only do installs via FTP. You cannot run any of the more complex query options on packages at an FTP site.

While these are simple commands, rpm can be used in a multitude of ways as seen from the 'Usage' message:

```
rpm version 1.4.5
Copyright (C) 1995 - Red Hat Software
This may be freely redistributed under the terms of the GNU Public License

usage: rpm {--help}
       rpm {--version}
       rpm {--install -i} [-v] [--hash -h] [--percent] [--force] [--test]
                          [--search] [--root <dir>] file1.rpm ... filen.rpm
       rpm {--upgrade -U} [-v] [--hash -h] [--percent] [--force] [--test]
                          [--search] [--root <dir>] file1.rpm ... fileN.rpm
       rpm {--query -q} [-afFpP] [-i] [-l] [-s] [-d] [-c] [-v]
                        [--root <dir>] [targets]
       rpm {--verify -V -y} [-afFpP] [--root <dir>] [targets]
       rpm {--uninstall -u} [--root <dir>] package1 package2 ... packageN
       rpm {-b}[plciba] [-v] [--short-circuit] [--clean] [--keep-temps]
                        [--test] [--time-check <s>] specfile
       rpm {--rebuild} [-v] source1.rpm source2.rpm ... sourceN.rpm
       rpm {--where} package1 package2 ... packageN
```

First, I'll go through a synopsis of what all the options mean (don't worry, there may be alot of options, but we tried to make them all as intuitive as possible).

Options are nested, so the possible options are many. Here's a description in parallel with the 'Usage' message:

* 'help' prints the usage message

* '-i' installs an rpm file.
    * '-hash, -h' is a very cool option for watching the package install (much like 'hash' in ftp).

    * '-percent' prints the percentages as a package installs (but is only useful for interfacing with other tools...is not really human readable).

    * '-force' will force an install of a binary package even though it may already exist in the database.

    * '-test' will tell you if installing would work or not (do you have a conflict with an already installed package).

    * '-root' will install a package using the root prefix specified instead of using the default of '/'.

* '-install' installs an rpm file.

* '-U' upgrades a package. This option installs the new package and then uninstalls the old one without hurting the new one. The upgrade option takes the same options as the install option.

* '-q' is the query option. In its simplest form, you can do 'rpm -q foobar' which would return 'foobar-1.0-1'. (1.0 is the version number, 1 is the release number.)

* Several options may be used with '-q':
    * '-a' will query all currently installed packages.

    * '-f <file>' will query the package owning <file>.

    * '-F' is the same as '-f' except you can give it filenames via stdin (ie. 'ls /usr/bin | rpm -qF').

    * '-p <packagefile>' will query the package. It is really only useful when combined with one of the Information Selection Options below.

    * '-P' is like -p, except it takes its package filenames from stdin (ie. 'ls /mnt/redhat/redhat-2.0/RPMS | rpm -qP').

    * '-root' will query a mounted filesystem.

* Several Information Selection Options can be used with any combination of the above options. If none is given, the package name only is displayed.
    * '-i' displays package information such as Name, Description, Release, etc.

    * '-l' will display the file list from the entire package (all files that get installed). You can also use a '-v' with this to make the file list much more verbose.

    * '-s' shows you the state of all the files in the package. There are only two possible states, normal and missing.

    * '-d' outputs a list of just the files marked as documentation (man pages, info pages, READMEs, etc). '-v' will give even more info.

    * '-c' outputs a list of only the configuration files (sendmail.cf, passwd, inittab, etc.) '-v' will give more info about the files.

* '{-V,-y,-verify}' are the verify options. All are interchangeable. They all take the same Package Specification and Information Selection options as the '-q' option. I'll list some examples:
    * To verify a package containing particular file, do:
        rpm -yf /bin/vi

    * To verify ALL your files, do:
        rpm -ya

```
      * To verify files on your system versus the files in a .rpm
        file, do:
             rpm -Vp foobar-1.0-1.rpm

  * '-uninstall, -u <package>' to uninstall a package

  * '-b' to build a package (from sources and a spec file).  This
    option will be discussed more at length in the next section,
    Building RPMs.

  * '-v' be verbose in the output of what's going on.

  * '-vv' be 'very' verbose in the output of what's going on.
```

1.13 rpm.guide/Now what can I `really' do with RPM?

```
Now what can I 'really' do with RPM?
************************************

   RPM is a very useful tool and, as you can see, has several options.
The best way to make sense of them is to look at some examples.  I
covered simple install/uninstall above, so here are some more examples:
   * Let's say you delete some files by accident, but you aren't sure
     what you deleted.  If you want to verify your entire system and
     see what might be missing, you would do:
          rpm -Va

   * Let's say you run across a file that you don't recognize.  To find
     out which package owns it, you would do:
          rpm -qf /usr/X11R6/bin/xjewel

     The output would be:
          xjewel-1.6-1

   * You find a new koules RPM, but you don't know what it is.  To find
     out some information on it, do:
          rpm -qpi koules-1.0-1.i386.rpm

     The output would be:
          Name        : koules                        Distribution: RHL 2.0
          Version     : 1.0                                Vendor: Red Hat Software
          Release     : 1                              Build date: Tue Aug 29  ↩
             12:53:21 1995
          Install date: <not installed>                Build host: daffy.redhat.com
          Group       : Games
          Size        : 403105
          Description : well done SVGAlib game

   * Now you want to see what files the koules RPM installs.  You would
     do:
          rpm -qpl koules-1.0-1.i386.rpm

     The output is:
          /usr/man/man6/koules.6
```

```
          /usr/lib/games/kouleslib/start.raw
          /usr/lib/games/kouleslib/end.raw
          /usr/lib/games/kouleslib/destroy2.raw
          /usr/lib/games/kouleslib/destroy1.raw
          /usr/lib/games/kouleslib/creator2.raw
          /usr/lib/games/kouleslib/creator1.raw
          /usr/lib/games/kouleslib/colize.raw
          /usr/lib/games/kouleslib
          /usr/games/koules
```

   These are just several examples. More creative ones can be thought of really easy once you are familiar with RPM.

1.14 rpm.guide/Building RPMs

```
                Building RPMs
*************
```

   Building RPMs is fairly easy to do, especially if you can get the software you are trying to package to build on its own.

   The basic procedure to build an RPM is as follows:
   * Make sure your ‘/etc/rpmrc’ is setup for your system.

   * Get the source code you are building the RPM for to build on your system.

   * Make a patch of any changes you had to make to the sources to get them to build properly.

   * Make a spec file for the package.

   * Make sure everything is in its proper place.

   * Build the package using RPM.

   Under normal operation, RPM builds both binary and source packages.

```
                The rpmrc File

                The Spec File

                The Header

                Prep

                Build

                Install

                Optional pre and post Install-Uninstall Scripts
```

```
                Files

                Building It

                Testing It

                What to do with your new RPMs
```

1.15 rpm.guide/The rpmrc File

```
The rpmrc File
==============

   Right now, the only configuration of RPM is available via the
'/etc/rpmrc' file.  An example one looks like:
     require_vendor: 1
     require_distribution: 1
     require_group: 1
     distribution: RHL 2.0
     vendor: Red Hat Software
     arch_sensitive: 1
     topdir: /usr/src/redhat-2.0

     optflags: i386 -O2 -m486
     optflags: axp -O2

   The 'require_vendor' line causes RPM to require that it find a
vendor line.  This can come from the '/etc/rpmrc' or from the header of
the spec file itself.  To turn this off, change the number to '0'.  The
same holds true for the 'require_distribution' and 'require_group'
lines.

   The next line is the 'distribution' line.  You can define that here
or later in the header of the spec file.  When building for a particular
distribution, it's a good idea to make sure this line is correct, even
though it is not required.  The 'vendor'  line works much the same way,
but can be anything (ie. Joe's Software and Rock Music Emporium).

   The next line is 'arch_sensitive'.  This specifies where the binary
RPMs go and what they are named.  Right now, i386 is defined as a type
within RPM.  That means if you are building on an Intel machine and
have this value set to true, your RPMs will go in
'/usr/src/redhat-2.0/RPMS/i386/' and their name will be something like
'foobar-1.0-1.i386.rpm'.  If you set this value to '0', the RPMs will
be placed in '/usr/src/redhat-2.0/RPMS/' and will be named something
like ' foobar-1.0-1.bin.rpm'.  This does not affect the name or
placement of the source RPM, however.

   RPM also now has support for building packages on multiple
architectures.  The 'rpmrc' file can hold an "optflags" variable for
building things that require architecture specific flags when building.
See later sections for how to use this variable.

   In addition to the above macros, there are several more.  You can
```

```
use:
   * 'topdir' to specify the top level directory for building.  In Red
     Hat 2.0, this directory is '/usr/src/redhat-2.0'.

   * 'specdir' is the directory under topdir to use for the spec files.
     The default for this is 'SPECS'.

   * ' builddir' specifies the top level of the build directory.  The
     default for this is 'BUILD'.

   * 'sourcedir' specifies the top level of the source directory.  The
     default for this is 'SOURCES'.  This is where the pristine tar
     files, the patches, and the icons go.

   * 'rpmdir' sets the directory for the binary RPMs.  The default for
     this is 'RPMS'.

   * 'srcrpmdir' sets the directory for the source RPMs.  The default
     for this is 'SRPMS'.

   * 'docdir' specifies where the documentation should be installed.
     By default, this is '/usr/doc'.

   * 'libdir' sets the path for the RPM database.  By default, this is
     '/var/lib/rpm'.

   * 'timecheck' sets whether or not to do a timecheck by default.
```

1.16 rpm.guide/The Spec File

```
The Spec File
=============

   We'll begin with discussion of the spec file.  Spec files are
required to build a package.  The spec file is a description of the
software along with instructions on how to build it and a file list for
all the binaries that get installed.

   You'll want to name your spec file according to a standard
convention.  It should be the package name-dash-version
number-dash-release number-dot-spec.

   Here is a small spec file (vim-3.0-1.spec):
     Description: VIsual editor iMproved
     Name: vim
     Version: 3.0
     Release: 1
     Icon: vim.gif
     Source: sunsite.unc.edu:/pub/Linux/apps/editors/vi/vim-3.0.tar.gz
     Patch: vim-3.0-make.patch
     Copyright: distributable
     Group: Applications/Editors

     %prep
```

```
%setup
%patch -p1
cd src
cp makefile.unix makefile

%build
cd src
make

%install
rm -f /bin/vim
cd src
make install
ln -sf vim /bin/vi

%files
%doc doc/reference.doc doc/unix.doc tutor/tutor
/bin/vim
/bin/vi
/usr/man/man1/vim.1
/usr/lib/vim.hlp
```

1.17 rpm.guide/The Header

```
The Header
==========

   The header has some standard fields in it that you need to fill in.
There are a few caveats as well.  The fields must be filled in as
follows:
   * 'Description:' This one is kind of obvious.  You can span multiple
     lines by ending each line with a backslash.

   * 'Name:' This must be the name string from the rpm filename you
     plan to use.

   * 'Version:' This must be the version string from the rpm filename
     you plan to use.

   * 'Release:' This is the release number for a package of the same
     version (ie. if we make a package and find it to be slightly
     broken and need to make it again, the next package would be
     release number 2).

   * 'Icon:' This is the name of the icon file for use by other high
     level installation tools (like Red Hat's "glint").  It must be a
     gif and resides in the SOURCES directory.

   * 'Source:' This line points at the HOME location of the pristine
     source file.  It is used if you ever want to get the source again
     or check for newer versions.  Caveat:  The filename in this line
     MUST match the filename you have on your own system (ie. don't
     download the source file and change its name).  You can also
     specify more than one source file using lines like:
```

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

These files would go in the 'SOURCES' directory. (The directory structure is discussed in a later section, "The Source Directory Tree".)

* 'Patch:' This is the place you can find the patch if you need to download it again. Caveat: The filename here must match the one you use when you make YOUR patch. You may also want to note that you can have multiple patch files much as you can have multiple sources.] You would have something like:

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

These files would go in the 'SOURCES' directory.

* 'Copyright:' This line tells how a package is copyrighted. You should use something like GPL, BSD, MIT, public domain, distributable, or commercial.

* 'Root:' This line allows you to specify a directory as the "root" for building and installing the new package. You can use this to help test your package before having it installed on your machine.

* 'Group:' This line is used to tell high level installation programs (such as Red Hat's "glint") where to place this particular program in its hierarchical structure. The group tree currently looks something like this:

```
        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
```

1.18 rpm.guide/Prep

```
Prep
====

   This is the second section in the spec file.  It is used to get the
sources ready to build.  Here you need to do anything necessary to get
the sources patched and setup like they need to be setup to do a 'make'.

   One thing to note:  Each of these sections is really just a place to
execute shell scripts.  You could simply make an 'sh' script and put it
after the '%prep' tag to unpack and patch your sources.  We have made
macros to aid in this, however.

   The first of these macros is the '%setup' macro.  In its simplest
form (no command line options), it simply unpacks the sources and 'cd"s
into the source directory.  It also takes the following options:

   * '-n name' will set the name of the build directory to the listed
     'name'.  The default is '$NAME-$VERSION'.  Other possibilities
     include '$NAME', '${NAME}${VERSION}', or whatever the main tar file
     uses.
```

* '-c' will create and cd to the named directory 'before' doing the untar.

* '-b #' will untar Source# 'before' cd'ing into the directory (and this makes no sense with '-c' so don't do it). This is only useful with multiple source files.

* '-a #' will untar Source# 'after' cd'ing into the directory.

* '-T' This option overrides the default action of untarring the Source and requires a '-b 0' or '-a 0' to get the main source file untarred. You need this when there are secondary sources.

* '-D' Do 'not' delete the directory before unpacking. This is only useful where you have more than one setup macro. It should 'only' be used in setup macros 'after' the first one (but never in the first one).

The next of the available macros is the '%patch' macro. This macro helps automate the process of applying patches to the sources. It takes several options, listed below:

* '#' will apply Patch# as the patch file.

* '-p #' specifies the number of directories to strip for the patch(1) command.

* '-P' The default action is to apply 'Patch' (or 'Patch0'). This flag inhibits the default action and will require a '0' to get the main source file untarred. This option is useful in a second (or later) '%patch' macro that required a different number than the first macro.

* You can also do '%patch#' instead of doing the real command: '%patch # -P'

That should be all the macros you need. After you have those right, you can also do any other setup you need to do via 'sh' type scripting. Anything you include up until the '%build' macro (discussed in the next section) is executed via 'sh'. Look at the example above for the types of things you might want to do here.

1.19 rpm.guide/Build

```
Build
=====
```

There aren't really any macros for this section. You should just put any commands here that you would need to use to build the software once you had untarred the source, patched it, and cd'ed into the directory. This is just another set of commands passed to 'sh', so any legal 'sh' commands can go here (including comments). Your current working directory is reset in each of these sections to the toplevel of the source directory, so keep that in mind. You can 'cd' into subdirectories if necessary.

1.20 rpm.guide/Install

```
Install
=======

   There aren't really any macros here, either.  You basically just
want to put whatever commands here that are necessary to install.  If
you have `make install' available to you in the package you are
building, put that here.  If not, you can either patch the makefile for
a `make install' and just do a `make install' here, or you can hand
install them here with `sh' commands.  You can consider your current
directory to be the toplevel of the source directory.
```

1.21 rpm.guide/Optional pre and post Install-Uninstall Scripts

```
Optional pre and post Install/Uninstall Scripts
===============================================

   You can put scripts in that get run before and after the installation
and uninstallation of binary packages.  A main reason for this is to do
things like run `ldconfig' after installing or removing packages that
contain shared libraries.  The macros for each of the scripts is as
follows:
   * `%pre' is the macro to do pre-install scripts.

   * `%post' is the macro to do post-install scripts.

   * `%preun' is the macro to do pre-uninstall scripts.

   * `%postun' is the macro to do post-uninstall scripts.

   The contents of these sections should just be any `sh' style script,
though you do `not' need the `#!/bin/sh'.
```

1.22 rpm.guide/Files

```
Files
=====

   This is the section where you `must' list the files for the binary
package.  RPM has no way to know what binaries get installed as a
result of `make install'.  There is `NO' way to do this.  Some have
suggested doing a `find' before and after the package install.  With a
multiuser system, this is unacceptable as other files may be created
during a package building process that have nothing to do with the
package itself.
```

There are some macros available to do some special things as well. They are listed and described here:

* '%doc' is used to mark documentation in the source package that you want installed in a binary install. The documents will be installed in '/usr/doc/$NAME-$VERSION-$RELEASE'. You can list multiple documents on the command line with this macro, or you can list them all separately using a macro for each of them.

* '%config' is used to mark configuration files in a package. This includes files like sendmail.cf, passwd, etc. If you later uninstall a package containing config files, any unchanged files will be removed and any changed files will get moved to their old name with a '.rpmsave' appended to the filename. You can list multiple files with this macro as well.

* '%dir' marks a single directory in a file list to be included as being owned by a package. By default, if you list a directory name 'WITHOUT' a '%dir' macro, 'EVERYTHING' in that directory is included in the file list and later installed as part of that package.

The biggest caveat in the file list is listing directories. If you list '/usr/bin' by accident, your binary package will contain 'every' file in '/usr/bin' on your system.

1.23 rpm.guide/Building It

```
                Building It
===========
```

The Source Directory Tree

Test Building

Generating the File List

Building the Package with RPM

1.24 rpm.guide/The Source Directory Tree

```
The Source Directory Tree
-------------------------
```

The first thing you need is a properly configured build tree. This is configurable using the '/etc/rpmrc' file. Most people will just use '/usr/src'.

```
   You may need to create the following directories to make a build
tree:
   * 'BUILD' is the directory where all building occurs by RPM.  You
     don't have to do your test building anywhere in particular, but
     this is where RPM will do it's building.

   * 'SOURCES' is the directory where you should put your original
     source tar files and your patches.  This is where RPM will look by
     default.

   * 'SPECS' is the directory where all spec files should go.

   * 'RPMS' is where RPM will put all binary RPMs when built.

   * 'SRPMS' is where all source RPMs will be put.
```

1.25 rpm.guide/Test Building

```
Test Building
-------------

   The first thing you'll probably want to to is get the source to
build cleanly without using RPM.  To do this, unpack the sources, and
change the directory name to $NAME.orig.  Then unpack the source again.
Use this source to build from.  Go into the source directory and
follow the instructions to build it.  If you have to edit things,
you'll need a patch.  Once you get it to build, clean the source
directory.  Make sure and remove any files that get made from a
'./configure'.  Then 'cd' back out of the source directory to its
parent.  Then you'll do something like:
             diff -uNr dirname.orig dirname > ../SOURCES/dirname-linux.patch

   This will create a patch for you that you can use in your spec file.
Note that the "linux" that you see in the patch name is just an
identifier.  You might want to use something more descriptive like
"config" or "bugs" to describe 'why' you had to make a patch.  It's
also a good idea to look at the patch file you are creating before
using it to make sure no binaries were included by accident.
```

1.26 rpm.guide/Generating the File List

```
Generating the File List
------------------------

   Now that you have source that will build and you know how to do it,
build it and install it.  Look at the output of the install sequence and
build your file list from that to use in the spec file.  We usually
build the spec file in parallel with all of these steps.  You can
create the initial one and fill in the easy parts, and then fill in the
other steps as you go.
```

1.27 rpm.guide/Building the Package with RPM

```
Building the Package with RPM
-----------------------------
```

Once you have a spec file, you are ready to try and build your package. The most useful way to do it is with a command like the following:

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

There are other options useful with the ‘-b’ switch as well:

* ‘p’ means just run the ‘prep’ section of the specfile.

* ‘l’ is a list check that does some checks on ‘%files’.

* ‘c’ do a prep and compile. This is useful when you are unsure of whether your source will build at all. It seems useless because you might want to just keep playing with the source itself until it builds and then start using RPM, but once you become accustomed to using RPM you will find instances when you will use it.

* ‘i’ do a prep, compile, and install.

* ‘b’ prep, compile, install, and build a binary package only.

* ‘a’ build it all (both source and binary packages).

There are several modifiers to the ‘-b’ switch. They are as follows:

* ‘-short-circuit’ will skip straight to a specified stage (can only be used with c and i).

* ‘-clean’ removes the build tree when done.

* ‘-keep-temps’ will keep all the temp files and scripts that were made in /tmp. You can actually see what files were created in /tmp using the ‘-v’ option.

* ‘-test’ does not execute any real stages, but does keep-temp.

* ‘-time-check #’ is very useful. By default, the time-check value is 7200 seconds (two hours). What this does is check all the files in ‘%files’ and warns you if they are more than ‘#’ seconds old (or the default). This lets you make sure that the newly created binaries are getting installed and not old ones that just happen to be still lying around. This author can attest to the value of this feature after having to release several RPP updates because old binaries were accidentally included. You can also turn this off (useful when building binary only packages of commercial software) by setting the value to zero.

1.28 rpm.guide/Testing It

```
Testing It
==========
```

Once you have a source and binary rpm for your package, you need to test it. The easiest and best way is to use a totally different machine from the one you are building on to test. After all, you've just done a lot of 'make install"s on your own machine, so it should be installed fairly well.

You can do an 'rpm -u packagename' on the package to test, but that can be deceiving because in building the package, you did a 'make install'. If you left something out of your file list, it will not get uninstalled. You'll then reinstall the binary package and your system will be complete again, but your rpm still isn't. Make sure and keep in mind that just because you do a 'rpm -ba package', most people installing your package will just be doing the 'rpm -i package'. Make sure you don't do anything in the 'build' or 'install' sections that will need to be done when the binaries are installed by themselves.

1.29 rpm.guide/What to do with your new RPMs

```
What to do with your new RPMs
=============================
```

Once you've made your own RPM of something (assuming its something that hasn't already been RPM'ed), you can contribute your work to others (also assuming you RPM'ed something freely distributable). To do so, you'll want to upload it to an FTP site somewhere. We hope RPM will become a standard that everyone starts using. If that is the case, you should probably upload your RPMs to sunsite.unc.edu. Until then, please upload them to our official Red Hat Mirror, ftp.pht.com:/pub/linux/redhat/Incoming. We are currently mirrored on several other sites, and this is the best place to find new RPMs.

1.30 rpm.guide/Advanced RPM Building

```
                Advanced RPM Building
**********************
```

RPM has some very advanced features available for larger, more complex packages. It has the ability to build and output multiple binary subpackages. An example of this is the ability to produce separate Tcl/Tk binary packages from one spec file. Another example is the ability to use one spec file to create a single XFree86 package with no servers, and a separate package for each of the servers.

How to Get Started

Sub-Packages

Adv The Header

Adv Prep

Adv Build

Adv Install

Adv Optional pre and post Install-Uninstall Scripts

Adv Files

What Now?
 What to do with your create RPM file

1.31 rpm.guide/How to Get Started

```
How to Get Started
==================

   The best way to get started is to look at an example spec file.  The
following tcl/tk spec file is a good one to start with (though you can
also view the spec file of any package by installing the sources and
looking in '/usr/src/redhat-2.0/SPECS'):
     %package tcl
     Description: Tool Command Language
     Name: tcltk
     Version: 7.4_4.0
     Release: 1
     Icon: tcl.gif
     Source0: ftp.cs.berkeley.com:/pub/tcl/tcl7.4.tar.Z
     Source1: ftp.cs.berkeley.com:/pub/tcl/tk4.0.tar.Z
     Copyright: BSD
     Group: Development/Languages/Tcl
     Patch0: sunsite.unc.edu:/pub/Linux/devel/tcl7.4-1.diff.gz
     Patch1: sunsite.unc.edu:/pub/Linux/devel/tk4.0-1.diff.gz
     %package tk
     Icon: tk.gif
     Description: Tk toolkit
     Group: Development/Languages/Tcl

     %prep
     %setup -T -c -a 0
     %setup -T -D -a 1
     %patch0 -p0
     %patch1 -p0

     %build
     cd tcl7.4
```

```
./configure --prefix=/usr
make
cd ../tk4.0
./configure --prefix=/usr
make

%install
cd tcl7.4
make install
ln -sf libtcl7.4.a /usr/lib/libtcl.a
ln -sf libtcl7.4.so.1 /usr/lib/libtcl.so.1
ln -sf libtk4.0.a /usr/lib/libtk.a
ln -sf libtk4.0.so.1 /usr/lib/libtk.so.1
cd ../tk4.0
make install

%post tcl
/sbin/ldconfig

%post tk
/sbin/ldconfig

%postun tcl
/sbin/ldconfig

%postun tk
/sbin/ldconfig

%files tcl
/usr/lib/libtcl7.4.a
/usr/lib/libtcl.a
/usr/lib/libtcl7.4.so.1
/usr/lib/libtcl.so.1
/usr/include/tcl/tcl*
/usr/bin/tclsh
/usr/bin/tclsh7.4
/usr/lib/tcl7.4
/usr/man/man1/*tcl
/usr/man/man3/*tcl

%files tk
/usr/lib/libtk4.0.a
/usr/lib/libtk.a
/usr/lib/libtk4.0.so.1
/usr/lib/libtk.so.1
/usr/include/tcl/ks_names.h
/usr/include/tcl/default.h
/usr/include/tcl/tk*
/usr/lib/tk4.0
/usr/man/man1/*tk
/usr/man/man3/*tk
/usr/bin/wish
/usr/bin/wish4.0
```

1.32 rpm.guide/Sub-Packages

```
Sub-Packages
============

   One of the main advanced features of RPM is the ability to build
subpackages.  They are easy to build as for most macros you can just
add the subpackage name as a parameter for anything specific to a
subpackage (and if you leave it off the section will apply to the main
package).
```

1.33 rpm.guide/Adv The Header

```
The Header
==========

   The header only has one major difference, the '%package' macro.
This macro is used in the header to tell which subpackage name to match
the description with.  If you omit the macro in the initial part of the
header, you will get a main package with no change to the name.  In the
XFree86 package, however, there is no '%package' macro in the top of
the header.  This is because we wanted a base XFree86 package with all
the common stuff in it and then several subpackages (XFree86-SVGA,
etc.) with the servers.  Tcl/Tk does not need a main package, so the
macro is at the top.

   Another difference is the fact that this package has multiple source
and patch lines.  If you'll notice, there is now a 'Source0' line
instead of just 'Source'.  They are functionally equivalent, though it
is a good idea to use 'Source0' when there is more than one source file
(and the same applies to patches as well).
```

1.34 rpm.guide/Adv Prep

```
Prep
====

   Prep is basically the same as in the simple example, except it uses
more of the options available to the setup and patch macros.
```

1.35 rpm.guide/Adv Build

```
Build
=====

   Build is basically the same, with the exception that the setup macro
```

above used the ‘-T’ option. Because of that, you have to do a manual ‘cd’ to get into the source directory.

You will also notice that the build does a ‘configure’ before it can build. This is the section where any of this type of configuration should go.

1.36 rpm.guide/Adv Install

Install
=======

Again, everything is pretty normal with the exception of the fact that you must manually ‘cd’ into the source directories.

1.37 rpm.guide/Adv Optional pre and post Install-Uninstall Scripts

Optional pre and post Install/Uninstall Scripts
===

This section is almost the same as in a simple RPM case (see the above section). It has two post install scripts that run ldconfig for each of the subpackages upon install. It ‘should’ have two post uninstall scripts to run ldconfig as well.

1.38 rpm.guide/Adv Files

Files
=====

Here you will declare which files go in which packages. You really have multiple file sections, each started with a new ‘%files’ macro and the name of the subpackage (except in the case where you have a main package...that ‘%files’ macro will have no argument given to it). The other macros (doc, config, etc) work exactly the same as in the simple case.

You also have the option to use the ‘*’ to glob filenames out of a directory. You need to be careful with this (perhaps test it first) so as not to include files you didn’t mean to. The above example does this with the man pages.

1.39 rpm.guide/What Now?

```
What Now?
=========
```

Please see the above sections on Testing and What to do with new RPMs. We want all the RPMs available we can get, and we want them to be good RPMs. Please take the time to test them well, and then take the time to upload them for everyone's benefit. Also, 'please' make sure you are only uploading 'freely available software'. Commercial software and shareware should 'not' be uploaded unless they have a copyright expressly stating that this is allowed.

1.40 rpm.guide/Multi-architectural RPM Building

```
                Multi-architectural RPM Building
*********************************
```

RPM can now be used to build packages for the Intel i386, the Digital Alpha running Linux, and the Sparc. It has been reported to work on SGI's and HP workstations as well. There are several features that make building packages on all platforms easy. The first of these is the "optflags" directive in the '/etc/rpmrc'. It can be used to set flags used when building software to architecture specific values. Another feature is the "arch" macros in the spec file. They can be used to do different things depending on the architecture you are building on. Another feature is the "Exclude" directive in the header.

Sample spec File

Optflags

Macros

Excluding Architectures from Packages

Finishing Up

1.41 rpm.guide/Sample spec File

```
Sample spec File
================
```

The following is the spec file for the "zoneinfo" package. It is setup to build on both the Alpha and the Intel.

```
    Description: Time zone utilities and data
    Name: zoneinfo
    Version: 95e
```

```
Release: 2
Copyright: Distributable
Group: Utilities/System
Source0: elsie.nci.nih.gov:/pub/tzcode95e.tar.gz
Source1: elsie.nci.nih.gov:/pub/tzdata95i.tar.gz
Patch0: zoneinfo-95e-make.patch
Patch1: zoneinfo-95e-64bit.patch

%prep
%setup -c -a 1
%patch0 -p1

%ifarch axp
%patch1 -p1
%endif

%build
make RPM_OPT_FLAGS="${RPM_OPT_FLAGS}"

%install
rm -rf /usr/lib/zoneinfo

make install

rm -f /usr/lib/zoneinfo/localtime /usr/lib/zoneinfo/posixrules /usr/lib/ ↩
   zoneinfo/posixtime
ln -sf ../../../etc/localtime /usr/lib/zoneinfo/localtime
ln -sf localtime /usr/lib/zoneinfo/posixrules
ln -sf localtime /usr/lib/zoneinfo/posixtime

strip /usr/sbin/zic /usr/sbin/zdump

%files
%doc README Theory
/usr/lib/zoneinfo
/usr/lib/libz.a
/usr/sbin/zic
/usr/sbin/zdump
/usr/man/man3/newctime.3
/usr/man/man3/newtzset.3
/usr/man/man5/tzfile.5
/usr/man/man8/zdump.8
/usr/man/man8/zic.8
```

1.42 rpm.guide/Optflags

```
Optflags
========

   In this example, you see how the "optflags" directive is used from
the `/etc/rpmrc'.  Depending on which architecture you are building on,
the proper value is given to `RPM_OPT_FLAGS'.  You must patch the
Makefile for your package to use this variable in place of the normal
directives you might use (like `-m486' and `-O2').  You can get a
```

better feel for what needs to be done by installing this source package
and then unpacking the source and examine the Makefile. Then look at
the patch for the Makefile and see what changes must be made.

1.43 rpm.guide/Macros

```
Macros
======
```

   The ‘%ifarch’ macro is very important to all of this. Most times you
will need to make a patch or two that is specific to one architecture
only. In this case, RPM will allow you to apply that patch to just one
architecture only.

   In the above example, zoneinfo has a patch for 64 bit machines.
Obviously, this should only be applied on the Alpha at the moment. So,
we add an ‘%ifarch’ macro around the 64 bit patch like so:

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

   This will insure that the patch is not applied on any architecture
except the alpha.

1.44 rpm.guide/Excluding Architectures from Packages

```
Excluding Architectures from Packages
=====================================
```

   So that you can maintain source RPMs in one directory for all
platforms, we have implemented the ability to "exclude" packages from
being built on certain architectures. This is so you can still do
things like

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

   and have the right packages build. If you haven’t yet ported an
application to a certain platform, all you have to do is add a line
like:

```
     Exclude: axp
```

   to the header of the spec file of the source package. Then rebuild
the package on the platform that it does build on. You’ll then have a
source package that builds on an Intel and can easily be skipped on an
Alpha.

1.45 rpm.guide/Finishing Up

Finishing Up
============

Using RPM to make multi-architectural packages is usually easier to do than getting the package itself to build both places. As more of the hard packages get built this is getting much easier, however. As always, the best help when you get stuck building an RPM is to look a similar source package.

1.46 rpm.guide/Copyright Notice

Copyright Notice
****