<http://fedoraproject.org/wiki/How_to_create_an_RPM_package>

## Introduction

This page describes in detail how to create an RPM package, and in particular, how to create a SPEC file. Unlike other RPM guides, this page explains the specifics for Fedora with links to Fedora-specific guidelines. Since it is maintained through the Fedora Wiki, it is likely to be more up-to-date than other guides. Despite the focus on Fedora, most of this document does apply to other RPM-based distributions. If you're impatient, you might start by looking at the shorter [How to create a GNU Hello RPM package](http://fedoraproject.org/wiki/How_to_create_a_GNU_Hello_RPM_package).

Please note that these are *not* the official package guidelines for Fedora, which can be viewed in the [Packaging Guidelines](http://fedoraproject.org/wiki/Packaging:Guidelines) and [Package Naming Guidelines](http://fedoraproject.org/wiki/Packaging:NamingGuidelines). Having said that, this page should be compatible with them.

If you plan to create an RPM package for the Fedora repository, follow the procedure for [joining the package collection maintainers](http://fedoraproject.org/wiki/Join_the_package_collection_maintainers).

## Preparing your system

Before you create RPM packages on Fedora, you need to install some core development tools and set up the account(s) you will use:

# yum install @development-tools

# yum install fedora-packager

You can create a dummy user specifically for creating RPM packages so that a build process gone wrong can't trash your files or send your private keys to the world.

[Stop (medium size).png](http://fedoraproject.org/wiki/File:Stop_%28medium_size%29.png)

**You should *never* create your packages as the root user.**

Create a new user named makerpm, add the user to the 'mock' group, set a password, and login as that user:

# /usr/sbin/useradd makerpm

# usermod -a -G mock makerpm

# passwd makerpm

Once you're logged in as the build/dummy user, create the required directory structure in your home directory by executing:

$ rpmdev-setuptree

The rpmdev-setuptree program will create the ~/rpmbuild directory and a set of subdirectories (e.g. SPECS and BUILD), which you will use for creating your packages. The ~/.rpmmacros file is also created, which can be used for setting various options.

[The packaging guidelines recommend preserving file timestamps](http://fedoraproject.org/wiki/Packaging:Guidelines#Timestamps); you can make this automatic if you use wget or curl to get the source files. If you use wget to get source files, add the text "timestamping = on" to ~/.wgetrc. If you use curl, add the text "-R" to ~/.curlrc.

You won't normally need to do these steps again.

## The basics of building RPM packages

To create an RPM package, you will need to create a ".spec" text file that provides information about the software being packaged. You then run the rpmbuild command on the SPEC file, which will go through a series of steps to produce your packages.

Normally, you should place your original (pristine) sources, such as .tar.gz files from the original developers, into the ~/rpmbuild/SOURCES directory. Place your .spec file in the ~/rpmbuild/SPECS directory and name it "*NAME*.spec", where *NAME* is the base name of the package. To create both binary and source packages, change directory to ~/rpmbuild/SPECS and run:

$ rpmbuild -ba *NAME*.spec

When invoked this way, rpmbuild will read the .spec file and go through in order the stages listed below. Names beginning with % are predefined macros (see the next table down).

|  |  |  |  |
| --- | --- | --- | --- |
| **Stage** | **Reads** | **Writes** | **Action** |
| %prep | %\_sourcedir | %\_builddir | This reads the sources and patches in the source directory %\_sourcedir. It unpackages the sources to a subdirectory underneath the build directory %\_builddir and applies the patches. |
| %build | %\_builddir | %\_builddir | This compiles the files underneath the build directory %\_builddir. This is often implemented by running some variation of "./configure && make". |
| %check | %\_builddir | %\_builddir | Check that the software works properly. This is often implemented by running some variation of "make test". Many packages don't implement this stage. |
| %install | %\_builddir | %\_buildrootdir | This reads the files underneath the build directory %\_builddir and writes to a directory underneath the build root directory %\_buildrootdir. The files that are written are the files that are supposed to be installed when the binary package is installed by an end-user. Beware of the weird terminology: The *build root directory* is **not** the same as the *build directory*. This is often implemented by running "make install". |
| bin | %\_buildrootdir | %\_rpmdir | This reads the files underneath the build root directory %\_buildrootdir to create binary RPM packages underneath the RPM directory %\_rpmdir. Inside the RPM directory is a directory for each architecture, and a "noarch" directory for packages that apply to any architecture. These RPM files are the packages for users to install. |
| src | %\_sourcedir | %\_srcrpmdir | This creates a source RPM package (.src.rpm) inside the source RPM directory %\_srcrpmdir. These files are needed for reviewing and updating packages. |

As you can tell, certain directories have certain purposes in rpmbuild. These are:

|  |  |  |  |
| --- | --- | --- | --- |
| **Macro Name** | **Name** | **Usually** | **Purpose** |
| %\_specdir | Specification directory | ~/rpmbuild/SPECS | RPM specifications (.spec) files |
| %\_sourcedir | Source directory | ~/rpmbuild/SOURCES | Pristine source package (e.g. tarballs) and patches |
| %\_builddir | Build directory | ~/rpmbuild/BUILD | Source files are unpacked and compiled in a subdirectory underneath this. |
| %\_buildrootdir | Build root directory | ~/rpmbuild/BUILDROOT | Files are installed under here during the %install stage. |
| %\_rpmdir | Binary RPM directory | ~/rpmbuild/RPMS | Binary RPMs are created and stored under here. |
| %\_srcrpmdir | Source RPM directory | ~/rpmbuild/SRPMS | Source RPMs are created and stored here. |

If a stage fails, look at the output to see *why* it falied and change the .spec file (or other input) as needed.

## Getting ready to package a particular program

If there are special programs that are required to build or run the program you are packaging, install those other programs and write down what they are.

To package a program for the Fedora repository, you must package pristine (original) sources, along with the patches and build instructions; it's **not** okay to start with pre-compiled code. Install the file with the original source (usually a .tar.gz file) in the ~/rpmbuild/SOURCES directory (of the RPM building user account).

Read through the manual installation instructions for your program. It's often a good idea to do a "dry run" by manually building the program before attempting to do so via RPM. With a few exceptions, all binaries and libraries included in Fedora packages must be built from the source code that is included in the source package.

### Split up the program

Application source code is often released with the source code of other external libraries "bundled" into them. [Do not bundle external libraries with the main application into a single package](http://fedoraproject.org/wiki/Packaging:No_Bundled_Libraries). Instead, split them up into separate packages.

### Licensing

Only package software that is legal for you to package. See [Packaging:Guidelines#Legal](http://fedoraproject.org/wiki/Packaging:Guidelines#Legal), [Licensing:Main](http://fedoraproject.org/wiki/Licensing:Main) and [Packaging:LicensingGuidelines](http://fedoraproject.org/wiki/Packaging:LicensingGuidelines). In general, only package software that is released as open source software (OSS) using an approved OSS license (such as the GPL, LGPL, BSD-new, MIT/X, or Apache 2.0 licenses). Check to make sure that the software really is licensed this way (e.g. spot-check source code headers, README files etc.). If there are bundled libraries, make sure they are also OSS.

### Reuse existing package information

Try to reuse what you can. Obviously, make sure you aren't packaging something that is already packaged. You can find a list of existing packages in Fedora Package Collection in the [Fedora Package Database](https://admin.fedoraproject.org/pkgdb/). Also check the [In Progress Review Requests](http://fedoraproject.org/wiki/PackageMaintainers/InProgressReviewRequests) and the [Retired Packages](http://fedoraproject.org/wiki/PackageMaintainers/RetiredPackages) list. You can use [Fedora Packages Git Repositories](http://pkgs.fedoraproject.org/cgit) directly to view SPEC files (and patches). You can download the SRPMS using a program from the yum-utils package:

$ yum -y install yum-utils

$ yumdownloader --source sourcepackage-name

Alternatively, get the source manually from the http/ftp page of a [Fedora mirror](http://mirrors.fedoraproject.org/publiclist) within the releases/17/Everything/source/SRPMS directory. Replace "17" with the Fedora release you want and download the .src.rpm package you want.

Once you have the SRPM, install it into ~/rpmbuild:

$ rpm -ivh sourcepackage-name\*.src.rpm

You can also unpack the SRPM into a directory using rpm2cpio:

$ mkdir PROGRAMNAME\_src\_rpm

$ cd PROGRAMNAME\_src\_rpm

$ rpm2cpio ../PROGRAMNAME-\*.src.rpm | cpio -i

Sometimes it's easiest to start with an existing package and then clean it up for Fedora. [RPM Find](http://rpmfind.net/) may help you find RPMs for non-Fedora systems. You can install SRPMS for other systems the same way as for Fedora. Failing that, you might look at the source package files (not binary .deb) for [Ubuntu](http://packages.ubuntu.com/) or [Debian](http://www.debian.org/distrib/packages) (source package files are standard tarballs with a "debian/" subdirectory). If the [FreeBSD ports collection](http://www.freebsd.org/ports/installing.html) has it, you could [download the FreeBSD ports tarball](ftp://ftp.freebsd.org/pub/FreeBSD/ports/ports/ports.tar.gz) and see if their packaging information helps as a starting point. However, this is sometimes not helpful at all. Different distributions have different rules, and what they do may be quite inappropriate for Fedora.

## Creating a SPEC file

You now need to create a SPEC file in the ~/rpmbuild/SPECS directory. You should name it after the program name (e.g. "program.spec"). Use the archive name or the name advocated by the software author where you can, but be sure to follow the [Package Naming Guidelines](http://fedoraproject.org/wiki/Packaging/NamingGuidelines).

### SPEC templates

When you're creating a SPEC file for the first time, vim or emacs will automatically create a template for you:

$ cd ~/rpmbuild/SPECS

$ vim program.spec

Here's an example of what that template will look like (**Note:** the provided template may not necessarily comply with Fedora Packaging Guidelines):

Name:

Version:

Release: 1%{?dist}

Summary:

Group:

License:

URL:

Source0:

BuildRoot: %{\_tmppath}/%{name}-%{version}-%{release}-root-%(%{\_\_id\_u} -n)

BuildRequires:

Requires:

%description

%prep

%setup -q

%build

%configure

make %{?\_smp\_mflags}

%install

rm -rf %{buildroot}

make install DESTDIR=%{buildroot}

%clean

rm -rf %{buildroot}

%files

%defattr(-,root,root,-)

%doc

%changelog

You can use $RPM\_BUILD\_ROOT instead of %{buildroot}. Both are acceptable, but just be consistent.

You may also use the rpmdev-newspec command to create a SPEC file for you. rpmdev-newspec NAME-OF-NEW-PACKAGE can create an initial SPEC file for a new package, tailored to various types of packages. It will guess what kind of template to use based on the package name, or you can specify a particular template. See /etc/rpmdevtools/spectemplate-\*.spec for available templates, and see rpmdev-newspec --help for more information. For example, to create a new SPEC file for a python module:

cd ~/rpmbuild/SPECS

rpmdev-newspec python-antigravity

vi python-antigravity.spec

### SPEC example

Here's a simple example showing a Fedora 16 SPEC file for the eject program:

Summary: A program that ejects removable media using software control

Name: eject

Version: 2.1.5

Release: 21%{?dist}

License: GPLv2+

Group: System Environment/Base

Source:  %{name}-%{version}.tar.gz

Patch1: eject-2.1.1-verbose.patch

Patch2: eject-timeout.patch

Patch3: eject-2.1.5-opendevice.patch

Patch4: eject-2.1.5-spaces.patch

Patch5: eject-2.1.5-lock.patch

Patch6: eject-2.1.5-umount.patch

URL: http://www.pobox.com/~tranter

ExcludeArch: s390 s390x

BuildRequires: gettext

BuildRequires: libtool

%description

The eject program allows the user to eject removable media (typically

CD-ROMs, floppy disks or Iomega Jaz or Zip disks) using software

control. Eject can also control some multi-disk CD changers and even

some devices' auto-eject features.

Install eject if you'd like to eject removable media using software

control.

%prep

%setup -q -n %{name}

%patch1 -p1

%patch2 -p1

%patch3 -p1

%patch4 -p1

%patch5 -p1

%patch6 -p1

%build

%configure

make %{?\_smp\_mflags}

%install

make DESTDIR=%{buildroot} install

install -m 755 -d %{buildroot}/%{\_sbindir}

ln -s ../bin/eject %{buildroot}/%{\_sbindir}

%find\_lang %{name}

%files -f %{name}.lang

%doc README TODO COPYING ChangeLog

%{\_bindir}/\*

%{\_sbindir}/\*

%{\_mandir}/man1/\*

%changelog

\* Tue Feb 08 2011 Fedora Release Engineering <rel-eng@lists.fedoraproject.org> - 2.1.5-21

- Rebuilt for https://fedoraproject.org/wiki/Fedora\_15\_Mass\_Rebuild

\* Fri Jul 02 2010 Kamil Dudka <kdudka@redhat.com> 2.1.5-20

- handle multi-partition devices with spaces in mount points properly (#608502)

## SPEC file overview

Other useful guides:

* [RPM Guide](http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch-creating-rpms.html) describes how to write a SPEC file.
* The IBM series "Packaging software with RPM" [Part 1](http://www.ibm.com/developerworks/library/l-rpm1/), [Part 2](http://www.ibm.com/developerworks/library/l-rpm2/), and [Part 3](http://www.ibm.com/developerworks/library/l-rpm3/).
* [Maximum RPM](http://rpm.org/max-rpm-snapshot/) has the most complete information, but is dated.

You will need to follow the Fedora guidelines: [Package Naming Guidelines](http://fedoraproject.org/wiki/Packaging/NamingGuidelines), [Packaging guidelines](http://fedoraproject.org/wiki/Packaging/Guidelines), and [Package review guidelines](http://fedoraproject.org/wiki/Packaging/ReviewGuidelines).

Insert comments with a leading "#" character, but avoid macros (beginning with %) that are potentially multiline (as they are expanded first). If commenting out a line, double the percent signs (%%). Also avoid inline comments on the same line as a script command.

The major tags are listed below. Note that the macros %{name}, %{version} and %{release} can be used to refer to the Name, Version and Release tags respectively. When you change the tag, the macros automatically update to use the new value.

* **Name**: The (base) name of the package, which should match the SPEC file name. It must follow the [Package Naming Guidelines](http://fedoraproject.org/wiki/Packaging/NamingGuidelines) and generally be lowercase.
* **Version**: The upstream version number. See [Version tag section](http://fedoraproject.org/wiki/Packaging/NamingGuidelines#Version_Tag) of the packaging guidelines. If the version contains tags that are non-numeric (contains tags that are not numbers), you may need to include the additional non-numeric characters in the Release tag. If upstream uses full dates to distinguish versions, consider using version numbers of the form yy.mm[dd] (e.g. 2008-05-01 becomes 8.05).
* **Release**: The initial value should normally be 1%{?dist}. Increment the number every time you release a new package for the same version of software. When a new upstream version is released, change the Version tag to match and reset the Release number to 1. See [Release tag section](http://fedoraproject.org/wiki/Packaging/NamingGuidelines#Release_Tag) of the packaging guidelines. The optional [Dist tag](http://fedoraproject.org/wiki/Packaging/DistTag) might be useful.
* **Summary**: A brief, one-line summary of the package. Use American English. Do **not** end in a period.
* **Group**: This needs to be a pre-existing group, like "Applications/Engineering"; run "less /usr/share/doc/rpm-\*/GROUPS" to see the complete list. Use the group "Documentation" for any sub-packages (e.g. kernel-doc) containing documentation. ***Note: This tag is deprecated since Fedora 17. See [***[***File Reference Preamble***](http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/Packagers_Guide/chap-Packagers_Guide-Spec_File_Reference-Preamble.html%7CSpec)***]***
* **License**: The license, which must be an open source software license. Do *not* use the old Copyright tag. Use a standard abbreviation (e.g. "GPLv2+") and be specific (e.g. use "GPLv2+" for GPL version 2 or greater instead of just "GPL" or "GPLv2" where it's true). See [Licensing](http://fedoraproject.org/wiki/Licensing) and the [Licensing Guidelines](http://fedoraproject.org/wiki/Packaging/LicensingGuidelines). You can list multiple licenses by combining them with "and" and "or" (e.g. "GPLv2 and BSD").
* **URL**: The full URL for more information about the program (e.g. the project website). ***Note: This is not where the original source code came from which is meant for the Source0 tag below***.
* **Source0**: The full URL for the compressed archive containing the (original) pristine source code, as upstream released it. "Source" is synonymous with "Source0". If you give a full URL (and you should), its basename will be used when looking in the SOURCES directory. If possible, embed %{name} and %{version}, so that changes to either will go to the right place. [Preserve timestamps](http://fedoraproject.org/wiki/Packaging:Guidelines#Timestamps) when downloading source files. If there is more than one source, name them Source1, Source2 and so on. If you're adding whole new files in addition to the pristine sources, list them as sources *after* the pristine sources. A copy of each of these sources will be included in any SRPM you create, unless you specifically direct otherwise. See [Source URL](http://fedoraproject.org/wiki/Packaging/SourceURL) for more information on special cases (e.g. revision control).
* **Patch0**: The name of the first patch to apply to the source code. If you need to patch the files after they've been uncompressed, you should edit the files and save their differences as a "patch" file in your ~/rpmbuild/SOURCES directory. Patches should make only one logical change each, so it's quite possible to have multiple patch files.
* **BuildArch**: If you're packaging files that are architecture-independent (e.g. shell scripts, data files), then add "BuildArch: noarch". The architecture for the binary RPM will then be "noarch".
* **BuildRoot**: This is where files will be "installed" during the %install process (after the %build process). This is now redundant in Fedora and is only needed for EPEL5. By default, the build root is placed in "%{\_topdir}/BUILDROOT/".
* **BuildRequires**: A comma-separated list of packages required for building (compiling) the program. This field can be (and is commonly) repeated on multiple lines. These dependencies are *not* automatically determined, so you need to include *everything* needed to build the program. [Some common packages can be omitted](http://fedoraproject.org/wiki/Packaging/Guidelines#BuildRequires), such as gcc. You can specify a minimum version if necessary (e.g. "ocaml >= 3.08"). If you need the file /EGGS, determine the package that owns it by running "rpm -qf /EGGS". If you need the program EGGS, determine the package that owns it by running "rpm -qf `which EGGS`". Keep dependencies to a minimum (e.g. use sed instead of perl if you don't really need perl's abilities), but beware that some applications permanently disable functions if the associated dependency is not present; in those cases you may need to include the additional packages. The [Package-x-generic-16.png](http://fedoraproject.org/wiki/File:Package-x-generic-16.png)[auto-buildrequires](https://admin.fedoraproject.org/pkgdb/acls/name/auto-buildrequires) package may be helpful.
* **Requires**: A comma-separate list of packages that are required when the program is installed. Note that the BuildRequires tag lists what is required to build the binary RPM, while the Requires tag lists what is required when installing/running the program; a package may be in one list or in both. In many cases, rpmbuild automatically detects dependencies so the Requires tag is not always necessary. However, you may wish to highlight some specific packages as being required, or they may not be automatically detected.
* **%description**: A longer, multi-line description of the program. Use American English. All lines must be 80 characters or less. Blank lines indicate a new paragraph. Some graphical user interface installation programs will reformat paragraphs; lines that start with whitespace will be treated as preformatted text and displayed as is, normally with a fixed-width font. See [RPM Guide](http://docs.fedoraproject.org/drafts/rpm-guide-en/ch09s03.html).
* **%prep**: Script commands to "prepare" the program (e.g. to uncompress it) so that it will be ready for building. Typically this is just "%setup -q"; a common variation is "%setup -q -n NAME" if the source file unpacks into NAME. See the %prep section below for more.
* **%build**: Script commands to "build" the program (e.g. to compile it) and get it ready for installing. The program should come with instructions on how to do this. See the %build section below for more.
* **%check**: Script commands to "test" the program. This is run between the %build and %install procedures, so place it there if you have this section. Often it simply contains "make test" or "make check". This is separated from %build so that people can skip the self-test if they desire.
* **%install**: Script commands to "install" the program. The commands should copy the files from the BUILD directory %{\_builddir} into the buildroot directory, %{buildroot}. See the %install section below for more.
* **%clean**: Instructions to clean out the build root. Note that this section is now redundant in Fedora and is only necessary for EPEL. Typically this contains only:

rm -rf %{buildroot}

* **%files**: The list of files that will be installed. See the %files section below for more.
* **%changelog**: Changes in the package. Use the format example above.
* **ExcludeArch**: If the package does not successfully compile, build or work on a particular architecture, list those architectures under this tag.
* You can add sections so that code will run when packages are installed or removed on the real system (as opposed to just running the %install script, which only does a pseudo-install to the build root). These are called "scriptlets", and they are usually used to update the running system with information from the package. See the "Scriptlets" section below for more.

RPM also supports the creation of several packages (called [subpackages](http://fedoraproject.org/wiki/How_to_create_an_RPM_package#Subpackages)) from a single SPEC file, such as name-libs and name-devel packages.

Do **not** use these tags:

* Packager
* Vendor
* Copyright

Do **not** create a "relocatable" package; they don't add value in Fedora and make things more complicated.

## SPEC file sections explained

### %prep section

The %prep section describes how to unpack the compressed packages so that they can be built. Typically, this includes the "%setup" and "%patch" commands with reference to the Source0 (and Source1 etc.) lines. See the [Maximum RPM section on %setup and %patch](http://rpm.org/max-rpm-snapshot/s1-rpm-inside-macros.html) for more details.

The %{patches} and %{sources} macros are available since RPM 4.4.2 and are useful if you have a large list of patches or sources:

for p in %{patches}; do

...

done

However, keep in mind that using these will make your SPEC incompatible with RPMS used in RHEL and other RPM-based dirstributions.

#### %prep section: %setup command

The "%setup" command unpacks a source package. Switches include:

* **-q** : Suppress unecessary output. This is commonly used.
* **-n *name*** : If the Source tarball unpacks into a directory whose name is not the RPM name, this switch can be used to specify the correct directory name. For example, if the tarball unpacks into the directory FOO, use "%setup -q -n FOO".
* **-c *name*** : If the Source tarball unpacks into multiple directories instead of a single directory, this switch can be used to create a directory named *name* and then unpack into it.

There are [more %spec options if you are unpacking multiple files](http://rpm.org/max-rpm-snapshot/s1-rpm-inside-macros.html), which is primarily useful if you are creating subpackages (see below). The key ones are:

|  |  |
| --- | --- |
| -a number | Only unpack the Source directive of the given number after changing directory (e.g. "–a 0" for Source0). |
| -b number | Only unpack the Source directive of the given number before changing directory (e.g. "–b 0" for Source0). |
| -D | Do not delete the directory before unpacking. |
| -T | Disable the automatic unpacking of the archives. |

#### %prep section: %patch commands

The "%patch0" command applies Patch0 (and %patch1 applies Patch1 etc.). Patches are the normal method of making necessary changes to the source code for packaging. The usual "-pNUMBER" option applies, which passes that argument onto the program patch.

Patch file names often look like "telnet-0.17-env.patch", which is the format %{name} - %{version} - REASON.patch" (though sometimes version is omitted). Patch files are typically the result of "diff -u"; if you do this from the subdirectory of ~/rpmbuild/BUILD then you won't have to specify a -p level later.

This is a typical procedure for creating a patch for a single file:

cp foo/bar foo/bar.orig

vim foo/bar

diff -u foo/bar.orig foo/bar > ~/rpmbuild/SOURCES/PKGNAME.REASON.patch

If editing many files, one easy method is to copy the whole subdirectory underneath BUILD and then do subdirectory diffs. After you have changed directory to "~rpmbuild/BUILD/NAME", do the following:

cp -pr ./ ../PACKAGENAME.orig/

... many edits ...

diff -u ../PACKAGENAME.orig . > ~/rpmbuild/SOURCES/*NAME*.*REASON*.patch

If you edit many files in one patch, you can also copy the original files using some consistent ending such as ".orig" before editing them. Then, you can use "gendiff" (in the rpm-build package) to create a patch with the differences.

Try to ensure that your patch match the context exactly. The default "fuzz" value is "0", requiring matches to be exact. You can work around this by adding "%global \_default\_patch\_fuzz 2" to revert to the value found in older versions of RPM in Fedora, but it is generally recommended to avoid doing this.

As explained in [Packaging/PatchUpstreamStatus](http://fedoraproject.org/wiki/Packaging/PatchUpstreamStatus), all patches should have a comment above them in the SPEC file about their upstream status. This should document the upstream bug/email that includes it (including the date). If it is unique to Fedora, you should mention why it is unique. The Fedora Project tries not to deviate from upstream; see [PackageMaintainers/WhyUpstream](http://fedoraproject.org/wiki/PackageMaintainers/WhyUpstream) for the importance of this.

#### %prep section: Unmodified files

Sometimes, one or more of the Source files do not need to be uncompressed. You can "prep" those into the build directory like this (where SOURCE1 refers to the relevant Source file):

cp -p %SOURCE1 .

### %build section

The "%build" section is sometimes complicated; here you configure and compile/build the files to be installed.

Many programs follow the GNU configure approach (or some variation). By default, they will install to a prefix of "/usr/local", which is reasonable for unpackaged files. However, since you are packaging it, change the prefix to "/usr". Libraries should be installed to either /usr/lib or /usr/lib64 depending on the architecture.

Since GNU configure is so common, the macro "%configure" can be used to automatically invoke the correct options (e.g. change the prefix to /usr). Some variation of this often works:

 %configure

make %{?\_smp\_mflags}

To override makefile variables, pass them as parameters to make:

make %{?\_smp\_mflags} CFLAGS="%{optflags}" BINDIR=%{\_bindir}

More more information, see ["GNU autoconf, automake, and libtool"](http://sourceware.org/autobook/) and ["Open Source Development Tools: An Introduction to Make, Configure, Automake, Autoconf" by Stefan Hundhammer](http://www.suse.de/%7Esh/automake/automake.pdf).

Some programs use cmake. See [Packaging/cmake](http://fedoraproject.org/wiki/Packaging/cmake).

### %check section

If self-tests are available, it is generally a good idea to include them. They should be placed in the %check section (which immediately follows the %build section) instead of within the %build section itself, so that they can be easily skipped when necessary.

Often, this section contains:

make test

### %install section

This section involves script commands to "install" the program, copying the relevant files from %{\_builddir} to %{buildroot} (which usually means from ~/rpmbuild/BUILD to ~/rpmbuild/BUILDROOT) and creating directories inside %{buildroot} as necessary.

Some of the terminology can be misleading:

* The "build directory", also known as %{\_builddir} is not the same as the "build root", also known as %{buildroot}. Compilation occurs in the former directory, while files to be packaged are copied from the former to the latter.
* During the %build section, the current directory will start at %{buildsubdir}, which is the subdirectory within %{\_builddir} that was created during %prep stage. This is usually something like ~/rpmbuild/BUILD/%{name}-%{version}.
* The %install section is **not** run when the binary RPM package is installed by the end-user, but is only run when creating a package.

Normally, some variation of "make install" is performed here:

%install

rm -rf %{buildroot}

make DESTDIR=%{buildroot} install

Removal of %{buildroot} is no longer necessary, except for EPEL 5.

Ideally you should use [DESTDIR=%{buildroot}](http://www.gnu.org/prep/standards/html_node/DESTDIR.html) if the program supports it, as it redirects file installations to the specified directory and is exactly what we want to happen during the %install section.

If the program does not support DESTDIR (and only if), you can workaround it in one of several (inferior) ways:

* Patch the makefile so that is supports DESTDIR. Create directories inside DESTDIR where necessary and submit the patch upstream.
* Use the "%makeinstall" macro. This method might work, but can lead to subtle failures. It expands to something like "make prefix=%{buildroot}%{\_prefix} bindir=%{buildroot}%{\_bindir} ... install", which can result in some programs failing to work properly. Create directories inside %{buildroot} where necessary.
* Consider using the auto-destdir package. This requires "BuildRequires: auto-destdir", and changing "make install" to "make-redir DESTDIR=%{buildroot} install". This only works well if the installation uses only certain common commands to install files, like cp and install.
* Perform the installation by hand. This would involve creating the necessary directories under %{buildroot} and copying files from %{\_builddir} to %{buildroot}. Be especially careful with updates, which often contain new or changed filenames. An example of this procedure:

%install

rm -rf %{buildroot}

mkdir -p %{buildroot}%{\_bindir}/

cp -p mycommand %{buildroot}%{\_bindir}/

As noted in [Packaging:Guidelines#Timestamps](http://fedoraproject.org/wiki/Packaging:Guidelines#Timestamps), try to preserve timestamps if the makefile lets you override commands:

make INSTALL="install -p" CP="cp -p" DESTDIR=%{buildroot} install

### %files section

This section declares which files and directories are owned by the package, and thus which files and directories will be placed into the binary RPM.

#### %files basics

The %defattr set the default file permissions, and is often found at the start of the %files section. Note that this is no longer necessary unless the permissions need to be altered. The format of this is:

%defattr(<file permissions>, <user>, <group>, <directory permissions>)

The fourth parameter is often omitted. Usually one uses %defattr(-,root,root,-), where "-" uses the default permissions.

You should then list all the files and directories to be owned by the package. Use macros for directory names where possible, which can be viewed at [Packaging:RPMMacros](http://fedoraproject.org/wiki/Packaging:RPMMacros) (e.g. use %{\_bindir}/mycommand instead of /usr/bin/mycommand). If the pattern begins with a "/" (or when expanded from the macro) then it is taken from the %{buildroot} directory. Otherwise, the file is presumed to be in the current directory (e.g. inside %{\_builddir}, such as documentation files that you wish to include). If your package only installs a single file /usr/sbin/mycommand, then the %files section can simply be:

%files

%{\_sbindir}/mycommand

To make your package less sensitive to upstream changes, declare all files within a directory to be owned by the package with a pattern match:

%{\_bindir}/\*

To include a single directory:

%{\_datadir}/%{name}/

Note that %{\_bindir}/\* does not claim that this package owns the /usr/bin directory, but only the files contained within. If you list a directory, then you are claiming that the package owns that directory and all files and subdirectories contained within. Thus, do **not** list %{\_bindir} and be careful of directories that may be shared with other packages.

An error will occur if:

* a pattern match does not match any file or directory
* a file or directory is listed or matched more than once
* a file or directory in the %{buildroot} has not been listed

It is also possible to exclude files from a previous match by using the %exclude glob. This can be useful for including almost all of the files included by a different pattern match, but note that it will also fail if it does not match anything.

#### %files prefixes

You may need to add one or more prefixes to lines in the %files section; seperate them with a space. See [Max RPM section on %files directives](http://www.rpm.org/max-rpm/s1-rpm-inside-files-list-directives.html).

Usually, "%doc" is used to list documentation files within %{\_builddir} that were not copied to %{buildroot}. A README and INSTALL file is usually included. They will be placed in the directory /usr/share/doc/%{name}-%{version}, whose ownership does not need to be declared.

**Note:** If specifying a %doc entry, then you can't copy files into the documentation directory during the %install section. If, for example, you want an "examples" subdirectory within the documentation directory, don't use %doc, but instead create the directories and copy files over manually into %{buildroot}%{\_defaultdocdir}/%{name}-%{version} during the %install section. They will be correctly marked as documentation. Make sure you include %{\_defaultdocdir}/%{name}-%{version}/ as an entry in the %files section.

Configuration files should be placed in /etc and are normally specified like this (which makes sure user changes aren't overwritten on update):

%config(noreplace) %{\_sysconfdir}/foo.conf

If the update uses a non-backwards-compatible configuration format, then specify them like this:

%config %{\_sysconfdir}/foo.conf

"%attr(mode, user, group)" can be used for finer control over permissions, where a "-" means use the default:

%attr(0644, root, root) FOO.BAR

If a file is in particular natural language, use %lang to note that:

%lang(de) %{\_datadir}/locale/de/LC\_MESSAGES/tcsh\*

Programs using Locale files should follow the [recommended method of handling i18n files](http://fedoraproject.org/wiki/Packaging:Guidelines#Handling_Locale_Files):

* find the filenames in the %install step:  %find\_lang ${name}
* add the required build dependencies: BuildRequires: gettext
* use the found filenames: %files -f ${name}.lang

These prefixes are **not** valid in Fedora: %license and %readme.

#### %files and Filesystem Hierarchy Standard (FHS)

You should follow the [Filesystem Hierarchy Standard (FHS)](http://www.pathname.com/fhs/). Executables go in /usr/bin, global configuration files go in /etc, libraries go into /usr/lib (or /usr/lib64) and so on. There is one exception: executables that should not normally be executed directly by users or administrators should go in a subdirectory of /usr/libexec, which is referred to as %{\_libexecdir}/%{name}.

Do **not** install files into /opt or /usr/local.

Unfortunately, many programs do not follow the FHS by default. In particular, architecture-independent libraries get placed in /usr/lib instead of /usr/share. The former is for architecture-dependent libraries, while the latter is for architecture-independent libraries, which means that systems with different CPU architectures can share /usr/share. There are many exceptions in Fedora (such as Python and Perl), but Fedora applies this rule more strictly than some distributions. rpmlint will generally complain if you put anything other than ELF files into /usr/lib.

#### %files example

Here's a simple example of a %files section:

%files

%doc README LICENSE

%{\_bindir}/\*

%{\_sbindir}/\*

%{\_datadir}/%{name}/

%config(noreplace) %{\_sysconfdir}/\*.conf

#### Finding duplicates

You can list any duplicates of two binary packages by doing:

cd ~/rpmbuild/RPMS/ARCH # Substitute "ARCH" for your architecture

rpm -qlp PACKAGE1.\*.rpm | sort > ,1

rpm -qlp PACKAGE2.\*.rpm | sort > ,2

comm -12 ,1 ,2

### Scriptlets

When an end-user installs the RPM, you may want some commands to be run. This can be achieved through scriptlets. See [Packaging/ScriptletSnippets](http://fedoraproject.org/wiki/Packaging/ScriptletSnippets).

Scriptlets can be run:

* before (**%pre**) or after (**%post**) a package is installed
* before (**%preun**) or after (**%postun**) a package is uninstalled
* at the start (**%pretrans**) or end (**%posttrans**) of a transaction

For example, every binary RPM package that stores shared library files in any of the dynamic linker's default paths, must call ldconfig in %post and %postun. If the package has multiple subpackages with libraries, each subpackage should also perform the same actions.

%post -p /sbin/ldconfig

%postun -p /sbin/ldconfig

If only running a single command, then the "-p" option runs the adjacent command without invoking the shell. However, for several commands, omit this option and include the shell commands beneath.

If you run any programs within the scriptlets, then you must specify any requirements in the form "Requires(CONTEXT)" (e.g. Requires(post)).

%pre, %post, %preun, and %postun provide the argument $1, which is the number of packages of this name which will be left on the system when the action completes. Don't compare for equality with 2, but instead check if they are greater than or equal to 2. For %pretrans and %posttrans, $1 is always 0.

For example, if the package installs an info manual, then the info manual index must be updated with install-info from the info package. Firstly, there is no guarantee that the info package will be available unless we explicitly declare it as required, and secondly, we don't want to fail completely if install-info fails:

Requires(post): info

Requires(preun): info

...

%post

/sbin/install-info %{\_infodir}/%{name}.info %{\_infodir}/dir || :

%preun

if [ $1 = 0 ] ; then

/sbin/install-info --delete %{\_infodir}/%{name}.info %{\_infodir}/dir || :

fi

There is one other glitch related to installing info manuals. The install-info command will update the info directory, so we should delete the useless empty directory from the %{buildroot} during the %install section:

rm -f %{buildroot}%{\_infodir}/dir

Another scriptlet-like abilility are "triggers", which can be run for your package when other packages are installed or uninstalled. See [RPM Triggers](http://rpm.org/api/4.4.2.2/triggers.html).

### Macros

Macros are text in the format %{string}. Typical macros:

|  |  |  |
| --- | --- | --- |
| **Macro** | **Typical Expansion** | **Meaning** |
| %{\_bindir} | /usr/bin | Binary directory: where executables are usually stored. |
| %{\_builddir} | ~/rpmbuild/BUILD | Build directory: files are compiled within a subdirectory of the build directory. See %buildsubdir. |
| %{buildroot} | ~/rpmbuild/BUILDROOT | Build root: where files are "installed" during the %install stage, which copies files from a subdirectory of %{\_builddir} to a subdirectory of %{buildroot}. (Historically, %{buildroot} was in "/var/tmp/".) |
| %{buildsubdir} | %{\_builddir}/%{name} | Build subdirectory: a subdirectory within %{\_builddir} where files are compiled during the %build stage. It is set after %setup. |
| %{\_datadir} | /usr/share | Share directory. |
| %{\_defaultdocdir} | /usr/share/doc | Default documentation directory. |
| %{dist} | .fc*NUMBER* | Distribution+version short name (e.g. ".fc9") |
| %{fedora} | *NUMBER* | Number of fedora release (e.g. "9") |
| %{\_includedir} | /usr/include |  |
| %{\_infodir} | /usr/share/info |  |
| %{\_initrddir} | /etc/rc.d/init.d |  |
| %{\_libdir} | /usr/lib |  |
| %{\_libexecdir} | /usr/libexec |  |
| %{\_localstatedir} | /var |  |
| %{\_mandir} | /usr/share/man |  |
| %{name} |  | Name of package, set by Name: tag |
| %{\_sbindir} | /usr/sbin |  |
| %{\_sharedstatedir} | /var/lib |  |
| %{\_sysconfdir} | /etc |  |
| %{version} |  | Version of package, set by Version: tag |

Learn more about macros by looking in /etc/rpm/\* and /usr/lib/rpm, especially /usr/lib/rpm/macros. Also use rpm --showrc to show values that RPM will use for macros (altered by rpmrc and macro configuration files).

You can set your own macro values using %global, but be sure to define them before you use them. (Macro definitions can also refer to other macros.) For example:

%global date 2012-02-08

Use the "-E" option of rpmbuild to find the value of a macro in a SPEC file:

rpmbuild -E '%{\_bindir}' myfile.spec

Also see [Packaging/RPMMacros](http://fedoraproject.org/wiki/Packaging/RPMMacros) and [RPM Guide chapter 9](http://docs.fedoraproject.org/drafts/rpm-guide-en/ch09s07.html).

### Other tags

In addition to Requires and BuildRequires tags, you can also use these for controlling dependencies:

* **Provides**: list virtual package names that this package provides. For example, there might be a package "foo" that demands a particular functionality "bar" from another program. If there are several packages that can satisfy that demand, those packages can specify "Provides: bar" and the "foo" package can specify "Requires: bar". You could also use the ["alternatives" system](http://dailypackage.fedorabook.com/index.php?/archives/6-Wednesday-Why-The-Alternatives-System.html), but avoid if multiple users on the same system might want different default, as these settings are system-wide. Use "rpm -q --provides PACKAGENAME" to see what a given package provides. Some examples of virtual packages in Fedora:
  + MTA: Used for mail transport agents, such as sendmail.
  + tex(latex): Used for latex
* **Obsoletes**: remove another named package(s) when this package is installed. Use when the package name changes or when it totally replaces a different package.
* **Conflicts**: state what other packages cannot be installed simultaneously to this one. Avoid this if you can. See [Packaging/Conflicts](http://fedoraproject.org/wiki/Packaging/Conflicts).
* **BuildConflicts**: state what packages cannot be installed when building this package. Avoid this if you can.

To manage different architectures, there are two tags:

* **ExcludeArch**: to exclude an architecture on which the package doesn't build. For example:

ExcludeArch: ppc

* **ExclusiveArch**: to include only the specified architecture. Avoid this unless absolutely correct.

Valid architectures are listed at [Architectures](http://fedoraproject.org/wiki/Architectures).

### Subpackages

A SPEC file can define several binary package. In other words, one SRPM with one SPEC file can result in several RPMS. Note that there is still only one creation (%prep, %build, %install etc.) process. name-doc and name-devel subpackages are common for documentation and development files respectively.

Use the %package directive to start defining a subpackage:

%package subpackage\_name

After each %package directive, list the tags for the subpackage. This should at least include the Summary and Group tags, as well as the %description subpackage\_name and %files subpackage\_name directives:

Anything not specified by the subpackage will be inherited from its parent.

By default, if the package name is "foo" and the subpackage name is "bar", then the resulting subpackage will be "foo-bar". You can override it with the "-n" option (but you'll need to use it in all other directives too if you specify it here):

%package -n new\_subpackage\_name

[See the RPM Guide section on subpackages](http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch10s04.html) for more information.

### Conditionals

You can insert conditional statements, for example to test if you are creating a binary for a certain architecture:

%ifarch ARCHITECTURE\_NAME

the negated version with:

%ifnarch ARCHITECTURE\_NAME

or the more general conditional:

%if TRUE\_OR\_FALSE

There is an optional "%else" section; all of these are closed with "%endif".

### Application Specific Guidelines

There are many application-specific guidelines that can help you (e.g., for specific programming languages, applications, libraries, and build systems). Many of them are listed as part of the [Application Specific Guidelines of Packaging/Guidelines](http://fedoraproject.org/wiki/Packaging/Guidelines#Application_Specific_Guidelines). Examples of application-specific guidelines are those for:

* [Cmake](http://fedoraproject.org/wiki/Packaging:Cmake)
* [Emacs](http://fedoraproject.org/wiki/Packaging:Emacs)

Failing that, some other ways of finding application-specific help are:

* The 'SEARCH' command on Fedoraproject.org.
* [PackagingDrafts](http://fedoraproject.org/wiki/PackagingDrafts)
* A [Special Interest Group (SIG)](http://fedoraproject.org/wiki/SIGs)
* [Wiki pages prefixed with 'Packaging'](http://fedoraproject.org/wiki/Special:PrefixIndex/Packaging)

### Miscellaneous hints

[Packaging/FrequentlyMadeMistakes](http://fedoraproject.org/wiki/Packaging/FrequentlyMadeMistakes) has information on frequently-made mistakes. There are also some recommendations and controversial tricks on [PackageMaintainers/Packaging Tricks](http://fedoraproject.org/wiki/PackageMaintainers/Packaging_Tricks).

Try to write your SPEC files so that it is likely to work when a new release is made upstream, without any changes aside from bumping the version number and refreshing the source files. For example, if it contains \*.txt files with execute bits, instead of doing

chmod a-x Filename1.txt Filename2.txt Filename3.txt

consider doing this, which will handle new filenames that use the same file naming convention:

chmod a-x \*.txt

If you want to see lots of examples of scriptlets, you can show all the scriptlets on installed programs using:

rpm -qa --queryformat "\n\nPACKAGE: %{name}\n" --scripts | less

Don't try to interact with the user; RPM is designed to support batch installs. If an application needs to show a EULA, that needs to be part of its initial execution, not its installation.

You might not want to start services, because in a big install that could slow things down. If you install an init or systemd script, consider using chkconfig or systemctl to arrange for the service to be started/stopped on the next reboot. Before uninstalling, you should normally try to stop its services if they are running.

Uninstalling should reverse most changes made during installation, but don't remove any user-created files.

Normally, if there are binary executables, then debugging symbols are stripped from the normal binary packages and placed into a name-debug subpackage. If this shouldn't happen, you can disable the stripping and creation of this subpackage by putting this at the top of your SPEC:

%global \_enable\_debug\_package 0

%global debug\_package %{nil}

%global \_\_os\_install\_post /usr/lib/rpm/brp-compress %{nil}

To prevent stripping you may also need to do this in the %install section:

export DONT\_STRIP=1

A way to check for the version of Fedora in a SPEC file for conditional builds is:

%if 0%{?fedora} <= <version>

The ? causes the macro to evaluate to evaluate to blank if %fedora is not defined. This causes the end result to be the 0 (which is a number and thus fine), while not interfering with the result if there is actually a value for %fedora. (Note that this trick does not work in Koji "scratch" builds, where %fedora is set during the creation of a SRPM.)

GUI programs must have a desktop entry so that people can invoke it from the graphical desktop menu. For .desktop files, see [Fedora packaging guidelines for desktop files](http://fedoraproject.org/wiki/Packaging/Guidelines#Desktop_files) and [desktop entry spec](http://standards.freedesktop.org/desktop-entry-spec/latest/). For icons within /usr/share/icons, see [icon theme spec](http://standards.freedesktop.org/icon-theme-spec/icon-theme-spec-latest.html).

## Building the binary package

### Test with rpmlint

To catch many common errors early, run rpmlint on your SPEC file before trying to build anything from it:

$ rpmlint program.spec

If the reported error doesn't make sense, run it again with the "-i" option for longer messages.

Aim to have no errors, but sometimes rpmlint reports false positives. The [Fedora packaging guidelines](http://fedoraproject.org/wiki/Packaging/Guidelines#Use_rpmlint) explains which ones to ignore.

### Create binary RPMS from the SPEC file

Once you've created your SPEC file, build the SRPM and binary RPMS by running this:

$ rpmbuild -ba program.spec

If successful, RPMS will be created within ~/rpmbuild/RPMS and SRPMS will be created within ~/rpmbuild/SRPMS.

If it fails, go to the appropriate directory and see what is left over. To help debug, you can skip earlier stages that succeeded with the "--short-circuit option. For example, to restart at the %install stage (skipping earlier stages), do this:

$ rpmbuild -bi --short-circuit program.spec

If you just want to create an SRPM (which does not run the %prep or %build or other stages), run this:

rpmbuild -bs program.spec

### Testing binary RPMS with rpmlint

rpmlint can be run on SPEC files, RPMS and SRPMS to check for errors. You need to eliminate or justify warnings before posting a package. If you are in the SPECS directory, do this:

$ rpmlint *NAME*.spec ../RPMS/\*/*NAME*\*.rpm ../SRPMS/*NAME*\*.rpm

Enter the ~/rpmbuild/RPMS directory and into the architecture subdirectory. You will find some binary RPMS. Quickly see their files and permissions (to check whether they are correct) by doing:

$ rpmls \*.rpm

If they look fine, install them as root:

# rpm -ivp package1.rpm package2.rpm package3.rpm ...

Test the programs in a few different ways to see if everything works correctly. If it is a GUI tool, make sure it shows up in the desktop menu, otherwise the .desktop entry is wrong.

Uninstall packages later by doing:

# rpm -e package1 package2 package3

## Mock and Koji

[Mock](http://fedoraproject.org/wiki/Projects/Mock) is a powerful tool that uses the SRPM you have created to build binary packages within a nearly empty environment. This can reveal if you have accurate build dependencies. If it fails, then you forgot to list something in BuildRequires. See [Using Mock to test package builds](http://fedoraproject.org/wiki/Using_Mock_to_test_package_builds). Once your account is a member of the "mock" group, you can run commands like this to do local testing:

$ mock -r fedora-9-i386 rebuild path\_to\_source\_RPM

You can use Koji (which uses mock) to do builds on many different systems, some of which you may not have. [PackageMaintainers/Join](http://fedoraproject.org/wiki/PackageMaintainers/Join) and [PackageMaintainers/UsingKoji](http://fedoraproject.org/wiki/PackageMaintainers/UsingKoji) have more information about Koji. Once it's set up, you can test your SRPM on a variety of platforms by running commands like:

$ koji build --scratch dist-f9 path\_to\_source\_RPM

Replace dist-f9 with any later release of Fedora, but don't use dist-rawhide. Remember, the values of %fedora, %fc9 and so on will not be correct for a scratch build, so this won't work if your SPEC file does something different based on those values.

Your Koji builds can only depend on packages that are actually in the TARGET distribution repository. Thus, you can't use Koji to build for released distributions if your package depends on other new packages that Bodhi hasn't released yet. If you need to build against a package that is not yet a stable released update, you can file a ticket with rel-eng at: <https://fedorahosted.org/rel-eng/newticket> and request that that package be added as a buildroot override.

## Helpful tools

The rpmdevtools package has a number of helpful tools; "rpm -qil rpmdevtools" will show you what it installs.

* rpmdev-bumpspec : bump the release tag in the spec file and add a changelog comment with the right date and version format:

rpmdev-bumpspec --comment=COMMENT --userstring=NAME+EMAIL\_STRING SPECFILES

The yum-utils package also has some useful tools:

* yumdownloader : download the SRPM of the package by running:

yumdownloader --source PACKAGENAME

The auto-buildrequires package has a pair of nice tools for helping to figure out the proper BuildRequires entries. After installing this package, replace "rpmbuild" with "auto-br-rpmbuild" and you'll see an automatically generated BuildRequires list.

You might find [RUST](http://rust.sourceforge.net/) useful (GPL), though it does not create SPEC files of suitable quality for Fedora packages. [Alien](http://kitenet.net/%7Ejoey/code/alien/) converts between package formats. It won't produce clean SRPMS, but converting an existing package might provide helpful information.

## Guidelines and rules

When you create your packages, you'll need to follow the following rules and guidelines:

* [How to join the Fedora Package Collection Maintainers](http://fedoraproject.org/wiki/Join_the_package_collection_maintainers)
* [Packaging Guidelines](http://fedoraproject.org/wiki/Packaging:Guidelines)
* [Package Naming Guidelines](http://fedoraproject.org/wiki/Packaging:NamingGuidelines)
* [Package Licensing Guidelines](http://fedoraproject.org/wiki/Packaging:LicensingGuidelines)
* [Dist Tag Guidelines](http://fedoraproject.org/wiki/Packaging:DistTag)
* [Package Review Guidelines](http://fedoraproject.org/wiki/Packaging:ReviewGuidelines)

There are many official guidelines that will help you with specific circumstances (e.g. Java programs, OCaml programs, GNOME programs). You can also learn more from the [SIGs](http://fedoraproject.org/wiki/SIGs) and [Package Maintainers](http://fedoraproject.org/wiki/Category:Package_Maintainers) sections. [You can also see the list of all Wiki pages about Packaging](https://fedoraproject.org/wiki/Special:Prefixindex/Packaging) to see if any apply.

Failing that, you might find some useful recommendations in the unofficial [Packaging Drafts](https://fedoraproject.org/wiki/Special:Search?ns0=1&search=PackagingDrafts%2F&searchx=Search) and [Packaging Drafts To Do](http://fedoraproject.org/wiki/PackagingDrafts).

## Maintaining the package

Once your package has been accepted, you and your co-maintainers will need to maintain it. See [Package update HOWTO](http://fedoraproject.org/wiki/Package_update_HOWTO) and [Package update guidelines](http://fedoraproject.org/wiki/Package_update_guidelines). If you update the version in multiple releases of Fedora, do it backwards in time (e.g. release for Fedora N, then once that's accepted, Fedora N-1). The system presumes that later versions of Fedora have the same or later versions of programs.

Encourage the upstream developers to use standard source code release conventions. Using standard conventions makes packaging much easier. For more information, see:

* [Releasing Free/Libre/Open Source Software (FLOSS) for Source Installation](http://www.dwheeler.com/essays/releasing-floss-software.html) (a quick summary)
* [GNU Coding Standards release process](http://www.gnu.org/prep/standards/html_node/Managing-Releases.html)
* [Software Release Practice HOWTO](http://en.tldp.org/HOWTO/Software-Release-Practice-HOWTO/)
* [Filesystem Hierarchy Standard (FHS)](http://www.pathname.com/fhs/)
* [Packaging Unix software](http://offog.org/articles/packaging/)

## For more information

The [Package Maintainers](http://fedoraproject.org/wiki/Category:Package_Maintainers) page links to many other useful pages, and the [Package update HOWTO](http://fedoraproject.org/wiki/Package_update_HOWTO) describes how to update an existing package you already maintain in Fedora.

For more information, outside of the Fedora Wiki, see:

* [How to build RPM packages on Fedora](http://www.g-loaded.eu/2006/04/05/how-to-build-rpm-packages-on-fedora/) - very brief run-through
* Packaging software with RPM (developerWorks) [Part 1](http://www.ibm.com/developerworks/library/l-rpm1/), [Part 2](http://www.ibm.com/developerworks/library/l-rpm2/), and [Part 3](http://www.ibm.com/developerworks/library/l-rpm3.html)
* Fedora Classroom had a IRC session on packaging and you can refer to the logs at <https://fedoraproject.org/wiki/Building_RPM_packages_%2820090405%29>
* [Fedora Packager's Handbook](http://koti.welho.com/vskytta/packagers-handbook/packagers-handbook.html)
* [When Sally met Eddie](http://www.redhatmagazine.com/2008/02/28/when-sally-met-eddie-the-fedora-package-story/) - a simple tale, but little detail
* [Maximum RPM Book](http://rpm.org/max-rpm-snapshot/) - most complete information, but in some cases old/obsolete
* [RPM Guide, section on creating RPMs](http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch-creating-rpms.html) - this has lots of good information, and is slightly more up-to-date, but is a draft
* [Developer's guide, section on building RPMs](http://docs.fedoraproject.org/developers-guide/ch-rpm-building.html)
* [Creating RPMS slides](http://www.gurulabs.com/GURULABS-RPM-LAB/GURULABS-RPM-GUIDE-v1.0.PDF) from Guru Labs
* [The fight, my first attempt to make a readable rpm package building introduction.](http://freshrpms.net/docs/fight/)
* [Cambridge RPM tutorial](http://www-uxsup.csx.cam.ac.uk/talks/rpmbuild/rpmbuild.pdf) is a presentation on creating basic RPMs
* [RPM HOWTO: RPM at Idle by Donnie Barnes](http://en.tldp.org/HOWTO/RPM-HOWTO/index.html)
* [RPM HowTo by Dawson](http://home.fnal.gov/%7Edawson/rpms/howto/index.html)
* [Cross-distribution package HOWTO](http://en.opensuse.org/Build_Service/cross_distribution_package_how_to) has hints if you're building one RPM for many distributions.
* [Mandriva Rpm HowTo (en)](http://wiki.mandriva.com/en/Development/Howto/RPM) is an RPM tutorial, though for Mandriva (nee Mandrake). Note: In Fedora, do *not* recompress original tarballs, as Mandriva suggests, because that would change their cryptographic hashes.
* [Creating Your Own Linux RPM's - The Initial Software Build](http://linuxshellaccount.blogspot.com/2008/03/creating-your-own-linux-rpms-initial.html) is another brief intro, but it makes the point that "The process of building RPM's is much simpler than creating packages for Solaris... Fewer steps, and the ability to add all of your software information into one specification file, makes for a much tighter (and easier to modify or reproduce) software packaging system."
* [All you need to know about RPM](http://fedoranews.org/alex/tutorial/rpm/) (more about installing packages than creating them)
* The [rpm.org Wiki](http://wiki.rpm.org/) has some useful information, such as the [list of known RPM problems](http://wiki.rpm.org/Problems)

Note: The [rpm5.org](http://rpm5.org/) site has some documentation, but do not depend on it; that is the home of a *fork* of RPM maintained by Jeff Johnson. The RPM used by Fedora (and Novell/SuSE) is instead based at [rpm.org](http://www.rpm.org). [lwn.net has a brief article](http://lwn.net/Articles/236029/) about this.

[Categories](http://fedoraproject.org/wiki/Special:Categories):

* [Package Maintainers](http://fedoraproject.org/wiki/Category:Package_Maintainers)
* [How to](http://fedoraproject.org/wiki/Category:How_to)