1 引言

通常由于某种实际应用，需要一个包含所有最近更新的 RPM 包的操作系统发布盘，以备在安装时一次完成所有的更新操作，或者是想定制一个有自己特色的操作系统发布盘，如将自己开发的应用程序通过创建 RPM 包，加入到操作系统中，在系统安装时一次完成，形成包含自己产品的操作系统发布盘。这些都需要重新生成安装盘，而且生成安装盘也是十分必要的，因为操作系统发布商在每一次正式发布后，总会对一些漏洞进行更新处理，有些还是与安全相关的，在重新生成安装盘时就可以将这些 bug 修复添加进你自己定制的安装盘中，对一些设备新开发的驱动程序提供支持也需要重新生成安装盘。

在一些嵌入式具体应用中，由于其对操作系统的要求较具体，不需要当前操作系统安装盘中自带的那么多的功能， 如 Fedora Core 2当前有4张安装盘，它包含了许多其它的应用，如 office、娱乐和游戏等等，而一些具体的应用根本不需要这么多的功能，因此，它们常常需要基于一个版本的操作系统，然后对之进行相应的裁减，使之能满足具体应用的实际需要，而不需要其它的多余的功能。因此，通过操作系统安装盘的定制，可以根据自己或实际的需要，选择有用的软件包，组成安装盘，从而通过定制操作系统的安装，满足具体应用的需要。

我们在定制操作系统安装盘之前，必须有一个蓝本作为安装盘的基础，比如是 Red Hat 9.0安装盘或 Fedora Core 2安装盘，也可以是 Red Hat 9.0或 Fedora Core 2安装盘的 iso 文件，这些我们可以从 Red Hat 的网站或其它一些网站上下载。现假设我们已经有了 Fedora Core 2的安装盘，下面我们先大略看一下 Fedora Core 2的安装盘里面的内容。

在安装盘中有一个目录为 Fedora,它包含了发布盘的核心内容，如下：

```
drwxr-xr-x    2 root     root         2048 May 13 2004 base
drwxr-xr-x    2 root     root        77824 May 13 2004 RPMS
```

RPMS 目录包含 Fedora Core 2发布盘的主要部分，它是一些 RPM 文件。RPM 包通常包含二进制可执行文件、有关的配置文件和文档，我们可以参考 RPM 帮助以获得更多信息。

base 目录中包含一些在安装过程中所需要的文件，如 comps.xml 文件，它定义哪个组件包含哪些 RPM 包以及 RPM 包之间的依赖关系，需要注意的是，在 comps.xml 文件中表示哪个组件有哪些 RPM 包采用的是 RPM 包名，而不是包的文件名。比如 perl-5.8.3-18.i386.rpm 这个文件名，在 comps.xml 中所表示的 RPM 包名为 perl。对于 comps.xml 文件，我们会在后面作进一步解释。另一个重要的文件是 hdlist 文件，它包含了 RPM 目录中的所有 RPM 包大部分的头文件，这意味着在 RPM 包中相互依赖关系可以通过读取 hdlist 文件而决定，而不需要读所有的 RPM 包。hdlist 文件的另一个作用是将包名映射到文件名，如将 perl 包名映射到 perl-5.8.3-18.i386.rpm，这意味着如果你想更新 RPM 包或添加你自己的包到 RPM 目录中，你就需要更新 hdlist 这个文件，这会在后面进行描述。

回页首

2 RPM 操作

RPM（Redhat Package Management）是由 RedHat 开发的，在 Linux 系统下的系统包管理工具。它的目标是：使包的安装和卸载过程更容易，它能够证实一个包是否已经正确安装了，可以简化包的建立过程，可以从源代码建立整个包，它能用于不同的体系结构。RPM 系统已经成为现在 Linux 系统下包管理工具事实上的标准，并且它也移植到很多商业的 unix 系统之下。

RPM 包由包标签对它标识，包标签包含软件名，软件版本，包的发行版本几部分。在包的内部还包含包的建立时间，包的内容描述，安装包的所有文件的大小，数字签名以证实包的完整性等信息。 RMP 包还包含包内的文件信息，其中包括：每个文件的文件名，每个文件的权限，文件的属组和拥有者，每个文件的md5校验和，文件的内容等。

RPM 包管理系统提供了下列功能：安装新的包，卸载旧的包，将一个旧包升级为新的包，获得已经安装包的信息等。

Red Hat 发布盘主要是由一些 RPM 包组成。RPM 包的名字包含一个后缀：arch.rpm，arch 指的是体系结构，对于 Intel

平台的有 i386、i586、i686等，你所安装的包必须要与机器上的共享库的版本相匹配。如果你发现某个 RPM 包没有安装，你可以自己安装。任何时候，你都可以（必须是root 用户）安装 RPM 包。RPM 命令使用轻参考相关资料。

回页首

3 RPM 包创建过程

为了完成 RPM 包的创建，需要执行以下步骤：

- 执行 spec 文件 prep 节的命令和宏；
- 检查文件列表的内容；
- 执行 spec 文件 build 节的命令和宏；
- 执行 spec 文件 install 节的命令和宏，同时也执行文件列表中的宏；
- 创建二进制包文件；
- 创建源码包。

为了执行打包的工作，RPM 需要一系列目录完成建立的工作。正常的目录结构通常由一个顶级目录和五个子目录构成。这五个子目录分别是：

1 SOURCES------包含原始的源文件和补丁文件。
2 SPECS--------包含控制 RPM 包建立过程的 spec 文件。
3 BUILD--------包含源码解包和软件建立的目录。
4 RPMS---------包含建立过程创建的二进制包文件。
5 SRPMS--------包含建立过程创建的源码包文件。

除了上述这五个主要的目录外，在 RPMS 或 SRPMS 目录下通常还会有关于 RPM 包目标平台的目录。例如，i386、i586、i686等代表与 Intel 兼容 cpu 的平台，noarch 目录下的 RPM 包代表可以在任何平台下执行。

3.1 SPEC 文件

spec 文件是整个 RPM 包建立过程的中心，它的作用就如同编译程序时的 Makefile 文件。spec 文件包含建立一个 RPM 包必需的信息，包括哪些文件是包的一部分以及它们安装在哪个目录下。这个文件一般分为如下的几节：

(1) Preamle（序言）

序言包含用户请求包的信息时所显示的内容。它可以包含包的功能描述、包的软件版本、版权信息和所属的包组等。Summary 是一行关于该软件包的描述，Name 是该软件包的基名，Version 是该软件的版本号，Release 是 RPM 本身的版本号，如果修复了 spec 文件中的一个错误并发布了该软件同一版本的新 RPM，就应该增加发行版号。License 应该给出一些许可术语（如："GPL"、"Commercial"、"Shareware"），Group 标识软件类型。那些试图帮助人们管理 RPM 的程序通常按照组列出 RPM。您可以在 usr/share/doc/rpm-4.0.4/GROUPS 文件看到一个 Red Hat 使用的组列表（假设您安装的 RPM 版本是 4.0.4）。但是您还可以使用那些组名以外的名称。Source0、Source1等等给这些源文件命名（通常为 tar.gz 文件）。%{name} 和 %{version} 是 RPM 宏，它们扩展成为头中定义的 rpm 名称和版本。

要注意的是，你不要在 Source 语句中包含任何路径。缺省情况下，RPM 会在 /usr/src/redhat/SOURCES 中寻找文件，请将您的源文件复制或链接到那里。（要使 spec 文件尽量可移植的话，应当尽量避免嵌入自己开发机器上的假想路径。其他开发人员就可以指示 RPM 在别的目录下查找源文件，而不用修改您的 spec 文件。）

接下来的部分从 %description 行开始。您应该在这里提供该软件更多的描述，这样任何人使用 rpm -qi 查询您的软件包时都可以看到它。您可以解释这个软件包做什么，描述任何警告或附加的配置指令，等等。

(2) Prep 节

Prep 节进行实际的打包准备工作，它是使用节前缀%prep 表示的。一般而言，这一节的主要工作是检查标签语法是否正确，删除旧的软件源程序，对包含源程序的 tar 文件进行解码。如果包含补丁（patch）文件，将补丁文件应用到解开的源码中。它一般包含%setup 与%patch 两个命令。%setup 用于将软件源码包解开，执行%patch 可将补丁文件加入解开的源程序中。

%setup

-n newdir---------将压缩的软件源程序在 newdir 目录下解开。

-c ----------在解开源程序之前先创建目录。

-b num----------在包含多个源程序时，将第 num 个源程序解压缩。

-T----------不使用缺省的解压缩操作。

例如：

%setup -T -b 0

/*解开第一个源程序文件。*/

%setup -c -n newdir

/*创建目录 newdir，并在此目录之下解开源程序。*/

%patch

%patchN-------这里 N 是数字，表示使用第 N 个补丁文件，等价于%patch -P N

-p0----------指定使用第一个补丁文件，-p1指定使用第二个补丁文件。 -s----------在使用补丁时，不显示任何信息。

-b name-------在加入补丁文件之前，将源文件名上加入 name。若为指定此参数，则缺省源文件加入.orig。

-T----------将所有打补丁时产生的输出文件删除。

(3) Build 节

这一节主要用于编译源码，它是使用节前缀%build 表示的。这一节一般由多个 make 命令组成。

(4) Install 节

这一节主要用于完成实际安装软件必须执行的命令，它是使用节前缀 %install 表示的。这一节一般是由 make install 指令构成，但是有时也会包含 cp、mv、install 等指令。

这一节还能指定在用户安装的系统上，包安装时运行的脚本。这样的脚本称为安装（卸载）脚本。它可以指定包安装前、包安装后、包除去前、包除去后的系统必须运行的外壳程序段。在用户安装的系统上，为了验证一个包是否已经成功安装的验证脚本也可由这一节指定。

(5) Clean 节

这一节所描述的内容表示在完成包建立的工作之后，自动执行此节下的脚本进行附加的清除工作，它是使用节前缀 %clean 表示的。一般而言，这一节的内容是简单地使用 rm -rf $RPM_BUILD_ROOT 命令，不需要指定此节的其它内容。

(6) 文件列表

这一节指定构成包的文件的列表，它是使用节前缀%files 表示的。此外，它还包含一系列宏控制安装后的文件属性和配置信息。

%files 列出应该捆绑到 RPM 中的文件，并能够可选地设置许可权和其它信息。在 %files 中，您可以使用 %defattr 来定义缺省的许可权、所有者和组；%defattr(-,root,root) 会安装 root 用户拥有的所有文件，使用当 RPM 从构建系统捆绑它们时它们所具有的任何许可权。

可以用 %attr(permissions,user,group) 覆盖个别文件的所有者和许可权。可以在 %files 中用一行包括多个文件。可以通过在行中添加 %doc 或 %config 来标记文件。%doc 告诉 RPM 这是一个文档文件，因此如果用户安装软件包时使用 --excludedocs，将不安装该文件。您也可以在 %doc 下不带路径列出文件名，RPM 会在构建目录下查找这些文件并在 RPM 文件中包括它们，并把它们安装到 /usr/share/doc/%{name}-%{version}。以 %doc 的形式包括 README 和 ChangeLog 这样的文件是个好主意。

%config 告诉 RPM 这是一个配置文件。在升级时，RPM 将会试图避免用 RPM 打包的缺省配置文件覆盖用户仔细修改过的配置。

注意：如果在 %files 下列出一个目录名，RPM 会包括该目录下的所有文件。通常这不是您想要的，特别对于 /bin 这样的目录。

(7) 改动日志

这一节主要描述软件的开发记录，它是使用节前缀%changlog 表示的。这个段的内容是为了开发人员能详细的了解该软件的开发过程，对于包的维护极有好处。

3.2 创建 RPM 包

如果我们需要对 RPM 包作修改，那么我们首先需要将源码包取来，比如我们要修改内核，那么我们可以从网上或光盘中取到内核的源代码 RPM 包，如 kernel-2.6.5-1.358.src.rpm，将源码包解开：rpm -i kernel-2. 6.5-1.358.src.rpm，则该 RPM 中的内容将存放在目录/usr/src/redhat/SOURCES 和/usr/src/redhat/SPEC 目录中，前者存放的是源码、补丁以及一些配置文件等，后者存放的是包对应的 spec 文件，如 kernel-2.6.spec，现在你就可以对内核进行修改。假定我们想另外再对内核打一个补丁，比如说：mypatch-2.6.5.patch，你需要将这个补丁文件复制到/usr/src/redhat/SOURCES/目录下，然后编辑 kernel-2.6.spec 文件。你需要先在定义补丁文件的最后加入对你补丁文件的初始定义，如：

```
……
Patch10000: linux-2.6.0-compile.patch
# Patch10010: linux-2.6.0-module-license.patch
Patch10030: mypatch-2.6.5.patch          /*新加入的补丁文件的定义*/
# END OF PATCH DEFINITIONS
……
```

然后在文件的后面加入对内核打补丁命令：

```
……
%patch10000 -p1
%patch10030 -p1          /*新加入的打补丁命令*/
# END OF PATCH APPLICATIONS
……
```

如果你还想对内核做其它的修改，你可以修改相应的文件或添加相应的文件，然后修改kernel-2.6.spec 文件。当 spec 文件修改完成之后，你只需要执行 rpmbuild -ba kernel-2.6.spec 就可以生成所需要的 RPM 包了。另外需要注意的是，以生成内核包为例，假如我们想生成 kernel-smp-2.6.5-1.358.i686.rpm 包，在 kernel-2.6.spec 文件中包含有一些开关选项，比如，在文件的开头需要定义创建哪些内核的RPM 包，如：

```
%define buildup   1
%define buildsmp   0
%define buildsource   1
```

在通常情况下，在执行 rpmbuild -ba kernel-2.6.spec 命令后，会创建一个 kernel-2.6.5-1.358.i386.rpm、kernel-source-2.6.5-1.358.i386.rpm 和源码 RPM 包 kernel-2.6.5-1.358.src.rpm。因此，当你需要创建支持 SMP 的内核的 RPM 包时，需要修改 kernel-2.6.spec 文件开头时的定义为：

```
%define buildup   1
```

```
%define buildsmp   1
%define buildsource   1
%define -target_cpu      i686
```

此外，在文件的开头还需要定义-target_cpu 为 i686，从而创建 i686的内核 RPM 包，并且需要对/usr/lib/rpm 目录下面的一些宏重新定义，比如当前目录下面的 macros 文件，需要重新定义 arch 和 build_arch 为 i686。最后，执行命令 rpmbuild -ba kernel-2.6.spec --with smp 就可以。当然，如果对内核进行了相应的修改，就必须生成多个内核 RPM 包，以适用于多个 arch，如 kernel-2.4.18-3-i586-smp.rpm, kernel-2.4.18-3-athlon.rpm 等。

回页首

4 操作系统安装盘的定制过程

你需要将原来操作系统发布盘上的内容拷贝到本机硬盘中，根据有几张发布盘而生成几个目录，比如 Fedora Core 2有四张盘，你则需要在系统上生成四个目录，如 disc1、disc2、disc3、disc4，分别将这四张盘上内容拷贝到这四个目录中，然后对相应的 RPM 包进行更新。

首先找到你想更新的 RPM 包，将新的 RPM 替换旧包。当然你也可以根据你的需要新增或删除 RPM 包，需要注意的是，你需要在 comps.xml 文件中将新增加或删除的 RPM 包名加入某个组件中，并且注意其与其它 RPM 包的依赖关系，也就是说你所放置的包的位置要恰当，否则它会依赖于在它之前而没有加入系统的某个RPM 包。

4.1 编辑 comps.xml 文件

在生成安装盘之前，需要注意对 comps.xml 文件进行修改。这个文件用来告知安装程序 anaconda，用户选择了某个组是应该有哪些包需要安装，定义了在安装过程中，包是如何被捆绑在一起的。在 Red Hat 8.0以前版本的发布盘中，对应的文件为 comps，它只是一个简单的文本文件，在 Red Hat 8.0之后的版本中，用 comps.xml 代替了原来的 comps 文件。comps.xml 是一个 XML 文件，易于对内容进行分析和说明。

comps.xml 文件开始是说明 xml 的版本和 DTD 断言，然后进入以<comps>标记开始的文件的主体内容。如：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE comps PUBLIC "-//Red Hat, Inc.//DTD Comps info//EN" "comps.dtd">
<comps>
```

comps.xml 主要由三部分组成，首先是组列表，它描述了在安装过程中需要的不同的组（或组件），包括组名、组的描述和包含的 RPM 包；其次是组的层次结构，它将组分成不同的类，并定义了组的一个顺序，从而可以决定哪些组需要先安装；最后为一系列 RPM 包以及它们之间的依赖关系。

下面分别介绍 comps.xml 文件的这三部分：

（1） 组列表

在系统安装时，需要用到一个组中的一些属性，下面就是属性列表以及它们如何使用。一个组被定义在 <group>和</group>标记之内.

一个简单的组定义可以是：

```
<group>
```

```
  <id>somegroup</id>
  <name>Sample Group</name>
  <default>true</default>
  <uservisible>false</uservisible>
  <description>This is a silly sample group</description>
  <packagelist>
    <packagereq type="mandatory">bash</packagereq>
    <packagereq type="default">cpio</packagereq>
  </packagelist>
</group>
```

下面分别说明组定义中一些参数的含义：

- id：组的 id 仅仅是在 comps.xml 文件中作为该组的一个标识，这是必须的；
- name：表示用户可以看到的组的名字，它也是必须的；
- default：它表示在系统安装过程中，当选择定制（custom）安装时，该组是否在缺省情况下被选中。如果没有说明，则缺省情况下为不选中。
- uservisible：它表示该组在缺省情况下是否在安装时可以看到，如果没有说明，缺省设置为 YES，为可以看到。
- description：它表示对该组进行简短的描述，这是必须的；
    - packagelist：它说明在该组内的一系列安装包，这也是必须的。

    packagereq：包名

    属性：

    type：当进行安装时，判定对应的包是否是组的"强制"部分、或"缺省"部分或"可选"部分。它可以是"mandatory"、"default"或"optional"之一。
    - requires：它说明只有当它所需要（依赖）的包也安装情况下，此包才安装进系统。

（2）组层次结构

它描述了组的树状层次结构，组层次结构定义在<grouphierarchy>和</grouphierarchy>标记之间，由定义的<category>标记组成类。

一个简单的组层次结构可以如下所述：

```
<grouphierarchy>
  <category>
    <name>Random Groups</name>
    <subcategories>
      <subcategory>somegroup</subcategory>
    </subcategories>
  </category>
</grouphierarchy>
```

一个类由下面这些属性组成：

- name: 它表示类名，是必须的；
- subcategories: 它表示此类的一些子类，由一列表<subcategory>和</subcategory> 元素组成
    - subcategory: 前面定义的组的 id

（3）RPM 包

此部分说明要安装的 RPM 包，它定义在<package>和</package>标记之内。一个简单的 RPM 包部分可以如：

```
<package>
  <name>bash</name>
  <dependencylist>
    <dependency>mktemp</dependency>
    <dependency>bash</dependency>
    <dependency>glibc</dependency>
    <dependency>libtermcap</dependency>
  </dependencylist>
</package>
```

- name：它指的是 RPM 包名，是必须的。
- dependencylist：它说明此包对应的依赖的 RPM 包。
    - dependency：此包需要的包的名字

4.2 产生完整的 comps.xml 文件

上述说明的 comps.xml 文件中的 RPM 包部分是是自动产生的，为了形成完全的 comps.xml 文件，需要在系统中安装 comps-extras RPM 包，然后进行下面的操作：

- 将 comps.xml 文件中的原来的 RPM 包部分删除；
- 运行：

  /usr/share/comps-extras/getfullcomps.py comps.xml /path/to/tree arch >/root/filelist 在此，/path/to/tree 是 Red Hat Linux 操作系统安装盘内容存储的地方，arch 指的是体系结构，为 i386。注意的是，假定 comps.xml 已经存放 /path/to/tree/arch/RedHat/base/目录下，将此输出重定向到一个临时文件，如/root/filelist。
- 将 comps.xml 文件中最后一行内容（为</comps>）删除
- 将前面生成的临时文件添加到 comps.xml 中
- 再将</comps>添加到 comps.xml 文件中

通过新增你的包到 comps.xml 文件，你可以根据你的需要做你自己的发布盘，确信你的包在缺省情况下会被安装。需要注意的一件事是你更新的包与其它包的依赖关系，这是你需要处理的，要注意你更新的包所应该放置的位置。另外，不要在文件中随意增加或删除其余的空格。在修改 comps.xml 之前，也最好对最初的 comps.xml 做个备份，以备恢复使用。

4.3 重新编译安装程序，调整安装阶段

安装程序是不可能一次就加载进来的，必须分阶段进行，通常我们就称为"stage"。第一个阶段所用程序很小，只有这样才能从一张软盘、tftp 服务器等等上面加载。通常这个阶段程序包含的只有一个精简过的 Linux 内核和在后续步骤当中必要的一些驱动程序（比如 SCSI）。

要采用一个新的 RedHat 安装，就会需要很多的映像，最明显的就是引导安装盘本身（从软驱或者光驱安装）的 boot.img，

但是我们也需要对从硬盘、网络文件系统等安装方式提供支持。

RedHat 就此提供了很好的脚本命令，只需一个简单的操作就可以完成所有的操作。这些脚本的工作就是把某些 RPM 包的内容提取出来，然后用来生成各安装步骤所用程序的映像。

所再强调的是，我们必须保证安装了 anaconda-runtime：

```
#rpm -i anaconda-runtime-xxxxx-i386.rpm
```

接着进入目录/usr/lib/anaconda-runtime，这里我们会看到一些非常有用的脚本，比如：

mk-images.i386：包涵有创建启动磁盘时 i386的专门设置（通常情况下，网络和 pcmcia）以及辅助磁盘驱动程序。在此您可以改变启动映像中所包含的模块，比如说在网络启动磁盘有：

```
……
NETWORKMODULES="$COMMONMODULES nfs 3c59x eepro100 tulip pcnet32
ne2k-pci 8139too"
……..
```

buildinstall 这是主要的。

```
#cd   /usr/lib/anaconda-runtime
#./buildinstall ~/disc1/
```

这个脚本命令会在~/disc1/images 目录下更新一些的文件。

4.4 生成新的 hdlist 文件

当安装时，安装程序需要依赖光盘上的 Fedora/base/hdlist 文件，它包含的是所有可用的 RPM 包的必要信息，这些信息在安装过程当中是用来显示每一个包的用途以及解决用户选择软件包后的依赖性问题。

用以生成 hdlist 文件的程序叫做 genhdlist，它是由 anaconda-runtime 这个包产生的。现在的 genhdlist 多了一个新的参数：--withnumbers，是用来记录 hdlist 文件中每个 RPM 包的媒介代号。

分步处理的过程如下：

```
#rpm -i anaconda-runtime-xxxxx-i386.rpm
#cd /usr/lib/anaconda-runtime
#./genhdlist -- withnumbers ~/disc1   ~/disc2   ~/disc3   ~/disc4
```

整个过程只需要执行一个脚本，见附录一：kernel-update.sh。

如果你在系统中添加了 RPM 包，那么在生成安装盘之前，最好将这四张盘上的内容复制到一个目录下，然后修改附录一的脚本文件，运行脚本，先网络安装一次，看是否存在包的依赖关系问题。如果没有，则可以生成安装盘。

回页首

5 生成 iso 映象

当前面系统进行网络安装成功后，则可以生成 iso 映象，然后进行刻盘，执行的操作如下：

```
# build disk 1
cd   ~/disc1        /*假设我们将第一张盘的内容放置在此外*/
mkisofs -R -J -T -no-emul-boot -boot-load-size 4 -boot-info-table -V fedora
    -b isolinux/isolinux.bin    -c isolinux/boot.cat -o /iso/exm-disc1.iso    .
/*使用 mkisofs 工具生成 iso 映象，将生成的 iso 映象放在/iso 目录中*/
# build disk 2
cd   ~/disc2        /*采用同样的方法，生成第二个 iso，依次。*/
mkisofs -R -J -T -V fedora -o /iso/exm-disc2.iso .
```

在生成 iso 映象之后，需要对它进行测试，你可以将它挂接到某个地方，比如：

```
mount -o loop /iso/exm1-disc1.iso   /mnt
```

在生成安装 iso（exm-disc1.iso）之后，我们可以将它复制到 windows 系统中，采用刻录程序进行刻录，然后可以从光盘安装，进行安装测试。

回页首

附录一：kernel-update.sh

```
#!/bin/sh
# current working directory
BASE=`pwd`
# generate hdlists
mkdir -p $BASE/SOURCES
echo
echo Copying disc1 to SOURCES directory, please wait...
cp -Rf $BASE/disc1/* $BASE/SOURCES
echo Copying disc2 to SOURCES directory, please wait...
cp -Rf $BASE/disc2/* $BASE/SOURCES
echo Copying disc3 to SOURCES directory, please wait...
cp -Rf $BASE/disc3/* $BASE/SOURCES
echo Copying disc4 to SOURCES directory, please wait...
cp -Rf $BASE/disc4/* $BASE/SOURCES
echo
echo Make sure anaconda, anaconda-runtime is installed...
rpm -U $BASE/SOURCES/Fedora/RPMS/anaconda-*.rpm
# generate hdlists*
cd /usr/lib/anaconda-runtime
```

```
./genhdlist --withnumbers $BASE/disc1 $BASE/disc2 $BASE/disc3    $BASE/disc4
# generate the package ordering
./pkgorder $BASE/SOURCES i386 |tee /root/pkgorder.txt
./buildinstall    --pkgorder /root/pkgorder.txt --version 1 --product "Fedora"
     --release "Fedora" $BASE/SOURCES
$BASE/SOURCES
cp -apRf $BASE/SOURCES/images/* $BASE/disc1/images
cp -apRf $BASE/SOURCES/isolinux/* $BASE/disc1/isolinux
cp -apRf $BASE/SOURCES/RedHat/base/* $BASE/disc1/RedHat/base
echo Cleaning up...
rm -rf $BASE/SOURCES
```