GNU/Linux
软件包安装

对 src.rpm 包重新打包 -rpmbuild

RPM 软件包的构成

元数据 : 关于软件包的数据

包含 : 软件包名称、版本、发布、构建程序、日期、依赖关系等

文件 : 软件包提供的文件 (包括文件属性)

脚本 : 安装软件、更新或删除软件包时 , 所执行的脚本

对 src.rpm 包重新打包 -rpmbuild

RPM 软件包规则 :

如由源代码构建 rpm 包 , 则需要 spec 文件。通过 spec 文件所包含的配置 , 来完成如何构建 RPM 包

对 src.rpm 包重新打包 -rpmbuild

RPM 软件包规则 :

spec 文件大致分为 5 个部分

1. 简介 : 列出关于软件包的元数据 (名称、版本、许可证等)

2. 构建说明 : 说明如何编写和准备软件

对 src.rpm 包重新打包 -rpmbuild

RPM 软件包规则 :

3. 脚本小程序 : 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

对 src.rpm 包重新打包 -rpmbuild

重新构建 rpm 包

1. 确认拥有 1 个 src.rpm 包

2. 安装 xxx.src.rpm 包

   #rpm -ivh xxx.src.rpm

3. 进入当前目录下的 rpmbuild/SPECS 目录

   #cd ~/rpmbuild/SPECS

对 src.rpm 包重新打包 -rpmbuild

重新构建 rpm 包

4. 查看此软件的 spec 文件

5. spec 文件有几个段落分别为
   %prep: 安装前脚本
   %build: 构建用的脚本
   %install: 安装用的脚本
   %clean: 安装后清理用的脚本

对 src.rpm 包重新打包 -rpmbuild

6. 确认 rpmbuild/gcc 已经存在，如果不存在请安装 rpm-build/gcc 软件包

7. 建立新的 RPM 包

```
#rpmbuild -ba  xxx.spec
```

8. 进入建立好的“ RPMS/ 你的架构”目录，找到已经完成的 rpm

```
#cd ~/rpmbuild/RPMS/x86_64/
#ls
```

对 src.rpm 包重新打包 -rpmbuild

9. 进入建立好的“ SRPMS/ 你的架构”目录 ,找到已经完成的 src.rpm

```
#cd ~/rpmbuild/SRPMS/x86_64/
#ls
```

对 src.rpm 包重新打包 -rpmbuild

命令 :rpmbuild

功能 : 建立新的 RPM/SRPM 包

语法格式 :rpmbuild < 选项 > < 软件的 .spec>

对 src.rpm 包重新打包 -rpmbuild

选项 :

-bp: 只做准备工作 (解包及打补丁)

-bc: 准备环境并编译

-bi: 编译并安装

-bl: 检测文件是否齐备

对 src.rpm 包重新打包 -rpmbuild

选项 :

-ba: 生成 rpm 及 src.rpm 包

-bb: 仅生成 rpm 包

-bs: 仅生成 src.rpm 包

发布新软件包

1. 生成数字签名 (gpg)
   1) 确认有可用的 gpg
      #gpg --list-key

   2) 如果没有，则生成新的 gpg
      #gpg --gen-key
         根据步骤完成 gpg 秘钥

发布新软件包

1. 生成数字签名 (gpg)

   3) 确认 gpg 的公钥以便于数字签署

   #gpg --list-key

   /root/.gnupg/pubring.gpg

   ---

   pub 2048R/0A86BCE3 2012-12-12

   uid chuai <root@localhost>

   sub 2048R/6F495C7A 2012-1212

发布新软件包

1. 生成数字签名 (gpg)

4) 将公钥以 ASCII 导出成为 RPM 的 GPG 签名

#gpg –a –o ~/RPM-GPG-KEY-test --export 0A86BCE3

5) 建立 .rpmmacros 添加公钥

#vim ~/.rpmmacros

内容如下

%_gpg_name 0A86BCE3”

发布新软件包

1. 生成数字签名 (gpg)

   6) 对 xx.rpm 软件签名 (需 rpm-sign 软件包)

   #rpm --resign xxx.rpm

   7) 将 RPM-GPG 导入至 RPM 数据库中

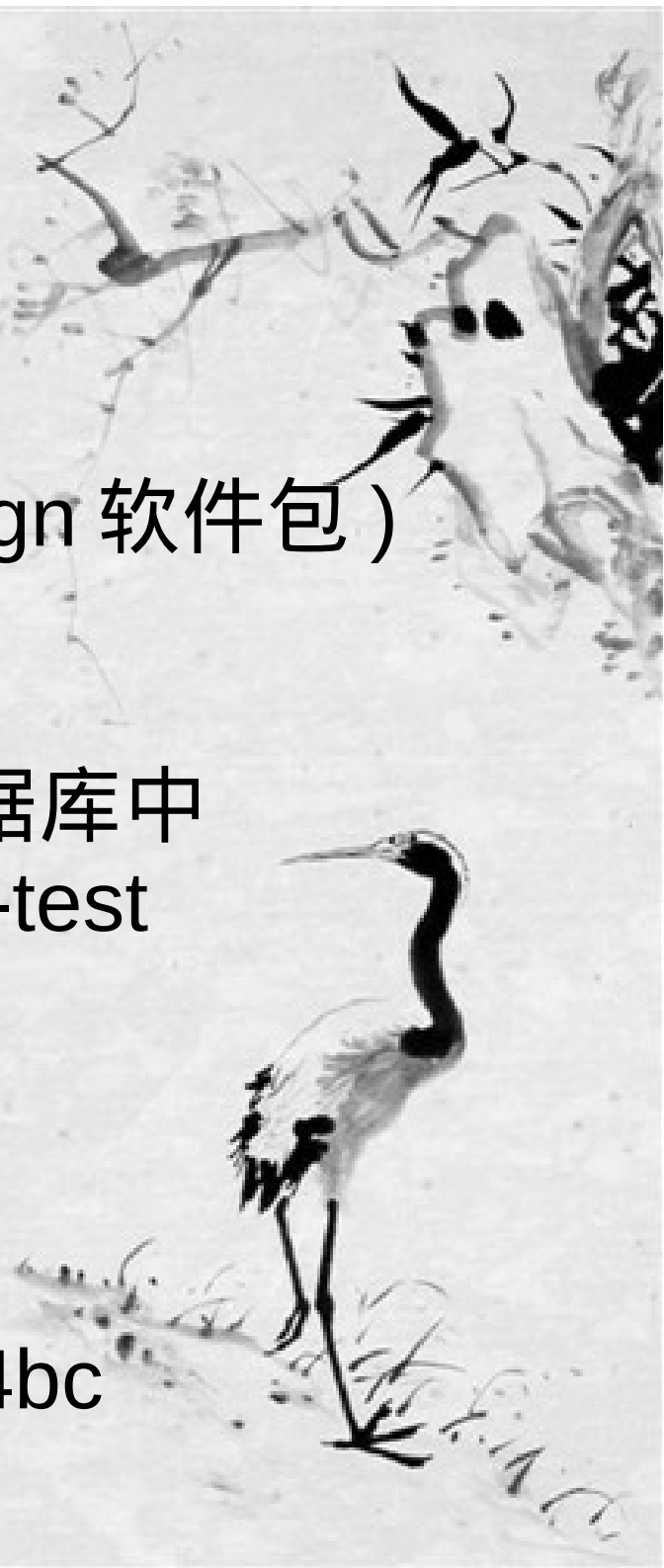
   #rpm --import RPM-GPG-KEY-test

   8) 检查导入是否成功

   #rpm –q gpg-pubkey-*

   .....

   gpg-pubkey-0a86bce3-52b024bc



发布新软件包

1. 生成数字签名 (gpg)

   9) 确认 RPM 包签名正确

   #rpm –K xxx.rpm

   xxx.rpm: rsa sha1 (md5) pgp md5 OK

   如果出现错误请重新生成签名

   10) 将软件包放在指定目录

   #cd /var/ftp/pub/Packgaes

   #cp ~/xxx.rpm ./

   #cp ~/RPM-GPG-KEY-test ./

发布新软件包

2. 发布软件到自定义 YUM 源中

11) 创建 YUM 源

#createrepo –v /var/ftp/pub/Packages

#cd /var/ftp/pub/Packages

#ls –ld repodata

发布新软件包

3. 客户端操作

1) 建立 YUM 源

2) 通过服务器下载 RPM-GPG-KEY-test

3) 客户端导入 RPM-GPG-KEY-test

```
#rpm  --import RPM-GPG-KEY-test
```

4) 测试

将 RPM 包转化为 CPIO-rpm2cpio

命令 :rpm2cpio

功能 : 将 RPM 包转化为 CPIO 归档格式

语法格式 :rpm2cpio [文件名]

将 RPM 包转化为 CPIO-rpm2cpio

示例 :

1. 将 xxx.rpm 文件转换为 cpio 格式并提取其内容

   #rpm2cpio xxx.rpm | cpio -idv

   cpio 参数 :

   -i: 进入 copy-in 模式

   -d: 自动建立目录

   -v: 显示动作

将 RPM 包转化为 CPIO-rpm2cpio

示例 :

2. 将 xxx.rpm 文件转换为 cpio 格式

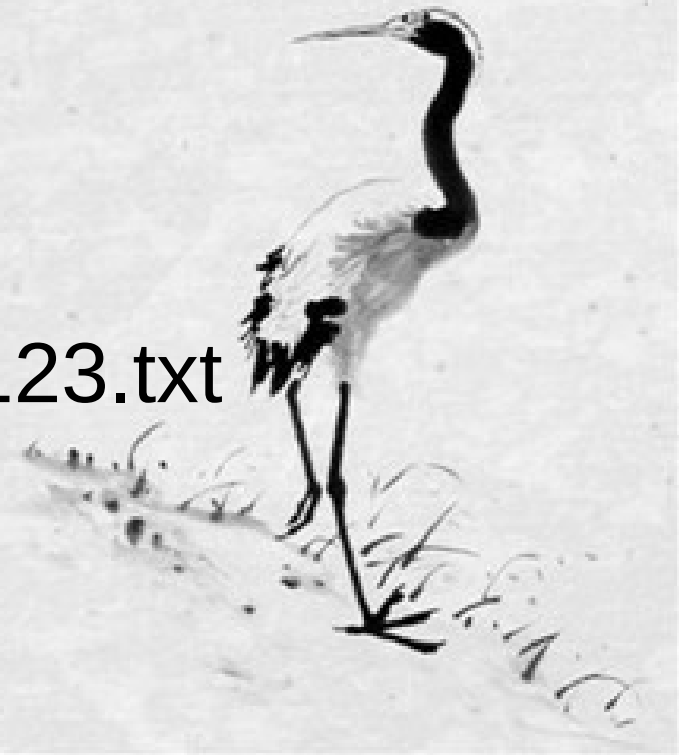
   #rpm2cpio xxx.rpm > xxx.cpio

3. 查看 xxx.rpm 文件中的内容

   #rpm2cpio xxx.rpm | cpio -t

4. 仅提取指定文件 123.txt

   #rpm2cpio xxx.rpm | cpio -idv ./123.txt



软件包管理 -Source

源代码安装

特点 :

1. 可以高度自定义
2. 安装时间较长 , 软件越大安装时间越长
3. 删除比较麻烦

软件包管理 -Source

源代码安装三部曲

1. 解压源代码包
   #tar xvfj xxx.tar.bz2

2. 进入源代码包的目录

3. 检测编译环境
   $./configure

软件包管理 -Source

源代码安装三部曲

4. 编译源代码
   $make

5. 安装源代码
   #make install

软件包管理 -Source

源代码安装

对于 ./configure 可以有许多的参数，详情可以看
$./configure --help | less

如

指定源代码在安装时指定安装到 /usr/local 目录
$./configure --prefix /usr/local

软件包管理 -Source

源代码删除

1. 查看源代码目录中有没有 uninstall 或类似功能的文件

2. 可以一步步删除

软件包管理 - 其他 2 进制包

在软件包中不乏有其他后缀的名字的 2 进制软件包。他们的安装基本上会更加方便．如：

1. bin
2. run
3. bundle

……

安装方法

对程序增加执行权限即可执行，完成安装