Using the RPM package manager for (LA)TEX packages

Tristan Miller

Abstract

RPM is a package management system which provides a uniform, automated way for users to install, upgrade, and uninstall programs. Because RPM is the default software distribution format for many operating systems (particularly GNU/Linux), users may find it useful to manage their library of TEX-related packages using RPM. This article explains how to produce RPM files for TEX software, either for personal use or for public distribution. We also explain how a (LA)TEX user can find, install, and remove TEX-related RPM packages.

1 Background

1.1 The evolution of package management systems

In the first decade or two of personal computing, most software was distributed on and run directly from floppy disks. Users lucky enough to have a hard drive could copy the contents of these floppies into a directory on their hard disk and run it from there. When a user wanted to delete the program, he had to remember which directory he had copied it to, find it in his file system, and manually delete it, taking care to first preserve any data files he had created.

As programs and hard disk capacities grew in size, software was increasingly distributed on multiple floppy disks or (later) on a CD-ROM. Vendors would provide tools — usually simple shell scripts — to automate the process of creating a directory on the hard drive, copying the contents of each floppy to it, configuring the installed copy of the program, and then finally deleting it when the user requested that it be uninstalled.

These tools grew in sophistication along with the underlying operating systems (OSes), which by the 1990s had begun to provide standard hardware drivers and graphical interface toolkits for third-party software to use. Now software installation programs could not merely copy themselves to the hard drive; they also had to search for the presence and location of requisite system and third-party software, register themselves with the OS so that they too could be found by other programs, and create icons for the user in the system menu or desktop. Some of the better installers would also do sanity checks such as making sure the user didn't install two copies

of the same software package, or automatically detecting and upgrading older installed versions of the software. With the advent of dial-up bulletin board systems and eventually home Internet access, it became important for software to be downloadable in a single file rather than as dozens or hundreds of individual files as was the case with physical media.

Because each vendor wrote its own installation program, users often found themselves confused by different interfaces and lacking a single tool with which to install, upgrade, and remove software. To remedy this, each operating system developed its own standard software package management tool to be used by all users and vendors. Software developers can now distribute their programs in specially prepared *packages* containing the source code and/or binary executables for the software along with important metadata such as the software's name, version number, vendor, and dependencies on other software. Packages might also include checksums or cryptographic signatures which the package management system can use to verify that they were not corrupted or tampered with during distribution.

Users install and remove these packages using standard system software, which keeps a database of all installed packages and makes sure that all dependencies are met — for example, by automatically fetching prerequisite packages, or warning the user if he is about to remove a package that other installed software depends on.

1.2 Package management and TEX

Unfortunately for fans of quality typesetting, TEX and friends are currently stuck in the Dark Ages of software package management. Though TEX distributions now mostly conform to the TEX Directory Structure [8], which specifies standard locations for the installation of certain types of files, there is currently no standard package format or associated tool for installing, upgrading, and removing macro packages, styles, classes, scripts, fonts, documentation, and other TEX-related paraphernalia available on CTAN and elsewhere.

As a result, users who download new packages must themselves check for prerequisites, manually process `.ins` and `.dtx` files, create the appropriate directories in their `texmf` tree, copy the files in, and perform necessary post-installation configuration (such as running `texhash`). Worse yet, when a user wishes to uninstall a package, he must manually remove the files, often from multiple directories. This usually entails consulting the original package installation instructions to help remember what got installed where.

Fortunately, until such time as the TEX community develops and settles on its own package management standard, users can avail themselves of their operating system's native package management system for the maintenance of TEX packages.[1] In this article, we describe how to do this using the RPM Package Manager, a packaging system originally developed by Red Hat and now in widespread use on several operating systems.

1.3 RPM versus other package managers

RPM has a number of good features to recommend itself to TEX package management. Most important is its portability — RPM enjoys the status of being the official package format specified by the Linux Standard Base [4], meaning that any LSB-compliant GNU/Linux distribution can handle RPM packages. Distributions which use RPM by default include Aurox, Fedora Core, Lycoris, Mandriva (formerly Mandrake), PCLinuxOS, PLD, Red Flag, Red Hat, and SUSE. Distributions which use a different native package format but which can handle RPM by virtue of their LSB compliance include Debian, MEPIS, Slackware, and Ubuntu.

However, RPM is by no means limited to GNU/Linux operating systems. The RPM tools have been ported to Mac OS X, Novell Netware, and some commercial Unixes. Because the tools and file format specifications are released under a free license, it is possible to reimplement them on virtually any operating system. In fact, some work has already gone into porting RPM to Microsoft Windows, with some rudimentary tools available now.

It bears mention, however, that there are many alternatives to RPM the user may wish to consider. At least one TEX distribution, MiKTEX for MS-Windows, provides its own packaging utility, `mpm` [6, §§3.2 and A.9], which has much the same functionality as RPM. However, `mpm` works only with the MiKTEX distribution, and moreover is a network tool which fetches packages from some central repository. As far as the present author is able to determine, it is not possible for package authors to create and distribute their own MikTEX packages.

Another alternative is to use your operating system's native package management system. In many cases, this requires purchasing the system's official software development kit (SDK). (In the case

[1] In this document "TEX" refers to the entire TEX system, including LATEX, METAFONT, BIBTEX, and other components. Similarly, a "TEX package" here means any set of related files distributed, installed, and maintained as a unit. This meaning includes but is not limited to LATEX 2ε packages, which are style files supplementing a document class.

of Microsoft Windows, this SDK can be downloaded for free, though it requires a gigabyte of hard drive space and comes with a restrictive license.) Another disadvantage is that the distributed packages are often bundled as executable files, adding considerable overhead (possibly several megabytes) to the size of the package. (Consider that most TEX packages are only a few kilobytes in size.) Users may also be wary of running executables for fear of viruses or spyware which the packager may have deliberately or unwittingly included.

Users of Mac OS X will be pleased to note that there exists an unofficial package management system, i-Installer,[2] which enjoys notable popularity among TEX users on the Mac. The author of this tool provides i-Installer packages, or *i-Packages*, for a number of TEX packages. Furthermore, the i-Installer distribution includes tools for users to create their own i-Packages.

1.4 About this article

This article makes liberal use of illustrative examples to help the reader understand how to use the RPM packaging tools. To help distinguish between various types of computer input and output, we employ the following typographical conventions:

- When depicting an interactive shell session, any text set in a `teletype font` marks that output by the computer, and **`bold teletype`** indicates text input by the user. At the beginning of a line, the `#` character indicates the superuser (root) shell prompt, while the `$` character indicates the shell prompt for a normal user account.
- Other instances of computer input and output are rendered in a `teletype font`. Placeholders for arguments the user is expected to specify as appropriate are rendered ⟨*like_this*⟩.
- The actual contents of configuration files created by the user are set in `small teletype text` surrounded by a box.

In §2, we give a brief overview of the RPM command-line interface and how it's used to install, upgrade, and remove packages. It is aimed at novice users who simply want to know how to install or remove a TEX RPM package they found on the Internet. Readers already familiar with installing RPM packages may wish to skip this section.

Section 3 describes how you can create RPM packages for existing TEX packages; this information is likely to be of greatest interest to package developers and distributors, but also to advanced TEX users who want to avoid the hassle of manual package management. This section assumes that you have a basic familiarity with downloading and manually installing TEX packages.

Finally, §4 briefly touches on some advanced topics for RPM packagers and distributors.

[2] `http://ii2.sourceforge.net/`

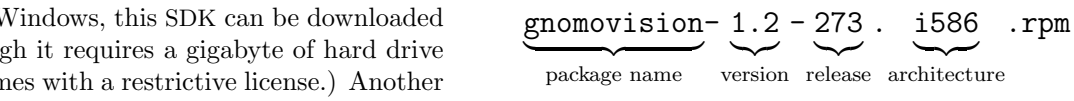

Figure 1: A sample RPM filename

2 RPM basics

2.1 RPM files and where to find them

Software for use with RPM is distributed in files known as RPM packages, which have the filename suffix `.rpm`. In order to distinguish between different versions of a package, a standard file naming scheme is employed which encodes the package name, its version and release number, and the computer architecture it is designed to work with. The syntax is illustrated with an example in Figure 1.

- The *package name* indicates the name of the software (or in our case, TEX package) packaged in the RPM.
- The *version* is the version of the software to be installed.
- The *release* field is used to indicate revisions of the packaging of that particular version of the software. For instance, sometimes the person packaging the software will make an error, such as leaving out a particular file. Every time the software is repackaged to fix such an error, its release number is increased.
- The *architecture* field indicates the type of computer processor the software is designed to work with. For most executable programs, this field will have a value such as `i586` (Intel 586), `ppc` (PowerPC), or `sparc` (Sun SPARC). Most TEX packages, however, do not depend on any particular computer processor and therefore have the value `noarch` in this field.

One important property of an RPM package which is *not* typically included in its filename is the operating system it is designed to work with. Different Unix and GNU/Linux distributions, such as SUSE and Fedora Core, may have slightly different conventions regarding how and where programs and documentation are installed. Therefore it is always important to install only those RPM packages which are meant for the OS distribution you are using.

(Usually whatever web page or FTP site you find the RPM on will indicate which distribution it is for.) This caveat is compounded by the fact that different TEX distributions may be available for the same operating system, and that these distributions may also have different conventions for how and where to install files, and may come with different default packages.

Therefore, when looking for TEX RPMs, you must ensure not only that they are specific to your operating system, but also to your TEX distribution. To help alleviate this problem, we recommend that packagers and distributors of RPMs prepend the name of the TEX distribution to the RPM package name. Thus, for example, an RPM package for the LATEX package `breakurl` for use with the teTEX distribution would have a filename such as

`tetex-breakurl-0.04-1.noarch.rpm`

However, whether this RPM is meant for SUSE, Red Hat, or some other GNU/Linux distribution must be indicated separately.

Most developers writing programs for Unix-like systems will provide RPM packages of their software on their official website. Alternatively, there exist several Internet search engines, such as `rpmfind.net`, which index RPM files. Currently TEX packages are not widely available as RPM packages, though hopefully this article will go some way towards encouraging TEX package developers and distributors to remedy the situation. In the meantime, a number of LATEX RPMs for the SUSE distribution of teTEX have been made available on the present author's home page at `http://www.nothingisreal.com/tetex`.

2.2 Installing, upgrading, and removing packages

The main utility for manipulating RPM packages is named, reasonably enough, `rpm`, and on Unix-like systems is usually located in the `/bin` directory.[3] `rpm` is a command-line utility, and we describe its use in this section. Though it is not difficult to use, most operating systems provide a graphical interface to it, so installing or upgrading a package is often as simple as clicking on the RPM file in your file explorer.

To install an RPM package you have obtained, you issue the following command (substituting the actual filename) while logged in as the superuser (root):[4]

```
# rpm --install \
gnomovision-1.2-273.i586.rpm
```

(For more verbose output and a progress meter, the `--verbose` and `--hash` options can also be specified.) On the other hand, if you haven't yet downloaded the file but know its location on the Internet, you can tell `rpm` to fetch it for you via HTTP or FTP:

```
# rpm --install ftp://ftp.foo.de/\
gnomovision-1.2-273.i586.rpm
```

`rpm` will then process the named file, make sure that all its dependencies are met and that no conflicts are caused, and install it. Once a TEX RPM is installed, no further work or setup should be needed; whatever TEX package it installed should be immediately available to your TEX installation. There should be no need to manually update TEX's filename database (*e. g.*, `texhash`).

If a certain RPM package is already installed on your system and you have downloaded a newer version, you can upgrade the existing installation as follows:

```
# rpm --upgrade \
gnomovision-1.2-273.i586.rpm
```

The `--erase` option uninstalls an RPM package you have previously installed. Note that you do not specify the complete package filename; just the name of the software is used:

```
# rpm --erase gnomovision
```

Conveniently, `rpm` will issue a warning if you try to remove a package which other installed packages require. You can then decide to remove those packages as well or abort the process.

2.3 Getting information on packages

`rpm` also provides the `--query` option for listing and getting information on installed packages. Used by itself and a package name, this option simply prints out the version and revision number of the package if it is installed. Alternatively, `--query` can be used with auxiliary options to perform various useful tasks. For example,

```
# rpm --query --info gnomovision
```

displays the details of the `gnomovision` package, including its size, packaging date, installation date, as well as its purpose and functionality. Adding the `--list` option will also show a list of each file the

[3] This article will hereinafter assume that the user is working on a GNU/Linux or Unix system; however, most of what is presented is easily applicable to other operating systems which support RPM.

[4] To fit the formatting of this journal, we sometimes break lines in shell command examples by using a backslash (\) followed by a newline. In practice you can type the commands on a single line, omitting the backslash and newline.

```
Name        : tetex-breakurl               Relocations: (not relocateable)
Version     : 0.04                              Vendor: (none)
Release     : 1                             Build Date: Wed 06 Jul 2005 04:52:35
Install date: (not installed)               Build Host: port-3108.kl.dfki.de
Group       : Productivity/Publishing/TeX/Base  Source RPM: tetex-breakurl-0.04-1.src.rpm
Size        : 131758                           License: LPPL
Signature   : (none)
Packager    : Tristan Miller <Tristan.Miller@dfki.de>
URL         : http://www.ctan.org/tex-archive/macros/latex/contrib/breakurl/
Summary     : An extension to hyperref for line-breakable urls in DVIs
Description :
This package provides a command much like hyperref's \url that
typesets a URL using a typewriter-like font.  However, if the dvips
driver is being used, the original \url doesn't allow line breaks in
the middle of the created link: the link comes in one atomic piece.
This package allows such line breaks in the generated links.

Note that this package is intended only for those using the dvips
driver.  Users of the pdflatex driver already have this feature.
Distribution: SuSE 9.0 (i586)
/usr/local/share/texmf/doc/latex/breakurl/README
/usr/local/share/texmf/doc/latex/breakurl/breakurl.dvi
/usr/local/share/texmf/doc/latex/breakurl/breakurl.pdf
/usr/local/share/texmf/tex/latex/breakurl/breakurl.sty
```

Figure 2: Output of `rpm --query --info --list --package tetex-breakurl-0.04-1.rpm`

package will install. (See Figure 2 for sample output of a more realistic package — our example in the next section, in fact.) Note that by default, the `--query` option searches only the database of installed packages. To use it on a RPM file you have downloaded, you must use it in conjunction with the `--package` option and the filename.[5] For example:

```
# rpm --query --info --package \
gnomovision-1.2-273.i586.rpm
```

Another useful command with `--query` is

```
# rpm --query --all
```

which lists all RPM packages installed on the system.

For most ordinary users, the above commands are all that is required to effectively use `rpm`. For advanced operations, consult the `rpm` man page, or use whatever graphical interface your system provides.

3 Creating RPM packages

So, you're a developer who has created a new TEX package, or perhaps you're just an ordinary user who has downloaded something from CTAN and wants to package it as an RPM. Before you can begin creating RPM packages, though, you first need to set up a few things; these need be done only once.

[5] Some pagers and file viewers, such as `less`, understand the RPM file format; using them to view RPM files will result in output similar to that of `rpm --query --info --list --package`.

3.1 First-time setup

The program used to create RPM packages is named `rpmbuild`. Before you can use it, however, you need to create a workspace for its use. You can do this with the following shell command:

```
$ mkdir -p ~/rpm/{BUILD,SOURCES,\
SPECS,SRPMS,RPMS/noarch}
```

It's OK to specify a directory other than `~/rpm` if you wish.

Next, you need to create in your home directory a configuration file named `.rpmmacros` which provides some default information to be used when building packages; Listing 1 shows a sample. The `%packager` line should specify your name and e-mail address, formatted as shown, so that people can contact you to report bugs or problems with your package. The `%_topdir` line should correspond to the workspace directory you created previously. (If you are unsure of the full path to your home directory, the `pwd` shell command can tell you what it is.)

```
%packager Tristan Miller <Tristan.Miller@dfki.de>
%_topdir /home/miller/rpm
```

Listing 1: A sample `~/.rpmmacros` file

3.2 Preparing the TEX package source

With the above one-time setup steps complete, you are now ready to begin building RPM packages. The first thing to do is to fetch the source to the TEX package for which you want to create an RPM. If you are a package developer, we assume you already have all the files; for those of you creating RPMs for others' TEX packages, you will have to download the files from the author's web page or from CTAN. Normally these will be available as a `tar.gz` or `zip` archive.

Let's assume that we are installing the package `breakurl`, which is available at `http://ctan.org/tex-archive/macros/latex/contrib/breakurl`. Follow the "get this entire directory" link, specify a mirror that supports directory archives, and download the package into a temporary directory on your machine, such as `/tmp`. Then decompress the archive using `unzip` or `tar` as appropriate:

```
$ cd /tmp
$ tar xzvf breakurl.tar.gz
breakurl/
breakurl/README
breakurl/breakurl.dtx
breakurl/breakurl.ins
breakurl/breakurl.pdf
```

3.3 Writing the `spec` file

Next you must prepare a `spec` file, which is a set of commands instructing `rpmbuild` how to compile the source files and where to install them. `spec` files are generally stored in the `SPECS` subdirectory of the workspace you created in §3.1, and are composed of a number of sections, or stanzas:

- the *Header stanza*, which defines custom macros and gives basic information about the package;
- the *Prep stanza*, which unpacks the package and prepares it for compilation;
- the *Build stanza*, which provides instructions for compiling the package;
- the *Install stanza*, which provides instructions for installing the package;
- the *Files stanza*, which lists all the files to be included in the package distribution;
- the *Scripts stanza*, which specify programs to be run before and after installation or uninstallation of the package; and
- the *Changelog stanza*, which contains a record of changes made to the RPM package.

In the following subsections we continue with our `breakurl` example by building its `spec` file, named `~/rpm/SPECS/breakurl.spec`. We show the various stanzas as they are being built; the completed `spec` file is presented at the end in Listing 9.

3.3.1 The Header stanza

The Header stanza appears, unlabelled, at the beginning of the `spec` file and typically contains two kinds of information: macro definitions, and fields containing important metadata about the package.

Macros. A number of macros are predefined by your RPM distribution. For example, the macro `%_tmppath` is predefined to some temporary directory in your file system, such as `/tmp` or `/var/tmp`. Other macros are automatically defined by `rpmbuild` as it processes the `spec` file, using information from the fields you specify. For example, `rpmbuild` assigns to the `%name` and `%version` macros the same values you specify for the `Name` and `Version` fields (see below) so that you can use these values later on in your `spec` file.

In addition to these predefined macros, you can create and use your own custom macros. A macro definition looks like this:

`%define` ⟨*macro_name*⟩ ⟨*macro_value*⟩

Once a macro has been defined, you can reference it later with the following syntax:

`%{`⟨*macro_name*⟩`}`

It is permissible to use a macro in the definition of a new macro.[6]

One useful macro we should define here is the root of our local TDS tree — that is, where new TEX packages should be installed on the system [8, §2.3].[7] The exact location of this directory varies with both your OS and TEX distribution, so you will need to consult the appropriate documentation. The teTEX distribution on SUSE 9.0, for example, uses `/usr/local/share/texmf`, so in that case we would define a macro as follows:

[6] Observant readers will note that what we entered into our `~/.rpmmacros` file in §3.1 were actually macro definitions.

[7] Why install to the local tree rather than the main `texmf` tree? Consider the case where the TEX distribution includes version 1.0 of a certain package `foo`. Say we then produce an RPM package of version 1.1 of `foo` which installs to the main `texmf` tree rather than the local tree. If the user installs this RPM and then later decides that version 1.1 is buggy and removes it, he will be unable to revert to version 1.0 without reinstalling his TEX distribution. Furthermore, if he does reinstall his TEX distribution, any other RPM packages that happened to install themselves in the root `texmf` tree will likely be overwritten. Installing new and upgraded versions of packages in the local tree avoids this problem; new TEX packages can be installed and removed while preserving older versions in the root tree. (When two versions of a package exist, most TEX distributions are configured to prefer the local-tree version over the root-tree version.)

Distribution	Group	Reference
Fedora Core	`Applications/Publishing`	[1, §13.2.2]
Mandriva	`Publishing`	[5]
PLD Linux	`Applications/Publishing/TeX`	
Red Hat	`Applications/Publishing`	[1, §13.2.2]
SUSE	`Productivity/Publishing/TeX/Base`	[7, §2.5]
Yellow Dog	`Applications/Publishing`	

Table 1: Groups for TEX packages by GNU/Linux distribution

```
%define texmf /usr/local/share/texmf
```

Fields. Fields are defined with the following syntax:

⟨*field_name*⟩: ⟨*field_value*⟩

The most commonly specified fields are as follows:

`Name` The name of the RPM package. As explained in §2.1, we recommend forming the RPM package name by combining your TEX distribution name with the name of the TEX package. For example, a `breakurl` RPM for teTEX would be called `tetex-breakurl`.

`Summary` A concise, one-line summary of the TEX package.

`Version` The version number of the TEX package. Normally this will be found in a `README` file or in the package's documentation, though in some cases you may need to examine the package source code. Sometimes a package will have a date but no formal version number; in these cases you should use the date, in the format ⟨*YYYYMMDD*⟩, as the version number.

`Release` The release number of the RPM package. Initially, this should be 1; every time you rebuild the RPM package (say, to fix an error in the `spec` file), this number should be incremented. Release numbers are specific to each version of the TEX package, so whenever you prepare a `spec` file for a new version of the same TEX package, the release number should be reset to 1.

`License` The license under which the TEX package is released. Normally this information will be found in a file named `README` or `COPYING`, or in the package documentation. Typically the value for the `License` field will be `LPPL` (LATEX Project Public License), though some packages are released other ways, such as under the GNU Public License (`GPL`) or as public domain. If the package is released under a custom or unusual license with no common abbreviation, then it's best to write here something like `Other` or `See package docs`.[8]

`Group` The category to which this package belongs. Different OS distributions have different categorization schemes, so you will need to consult your distribution's documentation. Table 1 lists the groups where TEX packages go for some common GNU/Linux distributions.

`URL` The home page of the TEX package. Normally this will be the package's location on CTAN.

`Requires` Any software or other RPM package required for this package to work. At a minimum, this field should contain the name of the TEX distribution you are using. You can also specify that a certain minimum (or exact) version of a package is required — for example:

```
Requires:  tetex >= 2.0.2
```

You can use as many `Requires` fields as there are prerequisites for your package, or you can use a single `Requires` field and separate the values with commas.

`Distribution` The name and version of the OS distribution for which this RPM is intended. This information is used by RPM search engines to properly categorize your package. It is possible to build RPMs for a distribution other than the one you are currently running, though this will require some knowledge of where it expects the local TDS tree to be rooted.

`Source` The source archive used to build the package. This basically corresponds to the `zip` or `tar.gz` file you downloaded from CTAN. However, it is generally expected that the source be archived as a `tar.bz2` file and given a standard name of the form ⟨*name*⟩`-`⟨*version*⟩`.tar.bz2`, where ⟨*name*⟩ and ⟨*version*⟩ are the name and version, respectively, of the TEX package.[9]

[8] If you are planning on making your RPM package available to the public, be sure to first check that the license allows it. Most free software licenses, including the LPPL and GPL, permit this, though some other licenses may stipulate that the software cannot be repackaged or redistributed.

[9] This is especially true if you intend to distribute "source" RPMs as well as binary RPMs — see §3.4.

```
Name: tetex-breakurl
Summary: An extension to hyperref for line-breakable urls in DVIs
Version: 0.04
Release: 1
License: LPPL
Group: Productivity/Publishing/TeX/Base
URL: http://www.ctan.org/tex-archive/macros/latex/contrib/breakurl/
Requires: tetex
Distribution: SuSE 9.0 (i586)
Source: %{name}-%{version}.tar.bz2
BuildRoot: %{_tmppath}/%{name}-%{version}-root
BuildArch: noarch
%define texmf /usr/local/share/texmf

%description
This package provides a command much like hyperref's \url that
typesets a URL using a typewriter-like font. However, if the dvips
driver is being used, the original \url doesn't allow line breaks in
the middle of the created link: the link comes in one atomic piece.
This package allows such line breaks in the generated links.

Note that this package is intended only for those using the dvips
driver. Users of the pdflatex driver already have this feature.
```

Listing 2: The Header stanza for `breakurl.spec`

Assuming we are working with version 0.04 of `breakurl`, the contents of this field would be `tetex-breakurl-0.04.tar.bz2`. If we want to use `rpmbuild`'s pregenerated macros, it would be `%{name}-%{version}.tar.bz2`. Since this file must actually be found by `rpmbuild`, you will also have to recompress and rename the archive from CTAN, and then move it into the `SOURCES` subdirectory of your workspace:

```
$ cd /tmp
$ gunzip breakurl.tar.gz
$ mv breakurl.tar \
tetex-breakurl-0.04.tar
$ bzip2 -9 tetex-breakurl-0.04.tar
$ mv tetex-breakurl-\
0.04.tar.bz2 ~/rpm/SOURCES
```

`BuildArch` The computer architecture the package is intended to run on. Since most TEX packages do not contain any binary computer code, a value of `noarch` will suffice in most cases.

`BuildRoot` A temporary directory in which to test installing the package. Normally this will be declared as `%{_tmppath}/%{name}-%{version}-root`.

`%description` A detailed description of the TEX package, possibly several paragraphs in length. Often this information can be copied from the package's `README` file. (Strictly speaking, the `%description` is not a field, since it is followed by a newline rather than a colon, and is terminated by the beginning of the Prep stanza.)

These fields can be specified in any order, except that the `%description` field must come last.

Listing 2 shows a complete Header stanza for `breakurl.spec`.

3.3.2 The Prep stanza

The Prep stanza, which always begins with the line

```
%prep
```

contains macros and/or shell commands which prepare the package for compilation. For TEX packages, this typically involves merely unpacking the source code archive (specified by the Header stanza's `Source` field) into the temporary build directory.

The RPM system provides the macro `%setup` for this purpose; it is usually used with the `-q` (quiet) option to suppress unwanted output. The `%setup` macro assumes that the source archive unpacks into a directory named ⟨*name*⟩`-`⟨*version*⟩ (the actual name and version being retrieved from the Header stanza) and will `cd` into it in preparation for the next stanza. If your tarball unpacks into a different directory, then the option `-n` ⟨*dirname*⟩ can be used to specify an alternative directory name.

A sample Prep stanza for `breakurl.spec` appears in Listing 3. Note that we specify the option `-n breakurl` to `%setup`. That's because, as we saw in §3.2, the `breakurl` tarball we downloaded from CTAN unpacked into a directory called simply `breakurl`.

```
%prep
%setup -q -n breakurl
```

Listing 3: The Prep stanza for `breakurl.spec`

3.3.3 The Build stanza

The Build stanza begins as follows:

```
%build
```

Following this token you should type whatever shell commands are necessary to build the TEX package. For most LATEX packages, this will involve running `latex` on any `ins` and `dtx` files; it may also involve running `bibtex`, `makeindex`, `dvips`, and other commands. On the other hand, some LATEX packages come prebuilt as ready-to-install `sty` and/or `cls` files; in such cases the Build stanza will be empty.

You should consult the package's build instructions to find out which commands, if any, you need to run on which files, and manually try them out yourself on the temporary copy of the package source you unpacked in §3.2. This way you can find out, for example, how many times you need to run `latex` before all references are resolved, and thus include the appropriate number of calls to it in the Build stanza.

A sample Build stanza for `breakurl.spec` appears in Listing 4. Note that the tarball already contained a PDF version of the package documentation, making our processing of `breakurl.dtx` to create the DVI documentation somewhat unnecessary. However, let's assume for illustrative purposes that we wish to include both PDF and DVI versions of the documentation in our RPM, thus necessitating the generation of the latter.

```
%build
latex breakurl.ins
latex breakurl.dtx
latex breakurl.dtx
```

Listing 4: The Build stanza for `breakurl.spec`

3.3.4 The Install stanza

After the Build stanza comes the Install stanza. Its beginning is denoted by the following token:

```
%install
```

Like the Build stanza, the Install stanza consists of shell commands, though this time the purpose is to copy the generated files into their correct places in the TEX directory structure (TDS). However, rather than installing the files into the system TDS (which can have disastrous effects if there is an error in the `spec` file), we instead simulate or "practice" installing them into a temporary copy of the file system. This temporary copy was specified by the `BuildRoot` field of the Header stanza, the value of which can be accessed here with the shell variable `$RPM_BUILD_ROOT`.

Another component of the Install stanza is a small script to remove the contents of the `BuildRoot` directory after a build. This script begins with

```
%clean
```

and usually consists of the command

```
rm -rf $RPM_BUILD_ROOT
```

Many packagers prefer to include a copy of this command as the first command after `%install`, just to make sure the `BuildRoot` is empty.

A sample Install stanza for `breakurl.spec` appears in Listing 5.

```
%install
rm -rf $RPM_BUILD_ROOT
mkdir -p %{texmf}/tex/latex/breakurl
cp breakurl.sty %{texmf}/tex/latex/breakurl
mkdir -p %{texmf}/doc/latex/breakurl
cp README %{texmf}/doc/latex/breakurl
cp breakurl.dvi %{texmf}/doc/latex/breakurl
cp breakurl.pdf %{texmf}/doc/latex/breakurl

%clean
rm -rf $RPM_BUILD_ROOT
```

Listing 5: The Install stanza for `breakurl.spec`

3.3.5 The Files stanza

The Files stanza, which begins with the line

```
%files
```

lists all the files and directories to be included in the RPM package. This is important, as during the build process a number of temporary files (*e. g.*, `aux`, `log`, `bbl`) may be created, and `rpmbuild` needs to know that these can be safely discarded.

The main part of the Files stanza is relatively simple; it simply consists of a list of files and directories, relative to the `BuildRoot` directory where they were temporarily installed in the Install stanza. Normally one file or directory per line is specified, though it is permissible to use wildcards. For example, the lines

```
%{texmf}/tex/latex/mypackage/*.sty
%{texmf}/tex/latex/mypackage/*.cls
```

specify including all the LATEX style and class files in the directory `$RPM_BUILD_ROOT/%{texmf}/tex/latex/mypackage`.

Be careful when specifying a directory name, because that indicates to `rpmbuild` that it should package *all* files in that directory. If you just want to indicate that a particular directory but none of its files should be packaged, precede the name of the directory with the `%dir` macro:

```
%dir %{texmf}/tex/latex/mypackage/tmp
```

There are other macros you can use before a filename to give important information about the file. The `%doc` macro indicates that a file is documentation. (Marking this is important because some users might want to save space by not installing the package's documentation. The `rpm --install` command has an auxiliary option, `--excludedocs`, to suppress installation of documentation.) Alternatively, the `%config` macro can be used to mark a file as being a user-modifiable configuration file. When upgrading a package, `rpm` will be careful not to overwrite any such file the user may have painstakingly modified. Here are some examples:

```
%doc %{texmf}/doc/latex/foo/guide.dvi
%doc %{texmf}/doc/latex/foo/README
%config %{texmf}/tex/latex/foo/foo.cfg
```

Finally, it is important to set the ownership and permissions for the files to be installed. This can be done collectively for all files by issuing the `%defattr` macro at the beginning of the Files stanza, which has this syntax (on two lines only for this presentation; the actual source must be all on one line):

`%defattr(`⟨*file_permissions*⟩`,`⟨*owner*⟩`,`
⟨*group*⟩`,`⟨*directory_permissions*⟩`)`

Both ⟨*owner*⟩ and ⟨*group*⟩ will normally be `root`. You should specify the file and directory permissions in the standard three-digit octal format used by `chmod`. For example, one might specify the directory permissions as `755`, which corresponds to `rwxr-xr-x` (*i. e.*, everyone can read and execute the directory, but only its owner can write to it). Instead of specifying an octal value, you can use the hyphen, `-`, to tell `rpmbuild` to package the files and directories with the same permissions it has in the `BuildRoot` tree. To override the default permissions or ownership on a particular file, you can prefix it with the `%attr` macro, which uses the same syntax as `%defattr`.

A sample Files stanza for `breakurl.spec` appears in Listing 6.

```
%files
%defattr(-,root,root,-)
%{texmf}/tex/latex/breakurl/breakurl.sty
%doc %{texmf}/doc/latex/breakurl/README
%doc %{texmf}/doc/latex/breakurl/breakurl.dvi
%doc %{texmf}/doc/latex/breakurl/breakurl.pdf
```

Listing 6: The Files stanza for `breakurl.spec`

3.3.6 The Scripts stanza

Sometimes, certain system commands need to be executed before or after software is installed or uninstalled. This is also true of (un)installing packages on most TEX distributions, usually because TEX has to update its file index. On teTEX and other TEX distributions which use the *Kpathsea* path searching library, for example, the command `texhash` or `mktexslr` must be run whenever a package is installed or uninstalled. Such commands can be specified in the Scripts stanza of the `spec` file.

Unlike most of the previous stanzas, the beginning of the Scripts stanzas is not marked by a token. Rather, scripts to be executed before installation, after installation, before uninstallation, and after uninstallation can be specified following the `%pre`, `%post`, `%preun`, and `%postun` tokens, respectively. Listing 7 shows the Script stanza for our `breakurl.spec` file.

```
%post
texhash

%postun
texhash
```

Listing 7: The Scripts stanza for `breakurl.spec`

3.3.7 The Changelog stanza

The Changelog stanza is where you should keep a human-readable log of changes to the `spec` file. This stanza begins with the line

```
%changelog
```

and contains a chronological list of entries (most recent first) in this format:

```
* ⟨date⟩ ⟨name⟩ <⟨email⟩> ⟨version⟩-⟨release⟩
- ⟨change made⟩
- ⟨other change made⟩...
```

The ⟨*date*⟩ field must be in the following format:

```
Tue Jul 05 2005
```

Such a date can be produced with the command

```
$ date +"%a %b %d %Y"
```

The initial Changelog stanza for `breakurl.spec` is illustrated in Listing 8.

```
%changelog
* Mon Jul 04 2005 Tristan Miller \
              <Tristan.Miller@dfki.de> 0.04-1
- Initial build.
```

Listing 8: The Changelog stanza for `breakurl.spec` (the \ and line break are for our presentation only)

3.4 Building the RPM

Now that you've written the `spec` and moved the source tarball (recompressed with `bzip2`) into your `~/rpm/SOURCES` directory, you are finally ready to build the RPM. Simply `cd` into the directory where `breakurl.spec` resides (`~/rpm/SPECS`) and run the following command (logged into your regular account — *not* as root!):

```
$ rpmbuild -ba breakurl.spec
```

`rpmbuild` will first scrutinize the `spec` file for any syntax errors and abort with an informative message if it finds any. If not, you should see the commands you've specified in the Build and Install stanzas being executed as if you had typed them yourself. You'll probably get several pages of LaTeX output plus various other diagnostic messages from `rpmbuild` itself. The resulting RPM package will be written to the file

`~/rpm/RPMS/noarch/tetex-breakurl-0.04-1.rpm`

Go ahead and examine the file with `less`, or with `rpm --query --info --list --package`, to make sure that all the package data is correct and that all files are set to install in the correct places. (The data should look similar to what is shown in Figure 2.) If not, you'll have to figure out where the error is, re-edit the `spec` file, increment its `Release` number, and try rebuilding.[10]

Another file produced by `rpmbuild`,

`~/rpm/SRPMS/tetex-breakurl-0.04-1.src.rpm`,

is what is known as a *source RPM* or SRPM. Unlike the RPM package, which contains only the precompiled, ready-to-install TeX package, the SRPM contains the original source tarball and your `spec` file. SRPMs are useful for users who want to modify the TeX package before it is compiled, or for those who wish to create a new RPM of the same TeX package for a different OS or TeX distribution and don't want to go to the trouble of creating their own `spec` file from scratch.

Note that `rpmbuild` does not, by default, install the RPM package; it only creates one. To actually install the RPMs you create, you need to invoke the `rpm` command as outlined in §2.2.

Once you have created and tested your RPM, it might be a good idea to save other users the trouble you've gone to by publishing it on the web or on a local FTP server. Eventually it will probably be spidered by an RPM search engine such as `rpmfind.net` so that others can find it. It is also possible that CTAN may accept submissions of RPM packages, either now or at some point in the future.

4 Advanced topics

The information presented in this article is sufficient for making basic RPMs for most TeX packages, though there are many other topics which are not addressed here. Most importantly, this article has assumed that the TeX package you are packaging is something like a font or a LaTeX package which contains no executable code. Utilities written in programming languages such as Perl, Python, or C, or those which make use Makefiles or the GNU Autotools must be handled slightly differently; though the process isn't necessarily more complicated, there are too many different cases to cover in an article of this scope. The reader is therefore referred to more general-purpose documents on RPM [1, 2, 3] for dealing with such packages.

Another topic worthy of mention is the use of cryptographic tools such as PGP and GnuPG to digitally sign and verify RPMs. RPM has built-in support for this, though because there are many more applications for digital signatures in the world of TeXing, we will reserve treatment of it for a future article on using GnuPG with TeX.

5 Bibliography

[1] Edward C. Bailey. *Maximum RPM*. Sams, August 1997.

[2] Eric Foster-Johnson. *Red Hat RPM Guide*. Red Hat Press, March 2003.

[3] Guru Labs. Creating RPMs (student version) 1.0, April 2005.

[4] Linux Standard Base Team. *Building Applications with the Linux Standard Base*. Prentice Hall, October 2004.

[5] Mandriva Linux Development Community. *MandrivaGroups*, May 2005. Revision r1.10.

[6] Christian Schenk. *MiKTeX 2.4 Manual*, February 2004. Revision 2.4.1520.

[7] SUSE Linux AG, Nuremberg. *SUSE Package Conventions*, January 2005. Revision 1.0.

[10] The tool `rpmlint` (from Mandriva) can assist in debugging RPM files.

[8] TUG Working Group on a TEX Directory Structure. A directory structure for TEX files, June 2004. Version 1.1.

◇ Tristan Miller
German Research Center for Artificial Intelligence (DFKI GmbH)
Postfach 20 80
67608 Kaiserslautern, Germany
Tristan.Miller@dfki.de
http://www.dfki.uni-kl.de/~miller/

```
Name: tetex-breakurl
Summary: An extension to hyperref for line-breakable urls in DVIs
Version: 0.04
Release: 2
License: LPPL
Group: Productivity/Publishing/TeX/Base
URL: http://www.ctan.org/tex-archive/macros/latex/contrib/breakurl/
Requires: tetex
Distribution: SuSE 9.0 (i586)
Source: %{name}-%{version}.tar.bz2
BuildRoot: %{_tmppath}/%{name}-%{version}-root
BuildArch: noarch
%define texmf /usr/local/share/texmf

%description
This package provides a command much like hyperref's \url that
typesets a URL using a typewriter-like font. However, if the dvips
driver is being used, the original \url doesn't allow line breaks in
the middle of the created link: the link comes in one atomic piece.
This package allows such line breaks in the generated links.

Note that this package is intended only for those using the dvips
driver. Users of the pdflatex driver already have this feature.


%prep
%setup -q -n breakurl

%build
latex breakurl.ins
latex breakurl.dtx
latex breakurl.dtx

%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/%{texmf}/tex/latex/breakurl
cp breakurl.sty $RPM_BUILD_ROOT/%{texmf}/tex/latex/breakurl
mkdir -p $RPM_BUILD_ROOT/%{texmf}/doc/latex/breakurl
cp README breakurl.{dvi,pdf} $RPM_BUILD_ROOT/%{texmf}/doc/latex/breakurl

%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr(-,root,root,-)
%{texmf}/tex/latex/breakurl/breakurl.sty
%doc %{texmf}/doc/latex/breakurl/README
%doc %{texmf}/doc/latex/breakurl/breakurl.dvi
%doc %{texmf}/doc/latex/breakurl/breakurl.pdf

%changelog
* Mon Jul 4 2005 Tristan Miller <psychonaut@nothingisreal.com> 0.04-1
- Initial build.
```

Listing 9: The complete `breakurl.spec` file