Руководство по сборке RPM-пакетов для дистрибутивов Альт

Валентин Соколов и другие.

Оглавление

Введение в пакетные менеджеры

RPM — это семейство пакетных менеджеров, применяемых в различных дистрибутивах GNU/Linux, в том числе и в проекте Сизиф и в дистрибутивах Альт. Практически каждый крупный проект, использующий RPM, имеет свою версию пакетного менеджера, отличающуюся от остальных.

Различия между представителями семейства RPM выражаются в:

- наборе макросов, используемых в .spec-файлах,
- различном поведении RPM при сборке «по умолчанию» — при отсутствии каких-либо указаний в .spec-файлах,
- формате строк зависимостей,
- мелких отличиях в семантике операций (например, в операциях сравнения версий пакетов),
- мелких отличиях в формате файлов.

Для пользователя различия чаще всего заключаются в невозможности поставить «неродной» пакет из-за проблем с зависимостями или из-за формата пакета.

RPM в проекте Сизиф также не является исключением. Основные особенности RPM в Альт и Сизиф от RPM других крупных проектов заключаются в следующем:

- обширный набор макросов для упаковки различных типов пакетов,
- отличающееся поведение «по умолчанию» для уменьшения количества шаблонного кода в .spec-файлах,
- наличие механизмов для автоматического поиска межпакетных зависимостей,
- наличие так называемых set-version зависимостей (начиная с `4.0.4-alt98.46`), обеспечивающих дополнительный контроль за изменением ABI библиотек,
- до `p8` и выпусков `8.x` включительно — очень древняя версия «базового» RPM (4.0.4), от которого началось развитие ветки RPM в Sisyphus (в Sisyphus и `p9` осуществлён частичный переход на `rpm 4.13`).

Вступление

Руководство по сборке RPM пакетов:

Как подготовить исходный код для сборкив RPM.

Ссылка на раздел для тех, кто не имеет опыта разработки ПО. [preparing-software-for-packaging].

Как собрать исходный код в RPM.

Ссылка на раздел для разработчиков ПО, которым необходимо собрать программное обеспечение в RPM пакеты. Программное обеспечение для упаковки.

Расширенные сценарии сборки.

Эта ссылка на справочный материал для сборщиков RPM, работающих с расширенными сценариями сборки RPM. [advanced-topics].

PDF Версия

Вы также можете скачать PDF версию данного документа.

Файловая структура

Перед тем, как приступить к сборке, нужно создать структуру каталогов, необходимую RPM, находящуюся в Вашем «домашнем» каталоге:

- Вывод команд и содержимое текстовых файлов, включая исходный код, размещаются в каталогах:

```
$ tree ~/RPM/
/home/user/RPM/
|-- BUILD
|-- BUILDROOT
|-- RPMS
|   |-- i586
|   |-- x86_64
|   `-- noarch
|-- SOURCES
|-- SPECS
`-- SRPMS
```

```
Name:           bello
Version:
Release:        alt1
Summary:
```

- Темы, представляющие интерес, или словарные термины упоминаются либо как ссылки

на соответствующую документацию или веб-сайт выделены **жирным** шрифтом, либо *курсивом*. Первые упоминания некоторых терминов ссылаются на соответствующую документацию.

- Названия утилит, команд и других элементов, обычно встречающихся в коде, написаны `моноширинным` шрифтом.

Вклад в руководство

Вы можете внести свой вклад в это руководство, отправив сообщение о проблеме или pull request на GitHub репозиторий.

Система управления пакетами. Знакомство с APT

Для установки, удаления и обновления программ и поддержания целостности системы в Linux в первую очередь стали использоваться *менеджеры пакетов* (такие, как RPM в дистрибутивах `RedHat` или dpkg в `Debian GNU/Linux`). С точки зрения менеджера пакетов программное обеспечение представляет собой набор компонентов — программных *пакетов*. Такие компоненты содержат в себе набор исполняемых программ и вспомогательных файлов, необходимых для корректной работы ПО. Менеджеры пакетов дают возможность унифицировать и автоматизировать сборку бинарных пакетов и облегчают установку программ, позволяя проверять наличие необходимых для работы устанавливаемой программы компонент подходящей версии непосредственно в момент установки, а также производя все необходимые процедуры для регистрации программы во всех операционных средах пользователя. Сразу после установки программа оказывается доступна пользователю из командной строки и появляется в меню всех графических оболочек.

Полное описание APT можно узнать, перейдя по ссылке: Установка и удаление программ (пакетов)

ПРИМЕЧАНИЕ	Установка пакетов в АЛЬТ Линукс осуществляется с помощью утилиты APT
ПРИМЕЧАНИЕ	Для сокращения команд, встречающихся в тексте, будет использоваться нотация:

- - команды без административных привелегий будут начинаться с символа “$”
- - команды без административных привелегий будут начинаться с символа “$”

ПРИМЕЧАНИЕ	По умолчанию `sudo` может быть отключено. Для получения административных привелегий используется команда `su`. Для включения `sudo` в стандартном режиме можно использовать команду:

```
# control sudowheel enabled
```

Установка необходимых пакетов для процесса сборки

Чтобы следовать данному руководству, Вам потребуется установить следующие пакеты:

ПРИМЕЧАНИЕ	Некоторые из этих пакетов устанавливаются по умолчанию в Альт. Установка проводится с правами суперпользователя.

```
# apt-get update

# apt-get install gcc rpm-build rpmlint make python gear hasher patch
```

Основные команды RPM

Для ознакомления с данным разделом потребуется скачать пакет Yodl-docs.

Как узнать информацию об RPM-пакете без установки?

После скачивания пакета можно посмотреть данные о нём перед установкой. Для этого используется **-qip**, чтобы вывести информацию о пакете.

```
$ rpm -qip yodl-docs-4.03.00-alt2.noarch.rpm
```

Вывод:

```
Name        : yodl-docs
Epoch       : 1
Version     : 4.03.00
Release     : alt2
DistTag     : sisyphus+271589.100.1.2
Architecture: noarch
Install Date: (not installed)
Group       : Documentation
Size        : 3701571
License     : GPL
Signature   : DSA/SHA1, Чт 13 мая 2021 05:44:49, Key ID 95c584d5ae4ae412
Source RPM  : yodl-4.03.00-alt2.src.rpm
Build Date  : Чт 13 мая 2021 05:44:44
Build Host  : darktemplar-sisyphus.hasher.altlinux.org
Relocations : (not relocatable)
Packager    : Aleksei Nikiforov <darktemplar@altlinux.org>
Vendor      : ALT Linux Team
URL         : https://gitlab.com/fbb-git/yodl
Summary     : Documentation for Yodl
Description :
Yodl is a package that implements a pre-document language and tools to
process it.  The idea of Yodl is that you write up a document in a
pre-language, then use the tools (eg. yodl2html) to convert it to some
final document language.  Current converters are for HTML, ms, man, LaTeX
SGML and texinfo, plus a poor-man's text converter.  Main document types
are "article", "report", "book" and "manpage".  The Yodl document
language is designed to be easy to use and extensible.

This package contais documentation for Yodl.
```

Как установить RPM-пакет?

Для установки используется параметр **-ivh**.

```
$ rpm -ivh yodl-docs-4.03.00-alt2.noarch.rpm
```

Вывод:

```
Подготовка...
################################################################# [100%]
Обновление / установка...
1: yodl-docs-1:4.03.00-alt2
################################################################# [100%]
Running /usr/lib/rpm/posttrans-filetriggers
```

Проверка установки пакета в системе.

```
$ rpm -q yodl-docs
```

Вывод:

```
yodl-docs-4.03.00-alt2.noarch
```

Просмотр файлов пакета, установленного в системе.

```
$ rpm -ql yodl-docs
```

Вывод:

```
/usr/share/doc/yodl
/usr/share/doc/yodl-doc
/usr/share/doc/yodl-doc/AUTHORS.txt
/usr/share/doc/yodl-doc/CHANGES
/usr/share/doc/yodl-doc/changelog
/usr/share/doc/yodl-doc/yodl.dvi
/usr/share/doc/yodl-doc/yodl.html
/usr/share/doc/yodl-doc/yodl.latex
/usr/share/doc/yodl-doc/yodl.pdf
/usr/share/doc/yodl-doc/yodl.ps
/usr/share/doc/yodl-doc/yodl.txt
/usr/share/doc/yodl-doc/yodl01.html
/usr/share/doc/yodl-doc/yodl02.html
/usr/share/doc/yodl-doc/yodl03.html
/usr/share/doc/yodl-doc/yodl04.html
/usr/share/doc/yodl-doc/yodl05.html
/usr/share/doc/yodl-doc/yodl06.html
/usr/share/doc/yodl/AUTHORS.txt
/usr/share/doc/yodl/CHANGES
/usr/share/doc/yodl/changelog
```

Просмотр недавно установленных пакетов.

```
rpm -qa --last|head
```

Вывод:

```
yodl-docs-4.03.00-alt2.noarch                 Чт 22 дек 2022 18:09:10
source-highlight-3.1.9-alt1.git.904949c.x86_64 Вт 20 дек 2022 18:38:29
libsource-highlight-3.1.9-alt1.git.904949c.x86_64 Вт 20 дек 2022 18:38:29
gem-asciidoctor-doc-2.0.10-alt1.noarch        Вт 20 дек 2022 18:34:04
w3m-0.5.3-alt4.git20200502.x86_64             Вт 20 дек 2022 18:23:05
sgml-common-0.6.3-alt15.noarch                Вт 20 дек 2022 18:23:05
libmaa-1.4.7-alt4.x86_64                      Вт 20 дек 2022 18:23:05
docbook-style-xsl-1.79.1-alt4.noarch          Вт 20 дек 2022 18:23:05
docbook-dtds-4.5-alt1.noarch                  Вт 20 дек 2022 18:23:05
dict-1.12.1-alt4.1.x86_64                     Вт 20 дек 2022 18:23:05
```

Поиск пакета в системе.

Команда **grep** поможет определить, установлен пакет в системе или нет:

```
$ rpm -qa | grep yodl-docs
```

Вывод:

```
yodl-docs-4.03.00-alt2.noarch
```

Проверка файла, относящегося к пакету.

Предположим, что нужно узнать, к какому конкретному пакету относится файл. Для этого используют команду:

```
$ rpm -qf /usr/share/doc/yodl-doc
```

Вывод:

```
yodl-docs-4.03.00-alt2.noarch
```

Вывод информации о пакете.

Чтобы получить информацию о пакете, установленном в систему, используем команду:

```
$ rpm -qi yodl-docs
```

Вывод:

```
Name        : yodl-docs
Epoch       : 1
Version     : 4.03.00
Release     : alt2
DistTag     : sisyphus+271589.100.1.2
Architecture: noarch
Install Date: Чт 22 дек 2022 18:09:10
Group       : Documentation
Size        : 3701571
License     : GPL
Signature   : DSA/SHA1, Чт 13 мая 2021 05:44:49, Key ID 95c584d5ae4ae412
Source RPM  : yodl-4.03.00-alt2.src.rpm
Build Date  : Чт 13 мая 2021 05:44:44
Build Host  : darktemplar-sisyphus.hasher.altlinux.org
Relocations : (not relocatable)
Packager    : Aleksei Nikiforov <darktemplar@altlinux.org>
Vendor      : ALT Linux Team
URL         : https://gitlab.com/fbb-git/yodl
Summary     : Documentation for Yodl
Description :
Yodl is a package that implements a pre-document language and tools to
process it.  The idea of Yodl is that you write up a document in a
pre-language, then use the tools (eg. yodl2html) to convert it to some
final document language.  Current converters are for HTML, ms, man, LaTeX
SGML and texinfo, plus a poor-man's text converter.  Main document types
are "article", "report", "book" and "manpage".  The Yodl document
language is designed to be easy to use and extensible.
```

Обновление пакета.

Для обновления пакета используется параметр **-Uvh**.

```
$ rpm -Uvh yodl-docs-4.03.00-alt2.noarch.rpm
```

Вывод:

```
 Подготовка...
########################################################### [100%]
    пакет yodl-docs-1:4.03.00-alt2.noarch уже установлен
```

Программное обеспечение для упаковки

RPM Пакеты

В этом разделе рассматриваются основы формата упаковки RPM. Дополнительные сведения смотри в разделе Дополнительные материалы.

Что такое RPM?

Пакет RPM - это просто файл, содержащий другие файлы и информацию о них, необходимую системе. В частности, пакет RPM состоит из архива cpio , который содержит файлы, и заголовка RPM, который содержит метаданные о пакете. Диспетчер пакетов `rpm` использует эти метаданные для определения зависимостей, места установки файлов и другой информации.

Существует два типа пакетов RPM:

- исходник RPM (SRPM)
- бинарный RPM

SRPMs и бинарные RPMs имеют общий формат файла и инструментарий, но имеют разное содержимое и служат разным целям. SRPM содержит исходный код, при необходимости исправления к нему и файл спецификации, в котором описывается, как встроить исходный код в бинарный RPM. Бинарный RPM содержит двоичные файлы, созданные из исходных текстов и патчей.

Инструменты для упаковки RPM

Рабочее пространство для упаковки RPM

Чтобы настроить макет каталога, который является рабочей областью упаковки RPM, используйте утилиту `rpmdev-setuptree`:

```
$ rpmdev-setuptree

$ tree ~/rpmbuild/
/home/user/rpmbuild/
|-- BUILD
|-- RPMS
|-- SOURCES
|-- SPECS
`-- SRPMS`

5 directories, 0 files
```

Созданные каталоги служат следующим целям:

Каталог	Назначение
BUILD	Содержит все файлы, которые появляются при создании пакета.
RPMS	Бинарные RPM создаются здесь, в подкаталогах для разных архитектур, например, в подкаталогах`x86_64` и `noarch`.
SOURCES	Здесь упаковщик помещает сжатые архивы исходного кода и патчи. Команда `rpmbuild` ищет их здесь.
SPECS	Упаковщик помещает сюда файлы спецификаций.
SRPMS	Когда rpmbuild используется для сборки SRPM вместо бинарного RPM, результирующий SRPM создается здесь.

Что такое SPEC файл?

Файл спецификации можно рассматривать как "рецепт", который утилита `rpmbuild` использует для фактической сборки RPM. Он сообщает системе сборки, что делать, определяя инструкции в серии разделов. Разделы определены в *Преамбуле* и в *Основной части*. *Преамбула* содержит ряд элементов метаданных, которые используются в *Основной части*. Тело содержит основную часть инструкций.

Пункты преамбулы

В этой таблице перечислены элементы, используемые в разделе преамбулы файла спецификации RPM:

SPEC Директива	Определение
`Name`	Базовое имя пакета, которое должно совпадать с именем файла спецификации.
`Version`	Версия upstream-кода.
`Release`	Релиз пакета используется для указания номера сборки пакета при данной версии upstream-кода. Как правило, установите начальное значение равным 1%{?dist} и увеличивайте его с каждым новым выпуском пакета. Сбросьте значение до 1 при создании новой версии программного обеспечения.
`Summary`	Краткое, в одну строку, описание пакета.
`License`	Лицензия на упаковываемое программное обеспечение. Для пакетов, распространяемых в дистрибутивах сообщества, таких как Fedora, это должна быть лицензия с открытым исходным кодом, соответствующая рекомендациям по лицензированию конкретного дистрибутива.
`URL`	Полный URL-адрес для получения дополнительной информации о программе. Чаще всего это веб-сайт upstream-проекта для упаковываемого программного обеспечения.

`Source0`	Путь или URL-адрес к сжатому архиву исходного кода (не исправленный, исправления обрабатываются в другом месте). Этот раздел должен указывать на доступное и надежное хранилище архива, например, на upstream-страницу, а не на локальное хранилище упаковщика. При необходимости можно добавить дополнительные исходные директивы, каждый раз увеличивая их количество, например: Source1, Source2, Source3 и так далее.
`Patch0`	Название первого исправления, которое при необходимости будет применено к исходному коду. При необходимости можно добавить дополнительные директивы PatchX, увеличивая их количество каждый раз, например: Patch1, Patch2, Patch3 и так далее.
`BuildArch`	Если пакет не зависит от архитектуры, например, если он полностью написан на интерпретируемом языке программирования, установите для этого значение `BuildArch: noarch`. Если этот параметр не задан, пакет автоматически наследует архитектуру компьютера, на котором он построен, например `x86_64`.
`BuildRequires`	Разделённый запятыми или пробелами список пакетов, необходимых для сборки программы, написанной на скомпилированном языке. Может быть несколько записей `BuildRequires`, каждая в отдельной строке в SPEC файле.
`Requires`	Разделённый запятыми или пробелами список пакетов, необходимых программному обеспечению для запуска после установки. Может быть несколько записей `Requires`, каждая в отдельной строке в SPEC файле.
`ExcludeArch`	Если часть программного обеспечения не может работать на определенной архитектуре процессора, Вы можете исключить эту архитектуру здесь.

Директивы `Name`, `Version` и `Release` содержат имя файла пакета RPM. Разработчики пакетов RPM и системные администраторы часто называют эти три директивы **N-V-R** или **NVR**, поскольку имена файлов пакетов RPM имеют формат `NAME-VERSION-RELEASE`.

Вы можете получить пример `NAME-VERSION-RELEASE`, выполнив запрос с использованием `rpm` для конкретного пакета:

```
$ rpm -q python
python-2.7.5-34.el7.x86_64
```

Здесь `python` - это имя пакета, `2.7.5` - версия, а `34.el7` - релиз. Последний маркер `x86_64` - сведения об архитектуре. В отличие от NVR, маркер архитектуры не находится под прямым управлением RPM упаковщика, а определяется средой сборки `rpmbuild`. Исключением из этого правила является архитектурно-независимый пакет `noarch`.

Составляющие основной части

В этой таблице перечислены элементы, используемые в разделе Body (Тело, основная часть) файла спецификации RPM:

SPEC Директива	Определение
`%description`	Полное описание программного обеспечения, входящего в комплект поставки RPM. Это описание может занимать несколько строк и может быть разбито на абзацы.
`%prep`	Команда или серия команд для подготовки программного обеспечения к сборке, например, распаковка архива в Source0. Эта директива может содержать сценарий оболочки.
`%build`	Команда или серия команд для фактической сборки программного обеспечения в машинный код (для скомпилированных языков) или байт-код (для некоторых интерпретируемых языков).
`%install`	Команда или серия команд для копирования требуемых артефактов сборки из `%builddir` (где происходит сборка) в`%buildroot` каталог (который содержит структуру каталогов с файлами, подлежащими упаковке). Обычно это означает копирование файлов из `~/rpmbuild/BUILD` в `~/rpmbuild/BUILDROOT` и создание необходимых каталогов `~/rpmbuild/BUILDROOT`. Это выполняется только при создании пакета, а не при установке пакета конечным пользователем. Подробности см. в разделе Работа со SPEC файлом.
`%check`	Команда или серия команд для тестирования программного обеспечения. Обычно включает в себя такие вещи, как модульные тесты.
`%files`	Список файлов, которые будут установлены в системе конечного пользователя.
`%changelog`	Запись изменений, произошедших с пакетом между различными `Version` или `Release` сборками.

Дополнительные элементы

Файл спецификации также может содержать дополнительные элементы. Например, файл спецификации может содержать *скриптлеты_ и* триггеры__. Они вступают в силу в разные моменты процесса установки в системе конечного пользователя (не в процессе сборки).

Дополнительную информацию см. Триггеры и скриптлеты.

BuildRoots

В контексте упаковки RPM "buildroot" - это среда chroot Это означает, что артефакты сборки размещаются здесь с использованием той же иерархии файловой системы, что и в системе конечного пользователя, при этом "buildroot" выступает в качестве корневого каталога. Размещение артефактов сборки должно соответствовать стандарту иерархии файловой системы конечного пользователя.

Файлы в "buildroot" позже помещаются в архив cpio , который становится основной частью RPM. Когда RPM устанавливается в системе конечного пользователя, эти файлы извлекаются в корневой каталог, сохраняя правильную иерархию.

ПРИМЕЧАНИЕ	Начиная с выпуска Red Hat Enterprise Linux 6, программа `rpmbuild` имеет свои собственные значения макросов по умолчанию. Поскольку переопределение этих значений по умолчанию приводит к ряду проблем, Red Hat не рекомендует определять собственное значение этого макроса. Вы можете использовать макрос `%{buildroot}` с параметрами по умолчанию из каталога `rpmbuild`.

Работа со SPEC файлами

Большая часть упаковки программного обеспечения в RPMs - это редактирование файла спецификации. В этом разделе мы обсудим, как создать и изменить SPEC файл.

Чтобы упаковать новое программное обеспечение, Вам необходимо создать новый файл спецификации. Вместо того, чтобы писать его вручную с нуля, используйте утилиту `rpmdev-newspec`. Она создаёт незаполненный файл спецификации, и Вы заполняете необходимые директивы и поля.

В этом руководстве мы используем три примера реализации программы 'Hello World!', созданной при подготовке программного обеспечения для упаковки:

- bello-0.1.tar.gz
- pello-0.1.1.tar.gz
- cello-1.0.tar.gz
    - cello-output-first-patch.patch

Переместите их в `~/rpmbuild/SOURCES`.

Создайте SPEC файл для каждой из трёх программ:

ПРИМЕЧАНИЕ	Некоторые текстовые редакторы, ориентированные на программистов, предварительно заполняют новый `.spec` файл с их собственным шаблоном спецификации. `rpmdev-newspec` предоставляет независимый от редактора метод, именно поэтому он используется в этом руководстве.

```
$ cd ~/rpmbuild/SPECS

$ rpmdev-newspec bello
bello.spec created; type minimal, rpm version >= 4.11.

$ rpmdev-newspec cello
cello.spec created; type minimal, rpm version >= 4.11.
```

```
$ rpmdev-newspec pello
pello.spec created; type minimal, rpm version >= 4.11.
```

`~/rpmbuild/SPECS/` каталог теперь имеет три SPEC файла с именами `bello.spec`, `cello.spec`, и `pello.spec`.

Изучите файлы. Директивы в них представляют собой директивы, описанные в разделе Что такое SPEC файл. В следующих разделах Вы заполните эти файлы спецификаций.

ПРИМЕЧАНИЕ

Утилита `rpmdev-newspec` не использует рекомендации или соглашения, характерные для какого-либо конкретного дистрибутива Linux. Однако этот документ предназначен для Fedora, CentOS и RHEL, поэтому В ы заметите, что:

- Используйте `rm $RPM_BUILD_ROOT` при сборке на *CentOS* (версии, предшествующие версии 7.0) или на Fedora (версии, предшествующие версии 18).
- Мы предпочитаем использовать обозначение `%{buildroot}` вместо `$RPM_BUILD_ROOT` при обращении к Buildroot RPM для обеспечения согласованности со всеми другими определенными или предоставленными макросами во всем файле спецификации..

Ниже приведены три примера. Каждый из них полностью описан, так что вы можете перейти к конкретному, если он соответствует вашим потребностям в упаковке. Или прочтите их все, чтобы полностью изучить упаковку различных видов программного обеспечения.

Имя программы	Объяснение примера
bello	Программа, написанная на необработанном интерпретируемом языке программирования. Пример демонстрирует, когда исходный код не нужно собирать, а нужно только установить. Если необходимо упаковать предварительно скомпилированный бинарный файл, Вы также можете использовать этот метод.
pello	Программа, написанная на интерпретируемом языке программирования с последующей байт-компиляцией. Пример демонстрирует байт-компиляцию исходного кода и установку байт-кода - результирующих, предварительно оптимизированных файлов.
cello	Программа, написанная на изначально скомпилированном языке программирования. Пример демонстрирует общий процесс компиляции исходного кода в машинный код и установки результирующих исполняемых файлов.

Сборка RPMS

RPMs собираются с помощью команды `rpmbuild`. Различные сценарии и желаемые результаты требуют различных комбинаций аргументов для `rpmbuild`. В этом разделе описываются два основных сценария:

1. сборка исходного RPM
2. сборка бинарного RPM

Команда `rpmbuild` ожидает определенную структуру каталогов и файлов. Это та же структура, что и в утилите `rpmdev-setuptree`. Предыдущие инструкции также подтвердили требуемую структуру.

Исходный RPMs

Зачем создавать исходный RPM (SRPM)?

1. Чтобы сохранить точный источник определенного Name-Version-Release RPM, который был развернут в среде. Это включает в себя точный SPEC файл, исходный код и все соответствующие исправления. Это полезно для просмотра истории и для отладки.
2. Чтобы иметь возможность создавать бинарный RPM на другой аппаратной платформе или архитектуре.

Для сборки SRPM:

```
$ rpmbuild -bs _SPECFILE_
```

Замените *SPECFILE* именем SPEC файла. Параметр `-bs` "исходный код сборки".

Здесь мы собираем SRPMs для `bello`, `pello` и `cello`:

```
$ cd ~/rpmbuild/SPECS/

$ rpmbuild -bs bello.spec
Wrote: /home/admiller/rpmbuild/SRPMS/bello-0.1-1.el7.src.rpm

$ rpmbuild -bs pello.spec
Wrote: /home/admiller/rpmbuild/SRPMS/pello-0.1.1-1.el7.src.rpm

$ rpmbuild -bs cello.spec
Wrote: /home/admiller/rpmbuild/SRPMS/cello-1.0-1.el7.src.rpm
```

Обратите внимание, что SRPMS были помещены в каталог `rpmbuild/SRPMS`, который является частью структуры, ожидаемой `rpmbuild`.

Это все, что нужно для сборки SRPM.

Бинарный RPMS

Существует два метода сборки бинарных RPMs:

1. Восстановление его из SRPM с использованием комманды`rpmbuild --rebuild`.
2. Собираем его из файла спецификации с помощью команды `rpmbuild -bb`. Опция `-bb` означает "собрать бинарный файл" (`build binary`).

Восстановление из исходного RPM

Чтобы перестроить `bello`, `pello` и `cello` из исходных RPM (SRPMs), запустите:

```
$ rpmbuild --rebuild ~/rpmbuild/SRPMS/bello-0.1-1.el7.src.rpm
[output truncated]

$ rpmbuild --rebuild ~/rpmbuild/SRPMS/pello-0.1.1-1.el7.src.rpm
[output truncated]

$ rpmbuild --rebuild ~/rpmbuild/SRPMS/cello-1.0-1.el7.src.rpm
[output truncated]
```

Теперь Вы собрали RPM. Несколько заметок:

- Выходные данные, генерируемые при сборке бинарного RPM, являются подробными, что полезно для отладки. Выходные данные различаются для разных примеров и соответствуют их SPEC файлам.
- Конечные бинарные RPM находятся в `~/rpmbuild/RPMS/YOURARCH`, где `YOURARCH` - это Ваша архитектура , или в `~/rpmbuild/RPMS/noarch/`, если пакет не зависит от архитектуры.
- Вызов `rpmbuild --rebuild` включает в себя:

  1. Установку содержимого RPM - файла спецификации и исходного кода - в каталог `~/rpmbuild/`.
  2. Сборка с использованием установленного содержимого.
  3. Удаление файла спецификации и исходного кода.

  Вы можете сохранить файл спецификации и исходный код после сборки. Для этого у Вас есть два варианта:

  - При сборке используйте опцию `--recompile` вместо `--rebuild`.
  - Установите SRPMS с помощью следующих команд:

  ```
  $ rpm -Uvh ~/rpmbuild/SRPMS/bello-0.1-1.el7.src.rpm
  Updating / installing...
     1:bello-0.1-1.el7                  ################################# [100%]

  $ rpm -Uvh ~/rpmbuild/SRPMS/pello-0.1.1-1.el7.src.rpm
  Updating / installing...
  ```

```
   1:pello-0.1.1-1.el7              ################################# [100%]

$ rpm -Uvh ~/rpmbuild/SRPMS/cello-1.0-1.el7.src.rpm
Updating / installing...
   1:cello-1.0-1.el7                ################################# [100%]
```

В этом руководстве выполните приведенные выше команды `rpm -Uvh` чтобы продолжить взаимодействие с файлами спецификаций и исходными кодами.

Создание бинарного файла из SPEC файла

Чтобы собрать `bello`, `pello`, и `cello` из их SPEC файлов, запустите:

```
$ rpmbuild -bb ~/rpmbuild/SPECS/bello.spec

$ rpmbuild -bb ~/rpmbuild/SPECS/pello.spec

$ rpmbuild -bb ~/rpmbuild/SPECS/cello.spec
```

Теперь Вы собрали RPM из SPEC файлов.

Большая часть информации, содержащейся в разделе Восстановление из исходного RPM применима здесь.