原创 享学IT 2016-12-04 16:23:03 44792 收藏 130 版权
分类专栏： 【打怪升级】Linux运维 文章标签： rpm

【打怪升级】Linux运维 专栏收录该内容

RPM（Redhat Package Manager）是用于Redhat、CentOS、Fedora等Linux 分发版（distribution）的常见的软件包管理器。因为它允许分发已编译的软件，所以用户只用一个命令就可以安装软件。看到这篇文章的朋友想必已经知道RPM是个啥，rpm/yum命令怎么用，废话不多说，直接进入正题，来看看RPM包咋打。

1 准备

首先请准备一个Linux环境，比如CentOS。
RPM打包使用的是`rpmbuild`命令，这个命令来自rpm-build包，这个是必装的。

```
$ yum install rpm-build
```

当然也可以直接安装rpmdevtools，这个工具还包含一些其他的工具，同时它依赖rpm-build，所以直接安装的话会同时把rpm-build装上。

```
$ yum install rpmdevtools
```

当然，根据不同的软件构建过程，还需要其他的编译打包工具，比如C语言的`make`、`gcc`，python的`setuptools`等，根据需要安装即可。

2 原理

RPM打包的时候需要编译源码，还需要把编译好的配置文件啊二进制命令文件啊之类的东西按照安装好的样子放到合适的位置，还要根据需要对RPM的包进行测试，这些都需要先有一个“工作空间”。`rpmbuild`命令使用一套标准化的“工作空间”：

```
$ rpmdev-setuptree
```

`rpmdev-setuptree`这个命令就是安装rpmdevtools带来的。可以看到运行了这个命令之后，在`$HOME`家目录下多了一个叫做`rpmbuild`的文件夹，里边内容如下：

```
$ tree rpmbuild
rpmbuild
├── BUILD
├── RPMS
├── SOURCES
├── SPECS
└── SRPMS
```

> 如果没有安装rpmdevtools的话，其实用`mkdir`命令创建这些文件夹也是可以的。
>
> ```
> mkdir -p ~/rpmbuild/{BUILD,RPMS,SOURCES,SPECS,SRPMS}
> ```

从这些文件的名字大体也能看得出来都是干嘛用的。具体来说：

默认位置	宏代码	名称	用途
~/rpmbuild/SPECS	%_specdir	Spec 文件目录	保存 RPM 包配置（.spec）文件
~/rpmbuild/SOURCES	%_sourcedir	源代码目录	保存源码包（如 .tar 包）和所有 patch 补丁
~/rpmbuild/BUILD	%_builddir	构建目录	源码包被解压至此，并在该目录的子目录完成编译
~/rpmbuild/BUILDROOT	%_buildrootdir	最终安装目录	保存 %install 阶段安装的文件
~/rpmbuild/RPMS	%_rpmdir	标准 RPM 包目录	生成/保存二进制 RPM 包
~/rpmbuild/SRPMS	%_srcrpmdir	源代码 RPM 包目录	生成/保存源码 RPM 包(SRPM)

SPECS下是RPM包的配置文件，是RPM打包的“图纸”，这个文件会告诉`rpmbuild`命令如何去打包。“宏代码”这一列就可以在SPEC文件中用来代指所对应的目录，类似于编程语言中的宏或全局变量。当然`~/rpmbuild`这个文件夹也是有宏代码的，叫做`%_topdir`。

打包的过程有点像是流水线，分好几个工序：
1. 首先，需要把源代码放到`%_sourcedir`中；
2. 然后，进行编译，编译的过程是在`%_builddir`中完成的，所以需要先把源代码复制到这个目录下边，一般情况下，源代码是压缩包格式，那么就解压过来即可；
3. 第三步，进行“安装”，这里有点类似于预先组装软件包，把软件包应该包含的内容（比如二进制文件、配置文件、man文档等）复制到`%_buildrootdir`中，并按照实际安装后的目录结构组装，比如二进制命令可能会放在`/usr/bin`下，那么就在`%_buildrootdir`下也按照同样的目录结构放置；
4. 然后，需要配置一些必要的工作，比如在实际安装前的准备啦，安装后的清理啦，以及在卸载前后要做的工作啦等等，这样也都是通过配置在SPEC文件中来告诉`rpmbuild`命令；
5. 还有一步可选操作，那就是检查软件是否正常运行；
5. 最后，生成的RPM包放置到`%_rpmdir`，源码包放置到`%_srpmdir`下。

以上这些步骤都是配置在SPEC文件中的，具体来说各个阶段：

阶段	读取的目录	写入的目录	具体动作
%prep	%_sourcedir	%_builddir	读取位于 `%_sourcedir` 目录的源代码和 patch 。之后，解压源代码至 `%_builddir` 的子目录并应用所有 patch。
%build	%_builddir	%_builddir	编译位于 `%_builddir` 构建目录下的文件。通过执行类似 `./configure && make` 的命令实现。
%install	%_builddir	%_buildrootdir	读取位于 `%_builddir` 构建目录下的文件并将其安装至 `%_buildrootdir` 目录。这些文件就是用户安装 RPM 后，最终得到的文件。注意一个奇怪的地方: 最终安装目录 不是 构建目录。通过执行类似 `make install` 的命令实现。
%check	%_builddir	%_builddir	检查软件是否正常运行。通过执行类似 `make test` 的命令实现。很多软件包都不需要此步。
bin	%_buildrootdir	%_rpmdir	读取位于 `%_buildrootdir` 最终安装目录下的文件，以便最终在 `%_rpmdir` 目录下创建 RPM 包。在该目录下，不同架构的 RPM 包会分别保存至不同子目录， `noarch` 目录保存适用于所有架构的 RPM 包。这些 RPM 文件就是用户最终安装的 RPM 包。
src	%_sourcedir	%_srcrpmdir	创建源码 RPM 包（简称 SRPM，以 `.src.rpm` 作为后缀名），并保存至 `%_srcrpmdir` 目录。SRPM 包通常用于审核和升级软件包。

3 示例

解释再多不如一个例子来的明白，这里用官方文档中的例子来操作一遍。
下面演示 GNU“Hello World” 项目的打包过程。虽然用 C 语言程序打印 “Hello World” 到标准输出是小菜一碟，但 GNU 版本包含了与一个典型的 FOSS 软件项目相关的最常用的外围组件，包括配置/编译/安装环境、文档、国际化等等。GNU 版本包含了一个由源代码和 configure/make 脚本组成的 tar 文件，但并不包含打包信息。因此，这是一个很好的 RPM 包打包示例。

3.1 下载源码

还记得前面介绍到的几个阶段吗，先准备源码，这里我们直接下载官方例子的源码，是个压缩包：

```
$ cd ~/rpmbuild/SOURCES
$ wget http://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
```

> 不知道为啥有时候源码包下起来特别慢甚至下不动，可以先用迅雷下载下来，然后传到虚拟机里。

3.2 编辑SPEC文件

然后后续的步骤就交给SPEC文件来配置了，编辑SPEC文件（Emacs 和 vi 的最新版本有 .spec 文件编辑模式，它会在创建新文件时打开一个类似的模板。所以可使用以下命令来自动使用模板文件）：

目录

分类专栏

```
$ cd ~/rpmbuild/SPECS
$ vim hello.spec
```

既然有模板，那么后边的工作就是填空题了：

```
Name:     hello
Version:  2.1
Release:  1%{?dist}
Summary:  The "Hello World" program from GNU
Summary(zh_CN):  GNU "Hello World" 程序
License:  GPLv3+
URL:      http://ftp.gnu.org/gnu/hello
Source0:  http://ftp.gnu.org/gnu/hello/%{name}-%{version}.tar.gz

%description
The "Hello World" program, done with all bells and whistles of a proper FOSS
project, including configuration, build, internationalization, help files, etc.

%description -l zh_CN
"Hello World" 程序，包含 FOSS 项目所需的所有部分，包括配置，构建，国际化，帮助文件等.

%prep
%setup -q


%build
%configure
make %{?_smp_mflags}


%install
make install DESTDIR=%{buildroot}


%files
%doc

%changelog
* Sun Dec 4 2016 Your Name <youremail@xxx.xxx> - 2.10-1
- Update to 2.10
* Sat Dec 3 2016 Your Name <youremail@xxx.xxx> - 2.9-1
- Update to 2.9
```

1. `Name` 标签就是软件名，`Version` 标签为版本号，而 `Release` 是发布编号。
2. `Summary` 标签是简要说明，英文的话第一个字母应大写，以避免 `rpmlint` 工具（打包检查工具）警告。
3. `License` 标签说明软件包的协议版本，审查软件的 License 状态是打包者的职责，这可以通过检查源码或 LICENSE 文件，或与作者沟通来完成。
4. `Group` 标签过去用于按照 /usr/share/doc/rpm-/GROUPS 分类软件包。**目前该标记已丢弃，vim的模板还有这一条，删掉即可**，不过添加该标记也不会有任何影响。
   `%changelog` 标签应包含每个 Release 所做的更改日志，尤其应包含上游的安全/漏洞补丁的说明。Changelog 日志可使用 `rpm --changelog -q <packagename>` 查询，通过查询可得知已安装的软件是否包含指定漏洞和安全补丁。`%changelog` 条目应包含版本字符串，以避免 `rpmlint` 工具警告。
5. 多行的部分，如 `%changelog` 或 `%description` 由指令下一行开始，空行结束。
6. 一些不需要的行 (如 BuildRequires 和 Requires) 可使用 '#' 注释。
7. `%prep`、`%build`、`%install`、`%file`暂时用默认的，未做任何修改。

3.3 构建RPM包

有点迫不及待了，尝试执行以下命令，以构建源码、二进制和包含调试信息的软件包：

```
$ rpmbuild -ba hello.spec
```

1）包含要安装的文件
不过上边的命令执行失败了0_0。
命令执行后，提示并列出未打包的文件：

```
RPM build errors:
    Installed (but unpackaged) file(s) found:
   /usr/bin/hello
   /usr/share/info/dir
   /usr/share/info/hello.info.gz
   /usr/share/locale/bg/LC_MESSAGES/hello.mo
   /usr/share/locale/ca/LC_MESSAGES/hello.mo
   ...
```

那些需要安装在系统中的文件，我们需要在 %files 中声明它们，这样 `rpmbuild` 命令才知道哪些文件是要安装的。
注意不要使用形如 `/usr/bin/` 的硬编码, 应使用类似 `%{_bindir}/hello` 这样的宏来替代。手册页应在 `%doc` 中声明：`%doc %{_mandir}/man1/hello.1.*`。
由于示例的程序使用了翻译和国际化，因此会看到很多未声明的 i18 文件。 使用 推荐方法 来声明它们：

- 包含程序安装的相关文件
- 查找 %install 中的语言文件： `%find_lang %{name}`
- 添加编译依赖： `BuildRequires: gettext`
- 声明找到的文件： `%files -f %{name}.lang`

这样下来，`%files` 部分的内容为：

```
%files -f %{name}.lang
%doc AUTHORS ChangeLog NEWS README THANKS TODO
%license COPYING
%{_mandir}/man1/hello.1.*
%{_infodir}/hello.info.*
%{_bindir}/hello
```

2）info文件的处理
如果程序使用 GNU info 文件，你需要确保安装和卸载软件包，不影响系统中的其他软件，按以下步骤操作：

- 在 %install 中添加删除 'dir' 文件的命令： `rm -f %{buildroot}/%{_infodir}/dir`
- 在安装后和卸载前添加依赖 `Requires(post): info` 和 `Requires(preun): info`
- 添加以下安装脚本（在%install和%files中间即可，分别对应安装后和卸载前的阶段，详见后边内容）：

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

%preun
if [ $1 = 0 ] ; then
/sbin/install-info --delete %{_infodir}/%{name}.info %{_infodir}/dir || :
fi
```

3）看看各个目录里边的东西
* `%_sourcedir` 下边仍然是源码的压缩包；
* `%_builddir` 下边是源码解压出来的文件夹hello-2.10及其下边的所有文件；
* `%_buildrootdir` 下边是一个名为"hello-2.10-1.el7.centos.x86_64"的文件夹（那么生成的RPM包的完整名称也是 `{Name}-{Version}-{Release}.{Arch}.rpm` ），这个文件夹下边有"usr"文件夹，其下还有"bin"、"lib"、"share"、"src"这几个文件夹，可以看到这里的目录结构和安装之后各个文件和文件夹的位置已经是基本一致的了。这里要注意的是，"usr"所在的"根目录"，也就是"hello-2.10-1.el7.centos.x86_64"这个文件夹，用宏表示就是 `%{buildroot}` ，有的地方也用 `$RPM_BUILD_ROOT` 代替 `%{buildroot}` ，不过跟 `%{_buildrootdir}` 不是一个概念，请注意。

为什么是"趁着失败"呢，因为成功打包之后有些文件夹（比如 `%_builddir` 和 `%_buildrootdir` ）内的内容就会被清理掉了，不过也可以

在 %build 和 %install 阶段的时候把这俩文件夹内的东西 tree 一下或者干脆复制到其他地方再看也行。
那么 %build 和 %install 以及其他几个阶段一般怎么配置呢？

4）本示例最终的完整SPEC

```
Name:     hello
Version:  2.10
Release:  1%{?dist}
Summary:  The "Hello World" program from GNU
Summary(zh_CN):  GNU "Hello World" 程序
License:  GPLv3+
URL:      http://ftp.gnu.org/gnu/hello
Source0:  http://ftp.gnu.org/gnu/hello/%{name}-%{version}.tar.gz

BuildRequires:  gettext
Requires(post): info
Requires(preun): info

%description
The "Hello World" program, done with all bells and whistles of a proper FOSS
project, including configuration, build, internationalization, help files, etc.

%description -l zh_CN
"Hello World" 程序，包含 FOSS 项目所需的所有部分，包括配置，构建，国际化，帮助文件等.

%prep
%setup -q

%build
%configure
make %{?_smp_mflags}

%install
make install DESTDIR=%{buildroot}
%find_lang %{name}
rm -f %{buildroot}/%{_infodir}/dir

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

%preun
if [ $1 = 0 ] ; then
/sbin/install-info --delete %{_infodir}/%{name}.info %{_infodir}/dir || :
fi

%files -f %{name}.lang
%doc AUTHORS ChangeLog NEWS README THANKS TODO
%license COPYING
%{_mandir}/man1/hello.1.*
%{_infodir}/hello.info.*
%{_bindir}/hello

%changelog
* Sun Dec 4 2016 Your Name <youremail@xxx.xxx> - 2.10-1
- Update to 2.10
* Sat Dec 3 2016 Your Name <youremail@xxx.xxx> - 2.9-1
- Update to 2.9
```

那么就开动起来，在执行一下 rpmbuild 命令撒欢吧：

```
$ rpmbuild -ba hello.spec
```

OK，执行成功了，看看成果吧：

```
$ tree ~/rpmbuild/*RPMS
/root/rpmbuild/RPMS
└── x86_64
    ├── hello-2.10-1.el7.centos.x86_64.rpm
    └── hello-debuginfo-2.10-1.el7.centos.x86_64.rpm
/root/rpmbuild/SRPMS
└── hello-2.10-1.el7.centos.src.rpm
```

在RPMS文件夹下生成了RPM包，在 x86_64 下，表示所应用的架构，由于没有指定arch为 noarch ，所以默认用本机架构。在SRPMS文件夹下生产了源码包，源码包当然木有架构这一说了。
所以有些人喜欢在装软件的时候从源码开始安装，因为更能贴合本机的物理情况，就像用光盘安装windows和GHOST安装windows，相对来说光盘一步一步安装更好一点点，不过我比较懒，还是直接 yum install 。

5）运行一下下
既然已经有RPM包了，那就安装上吧：

```
$ rpm -ivh ~/rpmbuild/RPMS/x86_64/hello-2.10-1.el7.centos.x86_64.rpm
```

运行一下：

```
$ hello
Hello, world!
$ which hello
/usr/bin/hello
$ rpm -qf `which hello`
hello-2.10-1.el7.centos.x86_64
```

可以看到编译好的二进制文件 hello 已经装到 /usr/bin 下了，其他位置的文件请自行查看吧^_^。因为这个示例程序五脏俱全，不妨 man 一下，看看使用文档~

```
$ man hello
```

4 详解

SPEC文件是RPM打包的核心，下面就对SPEC文件中漏掉的而且比较重要的关于各个部分的配置方法进行详细说明：

4.1 %prep阶段

%prep 部分描述了解压源码包的方法。一般而言，其中包含 %autosetup 命令。另外，还可以使用 %setup 和 %patch 命令来指定操作 Source0 、 Patch0 等标签的文件。

%autosetup 命令
%autosetup 命令用于解压源码包。可用选项包括：

- -n name：如果源码包解压后的目录名称与 RPM 名称不同，此选项用于指定正确的目录名称。例如，如果 tarball 解压目录为 FOO，则使用 “%autosetup -n FOO”。
- -c name：如果源码包解压后包含多个目录，而不是单个目录时，此选项可以创建名为 name 的目录，并在其中解压。

%setup 命令
如果使用 %setup 命令，通常使用 -q 抑止不必要的输出。
如果需要解压多个文件，有更多 %spec 选项可用，这对于创建子包很有用。常用选项如下：

- -a number：在切换目录后，只解压指定序号的 Source 文件（例如 “-a 0” 表示 Source0）。
- -b number ：在切换目录前， 只解压指定序号的 Source 文件（例如 “-b 0” 表示 Source0）。
- -D：解压前，不删除目录。
- -T：禁止自动解压归档。

%patch 命令
如果使用 %autosetup 命令，则不需要手动进行补丁管理。如果你的需求很复杂，或需要与 EPEL 兼容，需要用到此部分的内容。 %patch0 命令用于应用 Patch0（ %patch1 应用 Patch1，以此类推）。Patches 是修改源码的最佳方式。常用的 -pNUMBER 选项，向 patch 程序传递参数，表示跳过 NUM 个路径前缀。

补丁文件名通常像这样 `telnet-0.17-env.patch`，命名格式为 `%{name} - %{version} - REASON.patch`（有时省略 version 版本）。补丁文件通常是 `diff -u` 命令的输出；如果你在 `~/rpmbuild/BUILD` 子目录执行此命令，则之后便不需要指定 -p 选项。

> 为一个文件制作补丁的步骤：
>
> ```
> cp foo/bar foo/bar.orig
> vim foo/bar
> diff -u foo/bar.orig foo/bar > ~/rpmbuild/SOURCES/PKGNAME.REASON.patch
> ```
>
> 如果需要修改多个文件，简单方法是复制 BUILD 下的整个子目录，然后在子目录执行 diff。切换至 `~rpmbuild/BUILD/NAME` 目录后，执行以下命令：
>
> ```
> cp -pr ./ ../PACKAGENAME.orig/
> ... 执行修改 ...
> diff -ur ../PACKAGENAME.orig . > ~/rpmbuild/SOURCES/NAME.REASON.patch
> ```
>
> 如果你想在一个补丁中编辑多个文件，你可以在编辑之前，使用 `.orig` 扩展名复制原始文件。然后，使用 `gendiff`（在 rpm-build 包中）创建补丁文件。

4.2 %build阶段

`%build` 阶段顾名思义就是对解压到`%_builddir`下的源码进行编译的阶段，整个过程在该目录下完成。
许多程序使用 GNU configure 进行配置。默认情况下，文件会安装到前缀为 "/usr/local" 的路径下，对于手动安装很合理。然而，打包时需要修改前缀为 "/usr"。共享库路径视架构而定，安装至 /usr/lib 或 /usr/lib64 目录。
由于 GNU configure 很常见，可使用 `%configure` 宏来自动设置正确选项（例如，设置前缀为 `/usr`）。一般用法如下：

```
%configure
make %{?_smp_mflags}
```

若需要覆盖 makefile 变量，请将变量作为参数传递给 make：

```
make %{?_smp_mflags} CFLAGS="%{optflags}" BINDIR=%{_bindir}
```

> 你会发现SPEC中会用到很多预定义好的宏，用来通过一个简单的宏来完成一个或一系列常见的操作，比如：`%prep` 阶段用于解压的 `%setup` 和 `%autosetup` ， `%build` 阶段的 `%configure` 等。

4.3 %install阶段

此阶段包含安装阶段需要执行的命令，即从 `%{_builddir}` 复制相关文件到 `%{buildroot}` 目录（通常表示从 `~/rpmbuild/BUILD` 复制到 `~/rpmbuild/BUILDROOT/XXX`）目录，并根据需要在 `%{buildroot}` 中创建必要目录。

> 容易混淆的术语：
> * "build 目录"，也称为 `%{_builddir}`，实际上与 "build root"，又称为 `%{buildroot}`，是不同的目录。在前者中进行编译，并将需要打包的文件从前者复制到后者， `%{buildroot}`通常为 `~/rpmbuild/BUILD/%{name}-%{version}-%{release}.%{arch}`。
> * 在 `%build` 阶段，当前目录为 `%{buildsubdir}`，是 `%prep` 阶段中在 `%{_builddir}` 下创建的子目录。这些目录通常名为 `~/rpmbuild/BUILD/%{name}-%{version}`。
> * `%install` 阶段的命令不会在用户安装 RPM 包时执行，此阶段仅在打包时执行。

一般，这里执行 "make install" 之类的命令：

```
%install
rm -rf %{buildroot} # 仅用于 RHEL 5
%makeinstall
```

> 理想情况下，对于支持的程序，你应该使用 `%makeinstall`（这又是一个宏），它等同于 `DESTDIR=%{buildroot}`，它会将文件安装到 `%{buildroot}` 目录中。
>
> > 使用 "%makeinstall" 宏。此方法可能有效，但也可能失败。该宏会展开为 `make prefix=%{buildroot}%{_prefix} bindir=%{buildroot}%{_bindir} ... install`，可能导致某些程序无法正常工作。请在 `%{buildroot}` 根据需要创建必要目录。
>
> 使用 `auto-destdir` 软件包的话，需要 `BuildRequires: auto-destdir`，并将 `make install` 修改为 `make-redir DESTDIR=%{buildroot} install`。这仅适用于使用常用命令安装文件的情况，例如 cp 和 install。

- **手动执行安装。这需要在 `%{buildroot}` 下创建必要目录，并从 `%{_builddir}` 复制文件至 `%{buildroot}` 目录。要特别注意更新，通常会包含新文件。示例如下：**

```
%install
rm -rf %{buildroot}
mkdir -p %{buildroot}%{_bindir}/
cp -p mycommand %{buildroot}%{_bindir}/
```

4.4 %check 阶段

如果需要执行测试，使用 `%check` 是个好主意。测试代码应写入 `%check` 部分（紧接在 `%install` 之后，因为需要测试 `%{buildroot}` 中的文件），而不是写入 `%{build}` 部分，这样才能在必要时忽略测试。通常，此部分包含：

```
make test
```

有时候也可以用：

```
make check
```

请熟悉 Makefile 的用法，并选择适当的方式。

4.5 %files 部分

此部分列出了需要被打包的文件和目录。

%files 基础
`%defattr` 用于设置默认文件权限，通常可以在 %files 的开头看到它。注意，如果不需要修改权限，则不需要使用它。其格式为：

```
%defattr(<文件权限>, <用户>, <用户组>, <目录权限>)
```

第 4 个参数通常会省略。常规用法为 `%defattr(-,root,root,-)`，其中 "-" 表示默认权限。
您应该列出该软件包拥有的所有文件和目录。**尽量使用宏代替目录名**，具体的宏列表如下：

```
%{_sysconfdir}        /etc
%{_prefix}            /usr
%{_exec_prefix}       %{_prefix}
%{_bindir}            %{_exec_prefix}/bin
%{_libdir}            %{_exec_prefix}/%{_lib}
%{_libexecdir}        %{_exec_prefix}/libexec
%{_sbindir}           %{_exec_prefix}/sbin
%{_sharedstatedir}    /var/lib
%{_datarootdir}       %{_prefix}/share
%{_datadir}           %{_datarootdir}
%{_includedir}        %{_prefix}/include
%{_infodir}           /usr/share/info
%{_mandir}            /usr/share/man
%{_localstatedir}     /var
%{_initddir}          %{_sysconfdir}/rc.d/init.d
%{_var}               /var
%{_tmppath}           %{_var}/tmp
%{_usr}               /usr
%{_usrsrc}            %{_usr}/src
%{_lib}               lib (lib64 on 64bit multilib systems)
%{_docdir}            %{_datadir}/doc
%{buildroot}          %{_buildrootdir}/%{name}-%{version}-%{release}.%{_arch}
$RPM_BUILD_ROOT       %{buildroot}
```

如果路径以 "/" 开头（或从宏扩展），则从 `%{buildroot}` 目录取用。否则，假设文件在当前目录中（例如：在 `%{_builddir}` 中，包含需要的文档）。如果您的包仅安装一个文件，如 `/usr/sbin/mycommand`，则 %files 部分如下所示：

```
%files
%{_sbindir}/mycommand
```

若要使软件包不受上游改动的影响，可使用通配符匹配所有文件：

```
%{_bindir}/*
```

包含一个目录：

```
%{_datadir}/%{name}/
```

注意，`%{_bindir}/*` 不会声明此软件包拥有 /usr/bin 目录，而只包含其中的文件。如果您列出一个目录，则该软件包拥有这个目录，及该目录内的所有文件和子目录。因此，不要列出 %{_bindir}，并且要小心的处理那些可能和其他软件包共享的目录。

如果存在以下情况，可能引发错误：

- 通配符未匹配到任何文件或目录
- 文件或目录被多次列出
- 未列出 %{buildroot} 下的某个文件或目录

您也可以使用 %exclude 来排除文件。这对于使用通配符来列出全部文件时会很有用，注意如果未匹配到任何文件也会造成失败。

%files 前缀
上边的"hello"的示例中，`%files` 部分还有用到 `%doc` 等宏，可能您看得一知半解，这里详细介绍一下。
如果需要在 %files 部分添加一个或多个前缀，用空格分隔。

`%doc` 用于列出 `%{_builddir}` 内，但不复制到 `%{buildroot}` 中的文档。通常包括 `README` 和 `INSTALL` 等。它们会保存至 `/usr/share/doc` 下适当的目录中，不需要声明 `/usr/share/doc` 的所有权。

> 注意：如果指定 `%doc` 条目，`rpmbuild < 4.9.1` 在安装前会将 `%doc` 目录删除。这表明已保存至其中的文档，例如，在 `%install` 中安装的文档会被删除，因此最终不会出现在软件包中。如果您想要在 `%install` 中安装一些文档，请将它们临时安装到 build 目录（不是 build root 目录）中，例如 `_docs_staging`，接着在 `%files` 中列出，如 `%doc _docs_staging/*` 这样。

配置文件保存在 `/etc` 中，一般会这样指定（确保用户的修改不会在更新时被覆盖）：

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

如果更新的配置文件无法与之前的配置兼容，则应这样指定：

```
%config %{_sysconfdir}/foo.conf
```

"%attr(mode, user, group)" 用于对文件进行更精细的权限控制，"-" 表示使用默认值：

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

"%caps(capabilities)" 用于为文件分配 POSIX capabilities。例如：

```
%caps(cap_net_admin=pe) FOO.BAR
```

如果包含特定语言编写的文件，请使用 %lang 来标注：

```
%lang(de) %{_datadir}/locale/de/LC_MESSAGES/tcsh*
```

使用区域语言（Locale）文件的程序应遵循 处理 i18n 文件的建议方法：

- 在 %install 步骤中找出文件名： %find_lang ${name}
- 添加必要的编译依赖： BuildRequires: gettext
- 使用找到的文件名： %files -f ${name}.lang

4.6 Scriptlets

当用户安装或卸载 RPM 时，您可能想要执行一些命令。这可以通过 scriptlets 完成。

脚本片段可以：

- 在软体包安装之前（`%pre`）或之后（`%post`）执行
- 在软体包卸载之前（`%preun`）或之后（`%postun`）执行
- 在事务开始（`%pretrans`）或结束（`%posttrans`）时执行

> 例如，每个二进制 RPM 包都会在动态链接器的默认路径中存储共享库文件，并在 %post 和 %postun 中调用 ldconfig 来更新库缓存。如果软件包有多个包含共享库的子包，则每个软体包也需要执行相同动作。
>
> ```
> %post -p /sbin/ldconfig
> %postun -p /sbin/ldconfig
> ```
>
> 如果仅执行一个命令，则 "-p" 选项会直接执行，而不启用 shell。然而，若有许多命令时，不要使用此选项，按正常编写 shell 脚本即可。

如果你在脚本片段中执行任何程序，就必须以 `Requires(CONTEXT)`（例： `Requires(post)`）的形式列出所有依赖。

`%pre`、`%post`、`%preun` 和 `%postun` 提供 `$1` 参数，表示动作完成后，系统中保留的此名称的软件包数量。因此可用于检查软件安装情况，不过不要比较此参数值是否等于 2，而是比较是否大于等于 2。对于 `%pretrans` 和 `%posttrans`，$1 的值恒为 0。

例如，如果软件包安装了一份 info 手册，那么可以用 info 包提供的 install-info 来更新 info 手册索引。首先，我们不保证系统已安装 info 软件包，除非明确声明需要它；其次，我们不想在 install-info 执行失败时，使软件包安装失败：

```
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
```

上边的示例中还有一个安装 info 手册时的小问题需要解释一下。install-info 命令会更新 info 目录，所以我们应该在 `%install` 阶段删除 `%{buildroot}` 中无用的空目录：

```
rm -f %{buildroot}%{_infodir}/dir
```

5 命令及工具

5.1 rpmbuild打包

一旦 SPEC 编写完毕，请执行以下命令来构建 SRPM 和 RPM 包：

```
$ rpmbuild -ba program.spec
```

如果成功，RPM 会保存至 `~/rpmbuild/RPMS`，SRPM 会保存至 `~/rpmbuild/SRPMS`。

如果失败，请查看 BUILD 目录的相应编译日志。为了帮助调试，可以用 `--short-circuit` 选项来忽略成功的阶段。例如，若想要（略过

更早的阶段）重新从 `%install` 阶段开始，请执行：

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

如果只想创建 RPM，请执行：

```
rpmbuild -bb program.spec
```

如果只想创建 SRPM（不需要执行 %prep 或 %build 或其他阶段），请执行：

```
rpmbuild -bs program.spec
```

5.2 rpmlint检查

为避免常见错误，请先使用 `rpmlint` 查找 SPEC 文件的错误：

```
$ rpmlint program.spec
```

如果返回错误/警告，使用 "-i" 选项查看更详细的信息。

也可以使用 `rpmlint` 测试已构建的 RPM 包，检查 SPEC/RPM/SRPM 是否存在错误。你需要在发布软件包之前，解决这些警告。此页面提供一些常见问题的解释。如果你位于 SPEC 目录中，请执行：

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

进入 `~/rpmbuild/RPMS` 下的特定架构目录中，您会发现有许多二进制 RPM 包。使用以下命令快速查看 RPM 包含的文件和权限：

```
$ rpmls *.rpm
```

5.3 rpm安装

如果看上去正常，以 root 身份安装它们：

```
$ rpm -ivp package1.rpm package2.rpm package3.rpm ...
```

以不同方式来测试程序，看看是否全部都正常工作。如果是 GUI 工具，请确认其是否出现在桌面菜单中，否则表示 .desktop 条目可能有错。
最后卸载软件包：

```
$ rpm -e package1 package2 package3
```

6 备查

6.1 rpmbuild目录

默认位置	宏代码	名称	用途
~/rpmbuild/SPECS	%_specdir	Spec 文件目录	保存 RPM 包配置（.spec）文件
~/rpmbuild/SOURCES	%_sourcedir	源代码目录	保存源码包（如 .tar 包）和所有 patch 补丁
~/rpmbuild/BUILD	%_builddir	构建目录	源码包被解压至此，并在该目录的子目录完成编译
~/rpmbuild/BUILDROOT	%_buildrootdir	最终安装目录	保存 %install 阶段安装的文件
~/rpmbuild/RPMS	%_rpmdir	标准 RPM 包目录	生成/保存二进制 RPM 包
~/rpmbuild/SRPMS	%_srcrpmdir	源代码 RPM 包目录	生成/保存源码 RPM 包(SRPM)

6.2 spec文件阶段

阶段	读取的目录	写入的目录	具体动作
`%prep`	`%_sourcedir`	`%_builddir`	读取位于 `%_sourcedir` 目录的源代码和 patch 。之后，解压源代码至 `%_builddir` 的子目录并应用所有 patch。
`%build`	`%_builddir`	`%_builddir`	编译位于 `%_builddir` 构建目录下的文件。通过执行类似 `./configure && make` 的命令实现。
`%install`	`%_builddir`	`%_buildrootdir`	读取位于 `%_builddir` 构建目录下的文件并将其安装至 `%_buildrootdir` 目录。这些文件就是用户安装 RPM 后，最终得到的文件。注意一个奇怪的地方: 最终安装目录 不是 构建目录。通过执行类似 `make install` 的命令实现。
`%check`	`%_builddir`	`%_builddir`	检查软件是否正常运行。通过执行类似 `make test` 的命令实现。很多软件包都不需要此步。
`bin`	`%_buildrootdir`	`%_rpmdir`	读取位于 `%_buildrootdir` 最终安装目录下的文件，以便最终在 `%_rpmdir` 目录下创建 RPM 包。在该目录下，不同架构的 RPM 包会分别保存至不同子目录， `noarch` 目录保存适用于所有架构的 RPM 包。这些 RPM 文件就是用户最终安装的 RPM 包。
`src`	`%_sourcedir`	`%_srcrpmdir`	创建源码 RPM 包（简称 SRPM，以 `.src.rpm` 作为后缀名），并保存至 `%_srcrpmdir` 目录。SRPM 包通常用于审核和升级软件包。

6.3 代表路径的宏列表

```
%{_sysconfdir}        /etc
%{_prefix}            /usr
%{_exec_prefix}       %{_prefix}
%{_bindir}            %{_exec_prefix}/bin
%{_libdir}            %{_exec_prefix}/%{_lib}
%{_libexecdir}        %{_exec_prefix}/libexec
%{_sbindir}           %{_exec_prefix}/sbin
%{_sharedstatedir}    /var/lib
%{_datarootdir}       %{_prefix}/share
%{_datadir}           %{_datarootdir}
%{_includedir}        %{_prefix}/include
%{_infodir}           /usr/share/info
%{_mandir}            /usr/share/man
%{_localstatedir}     /var
%{_initddir}          %{_sysconfdir}/rc.d/init.d
%{_var}               /var
%{_tmppath}           %{_var}/tmp
%{_usr}               /usr
%{_usrsrc}            %{_usr}/src
%{_lib}               lib (lib64 on 64bit multilib systems)
%{_docdir}            %{_datadir}/doc
%{buildroot}          %{_buildrootdir}/%{name}-%{version}-%{release}.%{_arch}
$RPM_BUILD_ROOT       %{buildroot}
```

6.4 参考文档

- Fedora Packaging Guidelines
- How to create an RPM package
- How to create a GNU Hello RPM package
- Spec File Preamble

RPM打包全过程SPEC详解 02-23
描述了将java项目根据SPEC文件配置打包成RPM包，安装，卸载全过程

计算机组成原理实验教程 12-03
西北工业大学计算机组成原理实验课唐都仪器实验帮助，同实验指导书。分为运算器，存储器，控制器，模型计算机，输入输出系统5个章节

优质评论可以帮助作者获得更高权重 评论

柳下忙牛： 看了很多资料，这篇写的真的很详细，感谢抽时间整理分享。 4月前 回复

turtle_2： 写的非常详细，也别是"%install 阶段的命令不会在用户安装 RPM 包时执行，此阶段仅在打包时执行。 " 解决了我的疑问，谢谢！ 3年前 回复

登录 查看 11 条热评

相关推荐

CentOS下打包RPM的详细步骤(RPMBUILD命令的一个使用范... 7-16
首先,我们需要了解一些预备知识,在旧版本的CentOS中,rpmbuild工具默认的工作路径是/usr/src/redhat,因为权限原因,一般用户身份不能制作rpm软件包,只...

rpmbuild制作包的详细过程_码农崛起 6-30
安装速度比源码包快的多。缺点: 经过编译,不能看到源代码,功能选择不如源码灵活。依赖性强。首先说一下流程: 打rpm 包需要的东西有 源码、spec文件(...

CentOS下打包RPM的详细步骤（RPMBUILD命令的一个使用范例） u010384744的博客 2万+
日期2018-07-05综述最近在学习《鸟哥的LINUX私房菜》第23章中一个打包自己软件的范例。自己根据所学，自己总结了最新版本（CentOS Linux releas...

rpmbuild RPM制作教程 06-06
把源码包编译成rpm包不可缺少的一手资料。

1万+ rpm包制作rpmbuild_lxb122435677的博客 6-8
周排名 rpm -e:卸载 rpm -q:查询 rpmbuild:使用 spec 语法 %postun: %preun: 安装rpm包spec语法 %pre -> %install -> %post 卸载rpm包spec语法 %preun -> ...

95万+ 总排名 1123 粉丝 803 获赞 380 评论 1056 收藏

Linux——SRPM的使用:rpmbuild_开往春天的地铁 7-11
我们知道在 /root/rpmbuild/SOURCES 里面会放置原始档 (tarball) 以及相关的修补档 (patch file), 而我们也知道编译需要的步骤大抵就是 ./configure, make...

Linux下RPM打包制作过程 09-15
主要介绍了Linux下RPM打包制作的详细流程，并分享了相关实例代码，一起学习下。

Linux下rpmbuild命令的使用 qq_42303254的博客 1万+
rpmbuild中文手册：rpmbuild中文手册 一、RPM包的分类 RPM有五种基本的操作功能：安装、卸载、升级、查询和验证。 linux软件包分为两大类：（1...

rpmbuild制作rpm 包_哎幽的成长 7-22
可以先rpmbuild -bp ,再-bc 再-bi 如果没问题,rpmbuild -ba 生成src包与二进制包。 本文参考: http://laoguang.blog.51cto.com/6013350/1103628/ http://bl...

rpmbuild打包rpm_boog14的博客_rpmbuild 打包 7-3
也就是第几次制作rpm后面可使用%{release}引用License: GPL#许可,GPL还是BSD等(一般为GPL)Packager: RPMBuild#打包者的信息Group: Application...

rpm 讲解 Alan Zhuang的博客 4万+
CentOS7主要有rpm和yum这两种包软件的管理。两种包的管理各有用处，其中主要区别是：YUM使用简单但需要联网，YUM会去网上的YUM包源去获取...

RPM打包流程 sunzhengtaolz的博客 1222
一、rpm打包工具安装 yum install rpm-build yum install rpmdevtools 二、工具的使用方法 1. 创建文件夹 [root@localhost ~]# rpmdev-setuptree [root@lo...

两种姿势教你如何定制RPM包(rpmbuild与fpm)_weixin_341... 7-11
rpmbuid是用于制作rpm格式包的工具。rpm 4.4.x版本之前,rpmbuid工具默认的工作车间为/usr/src/redhat,所以造成普通用户不能制作rpm包。rpm 4.5.x版...

rpm包如何制作-个人经验 2万+
原文链接：http://blog.chinaunix.net/uid-21123336-id-1830527.html 一 RPM介绍 RPM 前是Red Hat Package Manager 的缩写，本意是Red Hat 软件包管...

RPM打包指南 02-18
Linux RPM打包指南，特别是一些全局变量宏定义，供开发查询使用

rpm-build 建立rpm包的命令 09-09
rpm-build 建立rpm包的命令rpm-build 建立rpm包的命令

RPM包rpmbuild SPEC文件深度说明 03-16
NULL 博文链接：https://hlee.iteye.com/blog/343499

java 打包rpm打包文档.zip 06-17
rpm 将springboot 应用 打包为rpm安装包，升级包的文档，打包文档，附完整案例，还有遇到的坑，如升级后不能自启动,包括springboot jar启动脚本，停...

Rpm打包原理详解.doc 12-18
制作 rpm 不仅仅是打包，更可以解决菜单创建、打补钉、完成大量预配置、与其他软件包互动等操作。使用源代码安装要求用户了解基本的编译过程、能...

rpm打包方法 xuleisdjn的专栏 484
bash build.sh 然后产生在RPMS/mips64el下 . 目录结构如下 #tree . . |-- BUILD |-- BUILDROOT |-- RPMS | |-- mips64el | `-- 123est-1.0.0-1001.mips64el.r...

rpm文件解压 一个CD包的博客 9993
检查命令是否安装解压rpm需要用到rpm2cpio命令，先在命令行中敲入rpm2cpio看是否已经安装了此命令，要是没有则先安装。# 安装rpm2cpio命令 yum i...

linux-centos浅谈之rpm和rpmbuild 乌托邦 1万+
一、简介 RPM（Red Hat Package Manager）是用于 Linux 分发版（distribution）的最常见的软件包管理器。RPM包命名方式：name-version-release.ar...

rpm 打包及其命令 Mr.Buffoon 6551
PRM打包： RPM打包使用的是rpmbuild命令，这个命令来自rpm-build包，这个是必装的。 $ yum install rpm-build 当然也可以直接安装rpmdevtools，这...

%ldconfig_scriptlets 替换 Linux 148
参考链接： https://fedoraproject.org/wiki/Changes/Removing_ldconfig_scriptlets %ldconfig_scriptlets 替换为： %post -p /sbin/ldconfig %postun -p /sbin...

享学IT
码龄7年 暂无认证
91
原创
5459
积分
私信 关注

搜博主文章

热门文章
Ubuntu apt-get彻底卸载软件包 295035
（5）Spring WebFlux快速上手——响应式Spring的道法术器 120528
RPM打包原理、示例、详解及备查 44767
CentOS7 修改设置静态IP和DNS 34936
响应式Spring的道法术器（Spring WebFlux教程） 34737

最新评论
（12）自定义数据流（实战Docker事件...
享学IT: 可以用于监控方面，不止docker，其他资源或中间件——如果支持的话——...
（1）什么是响应式编程——响应式Sprin...
羽三:
（5）Spring WebFlux快速上手——响应...
Arthuse: 请问一下博主，我用mysql+jpa，依赖正常配置，但是启动项目报错说JPA...
（12）自定义数据流（实战Docker事件...
wzng: 老师，这个docker推送有什么使用场景？
（20）操作符熔合——响应式Spring的道...
yangpenggg: 看完所有的，代码也跑了一遍，有些不理解，主要是思想，慢慢理解

最新文章
CentOS8使用国内源
Ceph性能测试（RBD、CephFS、NFS、Cache Tier）
CentOS8使用cephadm部署和配置Ceph Octopus

2020年 8篇 2018年 29篇
2017年 23篇 2016年 30篇
2015年 7篇