https://blog.csdn.net/mrbuffoon/article/details/82766664

# rpm 打包及其命令

yum install rpm-build rpmdevtools -y

rpm2cpio \*.rpm | cpio -div #查看rpm包

## 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”：

[root@localhost ~]# rpmdev-setuptree

[root@localhost ~]# tree rpmbuild

rpmbuild

├── BUILD

├── RPMS

├── SOURCES

├── SPECS

└── SRPMS

5 directories, 0 files

[root@localhost ~]#

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}

默认位置 宏代码 名称和用途

rpmbuild/SPECS: %\_specdir 保存RPM包配置文件(.spec)

rpmbuild/SOURCES: %\_sourcedir 保存源码和补丁(.tar, patch)

rpmbuild/BUILD: %\_builddir 构建目录，源码包被解压至此，在该子目录下完成编译

rpmbuild/BUILDROOT: %\_buildrootdir 最终安装目录，保存%install阶段安装的文件

rpmbuild/RPMS: %\_rpmdir 保准RPM包，生成/保存二进制RPM包

rpmbuild/SRPMS: %\_srcrpmdir 源码RPM包目录，生成/保存源码RPM包(SRPM)

## 从spec文档建立RPM包

rpmbuild -ba hello.spec

从spec文档建立有以下选项：

-bp #只执行spec的%pre 段(解开源码包并打补丁，即只做准备)

-bc #执行spec的%pre和%build 段(准备并编译)

-bi #执行spec中%pre，%build与%install(准备，编译并安装)

-bl #检查spec中的%file段(查看文件是否齐全)

-ba #建立源码与二进制包(常用)

-bb #只建立二进制包(常用)

## spec文档中常用的几个宏变量

1. RPM\_BUILD\_DIR: /usr/src/redhat/BUILD

2. RPM\_BUILD\_ROOT: /usr/src/redhat/BUILDROOT

3. %{\_sysconfdir}: /etc

4. %{\_sbindir}： /usr/sbin

5. %{\_bindir}: /usr/bin

6. %{\_datadir}: /usr/share

7. %{\_mandir}: /usr/share/man

8. %{\_libdir}: /usr/lib64

9. %{\_prefix}: /usr

10. %{\_localstatedir}: /usr/var

其实以上参数值都是在/usr/lib/rpm/macros文件中定义的。

## rpm的用法

-ivh：安装显示安装进度--install--verbose--hash

-e：删除包

-qf：查找指定文件属于哪个RPM软件包[Query File]；

-Uvh：升级软件包--Update；

-qpl：列出RPM软件包内的文件信息[Query Package list]；

-qpi：列出RPM软件包的描述信息[Query Package install package(s)]；

-Va：校验所有的RPM软件包，查找丢失的文件[View Lost]；

example:

rpm -ivh file.rpm ＃[安装新的rpm]--install--verbose--hash

rpm -e file.rpm ＃[删除一个rpm包]--erase

rpm -qa | grep httpd ＃[搜索指定rpm包是否安装]--all搜索\*httpd\*

rpm2cpio file.rpm |cpio -div #[抽出文件]

rpm -Uvh file.rpm ＃[升级一个rpm]—upgrade

rpm -q samba //查询程序是否安装

rpm -ivh /media/cdrom/RedHat/RPMS/samba-3.0.10-1.4E.i386.rpm //按路径安装并显示进度

rpm -ivh --relocate /=/opt/gaim gaim-1.3.0-1.fc4.i386.rpm //指定安装目录

rpm -ivh --test gaim-1.3.0-1.fc4.i386.rpm //用来检查依赖关系；并不是真正的安装；

rpm -Uvh --oldpackage gaim-1.3.0-1.fc4.i386.rpm //新版本降级为旧版本

rpm -ql httpd #[搜索rpm包]--list所有文件安装目录

rpm -qpi Linux-1.4-6.i368.rpm #[查看rpm包]--query--package--install package信息

rpm -qpf Linux-1.4-6.i368.rpm #[查看rpm包]--file

rpm -qpR file.rpm #[查看包]依赖关系

## src.rpm的用法

#解压缩，编译和安装

rpm --recompile vim-4.6-4.src.rpm

#安装完成后，还会把编译生成的可执行文件重新包装成i386.rpm的RPM软件包

rpm --rebuild vim-4.6-4.src.rpm

https://blog.csdn.net/Michaelwubo/article/details/105839082

**rpmbuild制作包的详细过程**

## 宏变量，spec模板，rpmdev-spec，rpmdev-newinit

yum install tree rpm-build rpmdevtools -y

#工作车间\_topdir

\_topdir默认值为$HOME /rpmbuild，定义在/usr/lib/rpm/macros

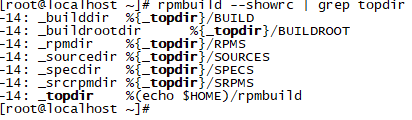
cat /usr/lib/rpm/macros | grep topdir 或者 rpmbuild --showrc | grep \_topdir

如果不想使用默认值，可以在~/.rpmmacros手工定义\_topdir，示例如下：

vi ~/.rpmmacros

%\_topdir /home/ibuler/rpmbuild

#查看系统默认的工作车间



#创建rpmbuild目录，手工创建亦可

#手工创建命令：mkdir -p ~/rpmbuild/{BUILD, RPMS, SOURCES, SPECS, SRPMS}

[root@localhost ~]# rpmdev-setuptree

[root@localhost ~]# tree rpmbuild

rpmbuild

├── BUILD

├── RPMS

├── SOURCES

├── SPECS

└── SRPMS

5 directories, 0 files

[root@localhost ~]#

#spec配置模板

[root@localhost ~]# ls /etc/rpmdevtools/

devscripts.conf spectemplate-dummy.spec spectemplate-ocaml.spec spectemplate-python.spec template.init

newspec.conf spectemplate-lib.spec spectemplate-perl.spec spectemplate-R.spec

rmdevelrpms.conf spectemplate-minimal.spec spectemplate-php-pear.spec spectemplate-ruby.spec

[root@localhost ~]#

#创建spec模板

rpmdev-newspec xxx.spec

#rpmdev还能自动生成/etc/init.d/下的服务脚本{start|stop|restart}

rpmdev-newinit appN

## 二：打包过程

### 1、目录介绍

BUILD：源代码解压以后放的位置，只需提供BUILD目录，具体里面放什么，不用我们管，所以真正的制作车间是BUILD目录。

BUILDROOT：假根，使用install临时安装到这个目录，把这个目录当作根来用的，所以在这个目录下的目录文件，才是真正的目录文件。

RPMS：制作完成后的rpm包存放目录，为特定平台指定子目录（i386,i686,ppc）。

SOURCES：收集的源文件，源材料，补丁文件等存放位置。

SPECS：存放spec文件，作为制作rpm包的领岗文件，文件以.spec结尾。

SRPMS：src格式的rpm包位置 ，既然是src格式的包，就没有平台的概念了

### 2、打包过程

基本格式：rpmbuild [options] [spec文档|tarball包(或者压缩包—以.gz或.xz或.bz2结尾的)|源码包]

options有下面的几种选择：

1.-bp #只执行spec的%pre段(解开源码包并打补丁,即只做准备)

2.-bc #执行spec的%pre和%build 段(准备并编译)

3.-bi #执行spec中%pre,%build与%install(准备,编译并安装)

4.-bl #检查spec中的%file段(查看文件是否齐全)

5.-ba #建立源码与二进制包(常用):即编译后做成\*.rpm和\*.src.rpm

6.-bb #只建立二进制包(常用):即编译后做成\*.rpm

7.-bs #只建立源码包:即只做成\*.src.rpm

-tc -ti -ta -tb -ts 的功能类似，只是所需参数由spec文件变成tar包。

a 构建二进制包和源代码包(在执行 %prep, %build, %install 之后)

b 构建二进制包(在执行 %prep, %build, %install 之后)

p 执行 spec 文件的"%prep"阶段。等价于解压缩及应用补丁。

c 执行 spec 文件的"%build"阶段(在执行了 %prep 之后)。等价于执行了"make"。

i 执行 spec 文件的"%install"阶段(在执行了 %prep, %build 之后)。等价于"make install"。

l 执行一次"列表检查"。spec 文件的"%files"段落中的宏被扩展，检测每个文件是否都在。

s 只构建源代码包

### 3、spec详解

#### 3.1 语法

https://www.cnblogs.com/schangech/p/5641108.html

https://blog.csdn.net/Michaelwubo/article/details/105886315

1、if结构

引用

%if %{str}

%else

动作

%endif

其中%{str}是条件，0为假，非0为真。

2、?:结构

引用

%{?变量:动作1}动作2

其中{}用于控制范围，而“？”号和“：”号是分割符，如果要判断条件是非的情况，可以在“？”号前加“！”号。

此条件与前面的%if有点不同，其只判断变量是否定义，定义了就为真，否则就为假，即使变量定义为0，也为真，并运行后面的语句。

关键字

spec脚本包括很多关键字，主要有：

Name: 软件包的名称，后面可使用%{name}的方式引用

Summary: 软件包的内容概要

Version: 软件的实际版本号，例如：1.0.1等，后面可使用%{version}引用

Release: 发布序列号，例如：1linuxing等，标明第几次打包，后面可使用%{release}引用

Group: 软件分组，建议使用标准分组

Group：

软件包所属类别，具体类别有：

Amusements/Games （娱乐/游戏）

Amusements/Graphics（娱乐/图形）

Applications/Archiving （应用/文档）

Applications/Communications（应用/通讯）

Applications/Databases （应用/数据库）

Applications/Editors （应用/编辑器）

Applications/Emulators （应用/仿真器）

Applications/Engineering （应用/工程）

Applications/File （应用/文件）

Applications/Internet （应用/因特网）

Applications/Multimedia（应用/多媒体）

Applications/Productivity （应用/产品）

Applications/Publishing（应用/印刷）

Applications/System（应用/系统）

Applications/Text （应用/文本）

Development/Debuggers （开发/调试器）

Development/Languages （开发/语言）

Development/Libraries （开发/函数库）

Development/System （开发/系统）

Development/Tools （开发/工具）

Documentation （文档）

System Environment/Base（系统环境/基础）

System Environment/Daemons （系统环境/守护）

System Environment/Kernel （系统环境/内核）

System Environment/Libraries （系统环境/函数库）

System Environment/Shells （系统环境/接口）

User Interface/Desktops（用户界面/桌面）

User Interface/X （用户界面/X窗口）

User Interface/X Hardware Support （用户界面/X硬件支持）

License: 软件授权方式，通常就是GPL

Source: 源代码包，可以带多个用Source1、Source2等源，后面也可以用%{source1}、%{source2}引用

BuildRoot: 这个是安装或编译时使用的“虚拟目录”，考虑到多用户的环境，一般定义为：

%{\_tmppath}/%{name}-%{version}-%{release}-root

或

%{\_tmppath}/%{name}-%{version}-%{release}-buildroot-%(%{\_\_id\_u} -n}

该参数非常重要，因为在生成rpm的过程中，执行make install时就会把软件安装到上述的路径中，在打包的时候，同样依赖“虚拟目录”为“根目录”进行操作。

后面可使用$RPM\_BUILD\_ROOT 方式引用。

URL: 软件的主页

Vendor: 发行商或打包组织的信息，例如RedFlag Co,Ltd

Disstribution: 发行版标识

Patch: 补丁源码，可使用Patch1、Patch2等标识多个补丁，使用%patch0或%{patch0}引用

Prefix: %{\_prefix} 这个主要是为了解决今后安装rpm包时，并不一定把软件安装到rpm中打包的目录的情况。这样，必须在这里定义该标识，并在编写%install脚本的时候引用，才能实现rpm安装时重新指定位置的功能

Prefix: %{\_sysconfdir} 这个原因和上面的一样，但由于%{\_prefix}指/usr，而对于其他的文件，例如/etc下的配置文件，则需要用%{\_sysconfdir}标识

Build Arch: 指编译的目标处理器架构，noarch标识不指定，但通常都是以/usr/lib/rpm/marcros中的内容为默认值

Requires: 该rpm包所依赖的软件包名称，可以用>=或<=表示大于或小于某一特定版本，例如：

libpng-devel >= 1.0.20 zlib

※“>=”号两边需用空格隔开，而不同软件名称也用空格分开

还有例如PreReq、Requires(pre)、Requires(post)、Requires(preun)、Requires(postun)、BuildRequires等都是针对不同阶段的依赖指定

Provides: 指明本软件一些特定的功能，以便其他rpm识别

Packager: 打包者的信息

%description 软件的详细说明

spec脚本主体

spec脚本的主体中也包括了很多关键字和描述，下面会一一列举。我会把一些特别需要留意的地方标注出来。

%prep 预处理脚本

%setup -n %{name}-%{version} 把源码包解压并放好

通常是从/usr/src/asianux/SOURCES里的包解压到/usr/src/asianux/BUILD/%{name}-%{version}中。

一般用%setup -c就可以了，但有两种情况：一就是同时编译多个源码包，二就是源码的tar包的名称与解压出来的目录不一致，此时，就需要使用-n参数指定一下了。

%patch 打补丁

通常补丁都会一起在源码tar.gz包中，或放到SOURCES目录下。一般参数为：

%patch -p1 使用前面定义的Patch补丁进行，-p1是忽略patch的第一层目录

%Patch2 -p1 -b xxx.patch 打上指定的补丁，-b是指生成备份文件

◎补充一下

引用

%setup 不加任何选项，仅将软件包打开。

%setup -n newdir 将软件包解压在newdir目录。

%setup -c 解压缩之前先产生目录。

%setup -b num 将第num个source文件解压缩。

%setup -T 不使用default的解压缩操作。

%setup -T -b 0 将第0个源代码文件解压缩。

%setup -c -n newdir 指定目录名称newdir，并在此目录产生rpm套件。

%patch 最简单的补丁方式，自动指定patch level。

%patch 0 使用第0个补丁文件，相当于%patch ?p 0。

%patch -s 不显示打补丁时的信息。

%patch -T 将所有打补丁时产生的输出文件删除。

%configure 这个不是关键字，而是rpm定义的标准宏命令。意思是执行源代码的configure配置

在/usr/src/asianux/BUILD/%{name}-%{version}目录中进行 ，使用标准写法，会引用/usr/lib/rpm/marcros中定义的参数。

另一种不标准的写法是，可参考源码中的参数自定义，例如：

引用

CFLAGS="$RPM\_OPT\_FLAGS" CXXFLAGS="$RPM\_OPT\_FLAGS" ./configure --prefix=%{\_prefix}

%build 开始构建包

在/usr/src/asianux/BUILD/%{name}-%{version}目录中进行make的工作 ，常见写法：

引用

make %{?\_smp\_mflags} OPTIMIZE="%{optflags}"

都是一些优化参数，定义在/usr/lib/rpm/marcros中

%install 开始把软件安装到虚拟的根目录中

在/usr/src/asianux/BUILD/%{name}-%{version}目录中进行make install的操作。这个很重要，因为如果这里的路径不对的话，则下面%file中寻找文件的时候就会失败。 常见内容有：

%makeinstall 这不是关键字，而是rpm定义的标准宏命令。也可以使用非标准写法：

引用

make DESTDIR=$RPM\_BUILD\_ROOT install

或 引用

make prefix=$RPM\_BUILD\_ROOT install

需要说明的是，这里的%install主要就是为了后面的%file服务的。所以，还可以使用常规的系统命令：

引用

install -d $RPM\_BUILD\_ROOT/

cp -a \* $RPM\_BUILD\_ROOT/

%clean 清理临时文件

通常内容为：

引用

[ "$RPM\_BUILD\_ROOT" != "/" ] && rm -rf "$RPM\_BUILD\_ROOT"

rm -rf $RPM\_BUILD\_DIR/%{name}-%{version}

※注意区分$RPM\_BUILD\_ROOT和$RPM\_BUILD\_DIR：

$RPM\_BUILD\_ROOT是指开头定义的BuildRoot，而$RPM\_BUILD\_DIR通常就是指/usr/src/asianux/BUILD，其中，前面的才是%file需要的。

%pre rpm安装前执行的脚本

%post rpm安装后执行的脚本

%preun rpm卸载前执行的脚本

%postun rpm卸载后执行的脚本

%preun %postun 的区别是什么呢？

前者在升级的时候会执行，后者在升级rpm包的时候不会执行

%files 定义那些文件或目录会放入rpm中

这里会在虚拟根目录下进行，千万不要写绝对路径，而应用宏或变量表示相对路径。 如果描述为目录，表示目录中除%exclude外的所有文件。

%defattr (-,root,root) 指定包装文件的属性，分别是(mode,owner,group)，-表示默认值，对文本文件是0644，可执行文件是0755

%exclude 列出不想打包到rpm中的文件

※小心，如果%exclude指定的文件不存在，也会出错的。

%changelog 变更日志

※特别需要注意的是：%install部分使用的是绝对路径，而%file部分使用则是相对路径，虽然其描述的是同一个地方。千万不要写错。

就是%file中必须明白，用的是相对目录 引用

%files

%defattr(-,root,root)

%{\_bindir}

%{\_libdir}

%{\_datadir}

%exclude %{\_libdir}/debug

制作补丁

详细看参考： [原]使用diff同patch工具

如何编写%file段

由于必须在%file中包括所有套件中的文件，所以，我们需要清楚编译完的套件到底包括那些文件？

常见的做法是，人工模拟一次编译的过程：

这样，整个套件的内容就会被放到/usr/local/xxx中，可根据情况编写%file和%exclude段。

※当然，这个只能对源码按GNU方式编写，并使用GNU autotool创建的包有效，若自定义Makefile则不能一概而论。

关于rpm中的执行脚本

如果正在制作的rpm包是准备作为放到系统安装光盘中的话，则需要考虑rpm中定义的脚本是否有问题。由于系统在安装的时候只是依赖于一个小环境进行，而该环境与实际安装完的环境有很大的区别，所以，大部分的脚本在该安装环境中都是无法生效，甚至会带来麻烦的。

所以，对于这样的，需要放到安装光盘中的套件，不加入执行脚本是较佳的方法。

另外，为提供操作中可参考的信息，rpm还提供了一种信号机制：不同的操作会返回不同的信息，并放到默认变量$1中。

引用

0代表卸载、1代表安装、2代表升级

#### 3.2 示例：nginx-1.2.2.tar.gz

https://www.cnblogs.com/bluevitality/p/6513097.html

#自定义宏，相当于Linux中"Key-Value"变量形式

%define Name nginx #---> 名称

%define Version 1.2.2 #---> 版本

%define CONFIGFILE 1.conf #---> 本rpm包中需更换的配置文件......

%define InstallPath /usr/local/nginx #---> 本rpm包默认安装的路径

#定义软件包信息，即："rpm -qi name.rpm " 查看到的内容

Name: %{Name} #---> 引用宏

Version: %{Version} #---> 引用宏

Release: 1%{?dist} #---> 引用宏（自带宏）

Summary: ....................................... #---> 一些描述信息

License: GPLv2 #---> 授权协议

URL: inmoonlight@.163.com

buildroot: %{\_topdir}/BUILDROOT #---> 指定生产车间（非常重要，因在生成rpm过程中执行make install时会把软件安装到此路径，打包时同样依此目录为“根目录”进行操作）

Source0: %{Name}-%{Version}.tar.gz #---> 指定源码编译的文件，默认路径：%{\_topdir}/SOURCES

SOURCE1: %{CONFIGFILE} #---> 指定要替换的配置文件，默认路径：%{\_topdir}/SOURCES

BuildRequires: gcc,make,automake,binutils #---> 软件依赖信息

Requires: bash >= 2.0 #---> 定义软件依赖信息，该rpm包所依赖的软件包名称，可用>=或<=表示大或小于特定版本

%description

This is %{Name} .....Just a test rpm suite.............

#安装前的准备工作，此处可写入执行脚本

%pre

useradd %{Name} -s /sbin/nologin

#安装前的准备：此段默认将Source目录内的源码包在BUILD目录解压为%{Name}-%{Version}格式的目录

%prep

%setup -q -n %{Name}-%{Version}　 #---> 参数：-c 解压缩之前先产生目录，-n newdir 将软件包解压在newdir目录，-q：quiet

#定义config动作

%build

./configure --prefix=%{InstallPath} --user=%{Name} --group=%{Name}

make %{?\_smp\_mflags}

#定义执行make install时的动作

%install

rm -rf %{buildroot} #---> 删除生产车间内的残留文件

%{\_\_make} install DESTDIR=%{buildroot} #---> 将软件安装至指定的目录

%{\_\_install} -p -D -m 0755 %{SOURCE1} %{buildroot}/usr/local/nginx/conf/%{CONFIGFILE} #---> 替换指定的配置文件

#赋予文件的默认权限及设置需在RPM包中保留的文件

%files

%doc

%defattr(-,root,root,-) #---> 指定包装文件属性，分别是(mode,owner,group)，- 表示默认值，文本文件是0644，可执行文件0755

%attr(0755,root,root) /usr/local/nginx/sbin/nginx #---> 针对单一文件设置权限

%{\_prefix}/\*

%{\_prefix}/local/nginx/conf/%{CONFIGFILE}

#制作完成后的清理工作

%clean

rm -rf %{buildroot}

#安装后的执行工作，此处可写入执行脚本

%post

chkconfig --add nginx

chkconfig --level 345 nginx on

#变更日志

%changelog

#### 3.3制作rpm步骤

我们在企业中有的软件基本都是编译的，我们每次安装都得编译，那怎么办呢？那就根据我们的需求制作RPM安装包吧。先来说说基本布骤:

1.Planning what you want 计划做什么rpm包。软件的？库的？

2.Gathering the software to package 收集原材料，即收集源码包

3.Patch the software as need 如果需要打补丁，收集补丁文件。此布骤不是必须

4.Outling any dependenies 确定依赖关系包

------------------ 上述动作可由我们手动编译一次软件来确定 -------------------

5.Building RPMs 开始动手制作RPM包

5.1 Set up the directory stucture 设定好目录结构，我们在这些目录中制作我们的RPM包，我们需要下列目录

BUILD 源代码解压后的存放目录

RPMS 制作完成后的RPM包存放目录，里面有与平台相关的子目录

SOURCES 收集的源材料，补丁的存放位置

SPECS SPEC文件存放目录

SRMPS 存放SRMPS生成的目录

5.2 Place the Sources in the right directory 把源材料放到正确的位置

5.3 Create a spec file that tell rpmbuild command what to do 创建spec文件，这是纲领文件，rpmbuild命令根据spec文件来制作合适的rpm包

5.4 Build the source and binary RPMS 制作src或二进制rpm包

6.Test RPMS 测试制作的PRM包

7.Add signature for RPM 为RPM包签名

#### 3.4制作rpm实例

我还是用连贯的 话为大家叙述一遍吧，我们首先确实我们要为什么做rpm包，通常我们是为一些软件，比如httpd,nginx等，然后去收集这些软件包的源代码，如果有 需要的话也收集这些补丁文件，手动编译安装一下这个软件(当然如果是不需要编译的就不用演练了)，确定依赖的软件包，并记录下来，下面开始准备制作 tengine的PRM包吧：

##### 1 创建普通用户

用root的可能会因为代码问题导致毁灭的后果

useradd ibuler

su - ibuler

##### 2. 手工指定工作车间\_topdir

工作车间即在哪个目录下制作RPM

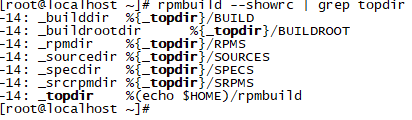
\_topdir默认值为$HOME /rpmbuild，定义在/usr/lib/rpm/macros

cat /usr/lib/rpm/macros | grep topdir 或者 rpmbuild --showrc | grep \_topdir

如果不想使用默认值，可以在~/.rpmmacros手工定义\_topdir

确定我们在 哪个目录下制作RPM，

通常这个目录我们topdir,这个需要在宏配置文件中指定，这个配置文件称为macrofiles，它们通常为 /usr/lib/rpm/macros:/usr/lib/rpm/macros.\*:~/.rpmmacros,这个在rhel 5.8中可以通过rpmbuild --showrc | grep macrofiles 查看，6.3的我使用这个找不到，但使用是一样的。你可以通过rpmbuild --showrc | grep topdir 查看你系统默认的工作车间



我们还是自定义工作目录(或车间)吧

vi ~/.rpmmacros

%\_topdir /home/ibuler/rpmbuild ##目录可以自定义

##### 3. 手工建立topdir及子目录

mkdir ~/rpmbuild

cd ~/rpmbuild

mkdir -pv {BUILD,BUILDROOT,RPMS,SOURCES,SPECS,SRPMS}

##### 4.把收集的源码放到SOURCES下

cp /tmp/tengine-1.4.2.tar.gz SOURCES ##事先放好的

##### 5.在SPECS下建立重要的spec文件

cd SPECS

vi tengine.spec #内容稍后

6.用rpmbuild命令制作rpm包，rpmbuild命令会根据spec文件来生成rpm包

rpmbuild

-ba 既生成src.rpm又生成二进制rpm

-bs 只生成src的rpm

-bb 只生二进制的rpm

-bp 执行到pre

-bc 执行到 build段

-bi 执行install段

-bl 检测有文件没包含

我们可以一步步试，先rpmbuild -bp ,再-bc 再-bi 如果没问题，rpmbuild -ba 生成src包与二进制包吧

7.安装和卸载测试

root用户测试安装:

rpm -ivh tengine-1.4.2-1.el6.x86\_64.rpm

rpm -e tengine

8.如果没问题为rpm包签名吧，防止有人恶意更改

到此一个简单的tengine RPM包制作好了。

#### 3.5 RPM包制作拓展

拓展计划：

增加服务脚本

更换默认首页index.html

fastCGI配置文件fastcgi\_params (默认的fastcgi\_params是不能直接连接php的)

更换默认配置文件

1 将修改过的index.html, init.nginx, fastcgi\_params, nginx.conf置于SOURCES

[ibuler@ng1 rpmbuild]$ ls SOURCES/

fastcgi\_params index.html init.nginx nginx.conf tengine-1.4.2.tar.gz

2 编辑tengine.spec，在3.2示例基础上增加：

完整的spec文件：



2.1 介绍区域的SOURCE0下增加如下

Source0: %{name}-%{version}.tar.gz

Source1: index.html

Source2: init.nginx

Source3: fastcgi\_params

Source4: nginx.conf

2.2 安装区域增加如下

make install DESTDIR=%{buildroot}

#%{\_\_install}这个宏代表install命令

%{\_\_install} -p -D %{SOURCE1} %{buildroot}/usr/html/index.html

%{\_\_install} -p -D -m 0755 %{SOURCE2} %{buildroot}/etc/rc.d/init.d/nginx

%{\_\_install} -p -D %{SOURCE3} %{buildroot}/etc/nginx/fastcgi\_params

%{\_\_install} -p -D %{SOURCE4} %{buildroot}/etc/nginx/nginx.conf

2.3 脚本区域增加如下

%post

if [ $1 == 1 ];then

/sbin/chkconfig --add nginx

fi

2.4 %file区域增加如下

%files

%defattr (-,root,root,0755)

/etc/

/usr/

/var/

%config(noreplace) /etc/nginx/nginx.conf #%config表明这是个配置文件noplace表明不能替换

%config(noreplace) /etc/nginx/fastcgi\_params

%doc /usr/html/index.html #%doc表明这个是文档

%attr(0755,root,root) /etc/rc.d/init.d/nginx #%attr后面的是权限，属主，属组

3. 生成rpm文件测试

rpmbuild -ba tengine.spec

4. 安装测试

到此RPM包制作完毕，你可以根据你的需求制作RPM包吧。

#### 3.6 RPM包签名

https://blog.csdn.net/Michaelwubo/article/details/105843605

生成秘钥过程，由于gpg在生成密码会卡死，需要很久的一段时间随机串，为了加快有两种办法

1、https://blog.csdn.net/ck784101777/article/details/101212318

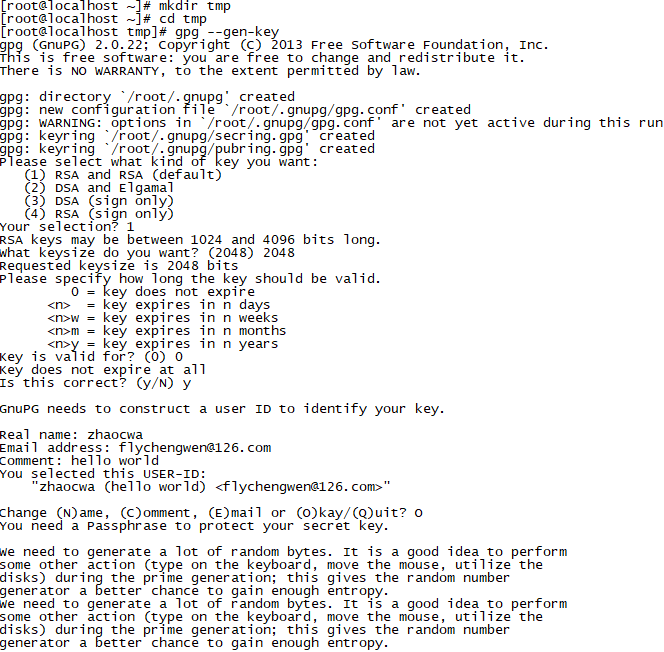
mv /dev/random /dev/random.bak

ln -s /dev/urandom /dev/random

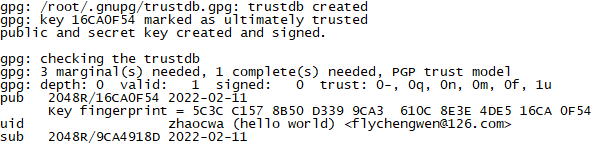
2、yum install rng-tools

rng -r /dev/urandom

3、[root@localhost noarch]# gpg --gen-key

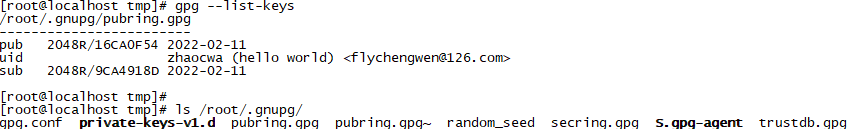


（漫长的等待过程）



4、获取列表

[root@localhost noarch]# gpg --list-keys



第一行显示公钥文件和所在的位置。

pub 行描述的是公钥大小（2048）／公钥 id（16CA0F54），公钥产生日期（2022-02-11）。

uid 行描述的是由名称、备注和邮件地址组成的字符串。

sub 行表述的是公钥的子钥（可以不用关心）。

5、导出公钥以供大家使用验证

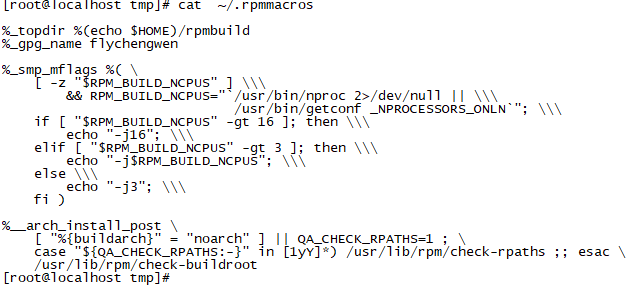
gpg --export -a "flychengwen" > RPM-GPG-KEY-flychengwen



6、编缉 .rpmmacros说明我们用哪一个密钥加密,我们用root加密的那就在/root下编辑

[root@localhost noarch]# cat ~/.rpmmacros

新增一行：%\_gpg\_name flychengwe



%\_topdir %(echo $PWD)/rpmbuild

%\_gpg\_name wu\_bo

%\_smp\_mflags %( \

[ -z "$RPM\_BUILD\_NCPUS" ] \\\

&& RPM\_BUILD\_NCPUS="`/usr/bin/nproc 2>/dev/null || \\\

/usr/bin/getconf \_NPROCESSORS\_ONLN`"; \\\

if [ "$RPM\_BUILD\_NCPUS" -gt 16 ]; then \\\

echo "-j16"; \\\

elif [ "$RPM\_BUILD\_NCPUS" -gt 3 ]; then \\\

echo "-j$RPM\_BUILD\_NCPUS"; \\\

else \\\

echo "-j3"; \\\

fi )

%\_\_arch\_install\_post \

[ "%{buildarch}" = "noarch" ] || QA\_CHECK\_RPATHS=1 ; \

case "${QA\_CHECK\_RPATHS:-}" in [1yY]\*) /usr/lib/rpm/check-rpaths ;; esac \

/usr/lib/rpm/check-buildroot

7、为rpm包加签名

yum install rpm-sign -y



到此签名添加成功，下面来验证

8、讲刚才导出的公钥导入rpm中

rpm --import RPM-GPG-KEY-flychengwen

9、验证

[root@localhost x86\_64]# rpm --checksig kernel-lt-4.14.141-1.el7.centos.x86\_64.rpm

|  |  |
| --- | --- |
| 签名前输出 | kernel-lt-4.14.141-1.el7.centos.x86\_64.rpm: sha1 md5 OK |
| 签名后输出 | kernel-lt-4.14.141-1.el7.centos.x86\_64.rpm: rsa sha1 (md5) pgp md5 OK |

#### 3.7 rpm发布及yum安装

##### rpm发布

https://blog.csdn.net/wutangkafei1990/article/details/49449537/

sudo yum install createrepo httpd -y 或sudo yum install createrepo nginx -y

https://blog.csdn.net/Michaelwubo/article/details/105878807

(创建本地源repodata作为软件的仓库)

nginx 的配置文件nginx.conf

location /{

#echo\_sleep 10;

#root html;

root html/centos7/x86\_64/;

autoindex on;

autoindex\_exact\_size off;

autoindex\_localtime on;

#index index.html;

#echo\_sleep 2;

}

/usr/local/nginx/sbin/nginx -c /usr/local/nginx/conf/nginx.conf

#首次创建仓库

createrepo -v /usr/local/nginx/html/centos7/x86\_64/或

createrepo -pdo /usr/local/nginx/html/centos7/x86\_64/ /usr/local/nginx/html/centos7/x86\_64/

将file\_tree-1.0-1.el6.x86\_64.rpm和RPM-GPG-KEY-wubo拷贝进仓库

[root@localhost aaa]# ls /usr/local/nginx/html/centos7/x86\_64/

RPM-GPG-KEY-wubo

file\_tree-1.0-1.el7.x86\_64.rpm

file\_tree-debuginfo-1.0-1.el7.x86\_64.rpm

repodata

第二次：createrepo --update /var/ftp/pub/testing/ 对http服务器上的yum源进行更新，每次有包的变动都要更新，这样客户才会知道新包的存在。

##### yum安装

加入yum源：

[yum]

name=yum

baseurl=http://ip:port/pub/testing #本地file:///var/www/html/repo/

enabled=1

gpgkey=http://ip:port/pub/testing/RPM-GPG-KEY-wubo

#gpgkey=file:///var/www/html/repo/RPM-GPG-KEY-wubo

#yum clean all ，清空yum缓存

#yum makecache #重建yum缓存

#yum update #更新yum源

rpm --import /var/www/html/repo/RPM-GPG-KEY-wubo #必须执行

yum repolist #查看yum源中是否有软件包

yum list

sudo yum install file\_tree -y

##### /etc/yum.repos.d/xxx.repo格式说明

[base]：容器名称，一定要放在[]中。

name：容器说明，可以自己随便写。

mirrorlist：镜像站点，这个可以注释掉。

baseurl：默认是 CentOS 官方的 yum 源服务器，如果慢可以改成华为源或阿里源。

enabled：此容器是否生效，缺省或写成 enabled 表示此容器生效；enable=0 则表示不生效。

gpgcheck：1：表示 RPM 的数字证书生效；0：表示 RPM 的数字证书不生效。

gpgkey：数字证书的公钥文件保存位置

baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/

$releasever表示版本号，例如7.9.2009

$basearch表示架构，例如x86\_64

[root@localhost ~]# cat /etc/redhat-release

CentOS Linux release 7.9.2009 (Core)

[root@localhost ~]# arch

x86\_64

https://blog.csdn.net/weixin\_34107739/article/details/93128861

**使用rpmbuild制作签名RPM包**

每个 RPM 软件包由三个基本组件组成:

1. 元数据 – 关于软件包的数据:软件包名称、版本、发布、构建程序、日期、依赖关系等。

2. 文件 – 软件包提供的文件存档(包括文件属性)，存储为cpio压缩包，rpm2cpio \*.rpm | cpio -div可以抽取

3. 脚本 – 安装、更新和 / 或删除软件包时,执行这些脚本

Spec文件是包含关于如何构建可安装的RPM软件包的信息的文本文件。其大致分为五部分:

1. 列出关于软件包的元数据(名称、版本、许可证等)

2. 构建说明,详细说明如何编写和准备软件

3. 脚本小程序,详细说明安装、卸载或升级时要运行的命令

4. 清单,软件包文件列表及其关于软件包安装的权限

5. changelog ,记录对此RPM软件包所做的更改

rpmbuild 步骤：

1. 准备

2. 构建

3. 安装

4. 打包完成的 RPM

5. 清理

## 举例

### 1)相关目录

environment：RHEL6.2

~/rpmbuild/SOURCES/ 存放源码压缩包和补丁文件

~/rpmbuild/BUILD/ 存放源码

~/rpmbuild/SPECS/ 存放spec文件

~/rpmbuild/RPMS/ 最终RPM包

相关命令：rpmbuild

### 2)准备GPG签名密钥

sudo yum install pinentry-gui -y

gpg --gen-key

gpg --list-key

/home/kevin/.gnupg/pubring.gpg

------------------------------

pub 2048R/8156471D 2012-03-07

uid kevin (kevin) <kevin@kevin.org>

sub 2048R/641335E8 2012-03-07

#导出公共密钥：

gpg -a -o ~/RPM-GPG-KEY-kevin --export 8156471D

#将以下内容添加到 ~/.rpmmacros 文件(用系统的密钥 ID 替换该八个字符密钥 ID ),因此RPM 将用您在上面创建的密钥签署软件包。

echo '%\_gpg\_name 8156471D' > ~/.rpmmacros

### 3) 编辑file\_tree和spec

#file\_tree是个shell脚本，模拟tree(shell命令)的功能。

#制作file\_tree-1.0.tar.gz

[kevin@desktop25 work]$ chmod +x file\_tree

[kevin@desktop25 work]$ mkdir file\_tree-1.0

[kevin@desktop25 work]$ mv file\_tree file\_tree-1.0/

[kevin@desktop25 work]$ tar -zcf file\_tree-1.0.tar.gz file\_tree-1.0/

[kevin@desktop25 work]$ vim file\_tree.spec

#将tar和spec拷贝到rpmbuild目录下

[kevin@desktop25 work]$ sudo yum install rpmbuild -y

[kevin@desktop25 work]$ rpmbuild ~

[kevin@desktop25 work]$ cp file\_tree.spec ~/rpmbuild/SPECS/

[kevin@desktop25 work]$ cp file\_tree-1.0.tar.gz ~/rpmbuild/SOURCES/

[kevin@desktop25 work]$ cd ~/rpmbuild/SPECS/

rpm build期间可以指定--sign来直接得到签名了的rpm包

rpmbuild --sign -ba file\_tree.spec 或

rpm --addsign file\_tree-1.0.1.rpm

rpm --resign file\_tree-1.0.1.rpm均可

注意：无论是先打包后签名还是打包时直接签名, 验签时都显示"pgp 确定"

对未签名的rpm包进行签名,签名后的文件与签名前的相比只是在文件头部(添加)修改了一些信息, 签名后大了344个字节。

最终生成带签名的rpm包file\_tree-1.0-1.el6.x86\_64.rpm

### 4）安装测试

通过安装密钥和软件包并运行命令来测试软件包:

[kevin@desktop25 x86\_64]$ sudo rpm --import ~/RPM-GPG-KEY-kevin

[kevin@desktop25 x86\_64]$ cd ~/rpmbuild/RPMS/x86\_64/

[kevin@desktop25 x86\_64]$ rpm -ivh file\_tree-1.0-1.el6.x86\_64.rpm

[kevin@desktop25 x86\_64]$ file\_tree(输出略)

### 5）发布RPM软件包

[kevin@desktop25 x86\_64]$ sudo yum install createrepo -y

（创建本地源repodata作为软件的仓库）

[kevin@desktop25 x86\_64]$ sudo mkdir /var/ftp/pub/testing

[kevin@desktop25 x86\_64]$ sudo cp file\_tree-1.0-1.el6.x86\_64.rpm /var/ftp/pub/testing/

[kevin@desktop25 x86\_64]$ sudo createrepo -v /var/ftp/pub/testing/

1/1 - file\_tree-1.0-1.el6.x86\_64.rpm

[kevin@desktop25 x86\_64]$ sudo cp ~/RPM-GPG-KEY-kevin /var/ftp/pub/testing/

### 6) 加入yum源

[yum]

name=yum

baseurl=http://192.168.0.25/pub/testing

enabled=1

gpgkey=http://192.168.0.25/pub/testing/RPM-GPG-KEY-kevin

验证：

[kevin@desktop20 ~]$ sudo yum install file\_tree -y

OK，发布成功～

https://gist.github.com/fernandoaleman/1376720

# 签名打包整体流程

# Step1: gpg --gen-key

# Step2: gpg --list-keys #验证gpg keys创建成功

# Export your public key from your key ring to a text file.

# Step3: gpg --export -a 'Fernando Aleman' > RPM-GPG-KEY-wu\_bo3

# Step4: rpm --import RPM-GPG-KEY-wu\_bo3

# Verify the list of gpg public keys in RPM DB

# Step5: rpm -q gpg-pubkey --qf '%{name}-%{version}-%{release} --> %{summary}\n'

# Configure your ~/.rpmmacros file

# Step6: vi ~/.rpmmacros

# %\_signature => This will always be gpg

# %\_gpg\_path => Enter full path to .gnupg in your home directory

# %\_gpg\_name => Use the Real Name you used to create your key

# %\_gpbin => run `which gpg` (without ` marks) to get full path

%\_signature gpg

%\_gpg\_path /root/.gnupg

%\_gpg\_name wu\_bo3

%\_gpgbin /usr/bin/gpg

# Step7: rpm --addsign \*.rpm

# Step8: rpm --checksig file\_tree-1.0.el7.x86\_64.rpm

#rpm build期间可以指定--sign来直接得到签名后的rpm包

rpmbuild -ba --sign file\_tree.spec

## 签名打包流程举例

### 制作rpm包的服务器

gpg --gen-key

gpg --list-keys

gpg --export -a 2D50D623 > RPM-GPG-KEY

vim ~/.rpmmacros

%\_signature gpg

%\_gpg\_path /root/.gnupg

%\_gpg\_name Fernando Aleman

%\_gpgbin /usr/bin/gpg

# root下对未签名的rpm包进行签名,签名后的文件与签名前的相比只是在文件头部(添加)

#修改了一些信息, 签名后大了344个字节

rpm --addsign git-1.7.7.3-1.el6.x86\_64.rpm

#rpm build期间可以指定--sign来直接得到签名了的rpm包

rpmbuild -ba --sign SPECS/hello.spec

### 下载rpm包的节点

1)导入发布者公钥RPM-GPG-KEY

#成功导入公钥后才能验签 (观察'gpg OK')---先打包后签名的rpm包

sudo rpm --import RPM-GPG-KEY

2)Verify the list of gpg public keys in RPM DB。查看当前服务器导入了那些gpg公钥信息

rpm -q gpg-pubkey --qf '%{name}-%{version}-%{release} --> %{summary}\n'

3) rpm --checksig /usr/local/hello-0.1-1.x86\_64.rpm