

Руководство по сборке RPM-пакетов для дистрибутивов АЛЬТ

Валентин Соколов и другие.

Оглавление

Введение в пакетные менеджеры	1
Вступление	2
PDF Версия	2
Структура документации	2
Система управления пакетами. Знакомство с APT	4
Установка необходимых пакетов для процесса сборки	5
Основные команды RPM	6
Программное обеспечение для упаковки	11
RPM-пакеты	11
Что такое RPM-пакет?	11
Инструменты для сборки RPM-пакетов	11
Рабочее пространство для сборки RPM-пакетов	11
Что такое СПЕС-файл?	12
Пример .спес-файла	12
Инструмент Gear	15
Вступление	15
Структура репозитория	16
Правила экспорта	17
Основные типы устройства gear-репозитория	17

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 .спес-файлах,
- различном поведении RPM при сборке «по умолчанию» — при отсутствии каких-либо указаний в .спес-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 отличия RPM в Альт и Сизиф от RPM других крупных проектов заключаются в следующем:

- обширный набор макросов для сборки различных типов пакетов,
- отличающееся поведение «по умолчанию» для уменьшения количества шаблонного кода в .спес-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 сборки 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.](#)

Структура документации

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

- Отображение фаловой структуры будет представлено следующим образом:

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

- В дальнейшем вывод команд будет продемонстрирован следующим образом:

```
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.

Система управления пакетами.

Знакомство с АРТ

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 АРТ можно узнать, перейдя по ссылке: [Установка и удаление программ \(пакетов\)](#)

ПРИМЕЧАНИЕ

Установка пакетов в АЛЪТ Линукс осуществляется с помощью утилиты АРТ

ПРИМЕЧАНИЕ

Для сокращения команд, встречающихся в тексте, будет использоваться нотация:

- - команды без административных привелегий будут начинаться с символа “\$”
- - команды с административными привелегиями 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 пакет. В качестве примера мы будем использовать пакет [Yodl-docs](#).

Как узнать информацию о RPM-пакете без установки?

После скачивания пакета можно посмотреть данные о нём перед установкой. Для этого используется **-qip**, (Query|Install|Package) чтобы вывести информацию о пакете.

ПРИМЕЧАНИЕ

ключ **-p** (-package) работает не с базой RPM-пакетов, а с конкретным пакетом. Например: чтобы получить информацию о файлах, находящихся в пакете, который не установлен в систему, используют ключи **-qpl**(Query|Package|List).

```
$ 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 contains documentation for Yodl.

Как установить RPM-пакет?

Для установки используется параметр **-ivh** (Install | Verbose | Hash).

ПРИМЕЧАНИЕ

Ключи **-v** и **-h** не влияют на установку, а служат для вывода наглядного процесса сборки в консоль. Ключ **-v** (verbose) выводит детальные значения. Ключ **-h** (hash) выводит "#" по мере установки пакета.

```
$ 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

ПРИМЕЧАНИЕ

Справку по ключам можно получить, набрав в консоли команду `rpm --help`

Программное обеспечение для упаковки

RPM-пакеты

В этом разделе рассматриваются основы сборки RPM-пакетов. Дополнительные сведения смотри в разделе [Дополнительные материалы](#).

Что такое RPM-пакет?

RPM-пакет - это архив, содержащий в себе архив [.cpio](#) с файлами (скомпилированные исполняемые файлы, библиотеки и данные), а также метаданные - имя пакета, его описание, зависимости и т.д. Менеджер пакетов RPM использует эти метаданные для проверки наличия необходимых пакетов из списка зависимостей, исполнения инструкций по установке файлов и сохранения общей информации о пакете у себя в базе.

Существует два типа RPM-пакетов:

- SRPM-пакеты (исходники) - архив с расширением [.src.rpm](#)
- RPM-пакеты (бинарники) - архив с расширением [.rpm](#)

SRPM и RPM-пакеты имеют общий формат и инструментарий, но имеют разное содержимое и служат разным целям. SRPM содержит исходный код, при необходимости патчи к нему и спес-файл, в котором описывается, как собрать исходный код в бинарный RPM-пакет. Бинарный RPM-пакет содержит бинарные файлы, созданные из исходных текстов и патчей, если таковые имелись.

Инструменты для сборки RPM-пакетов

Пакет [rpmdevtools](#), установленный на этапе [Необходимые пакеты](#), предоставляет несколько утилит для сборки RPM-пакетов. Чтобы перечислить эти утилиты, выполните в консоли следующую команду:

```
$ rpm -ql rpmdevtools | grep bin
```

Для получения дополнительной информации о вышеуказанных утилитах см. их страницы руководства или диалоговые окна справки. /

Рабочее пространство для сборки RPM-пакетов

Чтобы создать дерево каталогов, которое является рабочей областью сборки RPM-пакетов, используйте утилиту [rpmdev-setuptree](#):

```
$ rpmdev-setuptree  
  
$ tree ~/rpmbuild/  
/home/user/rpmbuild/
```

```
|-- BUILD
|-- RPMS
|-- SOURCES
|-- SPECS
-- SRPMS
```

Созданные каталоги служат следующим целям:

Каталог	Назначение
BUILD	Содержит все файлы, которые появляются при сборке пакета.
RPMS	Здесь появляются готовые бинарные RPM-пакеты (<i>.rpm</i>), в подкаталогах для разных архитектур, например, в подкаталогах <i>x86_64</i> и <i>noarch</i> .
SOURCES	Здесь находятся архивы исходного кода и патчи. Утилита <i>rpmbuild</i> ищет их здесь.
SPECS	Здесь хранятся спес-файлы.
SRPMS	Здесь находятся пакеты с исходниками (<i>.src.rpm</i>).

Что такое СПЕС-файл?

Спес-файл можно рассматривать как "инструкцию", который утилита *rpmbuild* использует для фактической сборки RPM-пакет. 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.

Пример .спес-файла

Данный пример взят из [ALT Linux Wiki](#).

```
Name: sampleprog
Version: 1.0
Release: alt1

Summary: Sample program specfile
Summary(ru_RU.UTF-8): Пример спек-файла для программы

License: GPLv2+
Group: Development/Other
Url: http://www.altlinux.org/SampleSpecs/program

Packager: Sample Packager <sample@altlinux.org>

Source: %name-%version.tar
Patch0: %name-1.0-alt-makefile-fixes.patch

%description
```

This specfile is provided as sample specfile for packages with programs.
It contains most of usual tags and constructions used in such specfiles.

```
%description -l ru_RU.UTF-8
Этот спек-файл является примером спек-файла для пакета с программой. Он содержит
основные теги и конструкции, используемые в подобных спек-файлах.

%prep
%setup
%patch0 -p1

%build
%configure
%make_build

%install
%makeinstall_std
%find_lang %name

%files -f %name.lang
%doc AUTHORS ChangeLog NEWS README THANKS TODO contrib/ manual/
%_bindir/*
%_mandir/*

%changelog
* Sat Sep 33 3001 Sample Packager <sample@altlinux.org> 1.0-alt1
- initial build
```

Пункты преамбулы

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

СРЕС Директива	Определение
Name	Базовое имя пакета, которое должно совпадать с именем спес-файла.
Version	Версия upstream-кода.
Release	Релиз пакета используется для указания номера сборки пакета при данной версии upstream-кода. Как правило, установите начальное alt1 и увеличивайте его с каждым новым выпуском пакета, например: alt1, alt2, alt3 и т.д. Сбросьте значение до alt1 при создании новой версии программного обеспечения.
Summary	Краткое, в одну строку, описание пакета.
License	Лицензия на собираемое программное обеспечение.
URL	Полный URL-адрес для получения дополнительной информации о программе. Чаще всего это ссылка на GitHub upstream-проекта для собираемого 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 сборщика. При необходимости можно добавить дополнительные исходные директивы, каждый раз увеличивая их количество, например: Source1, Source2, Source3 и так далее.
Patch0	Название первого патча, который при необходимости будет применен 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 собран, 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. Это его зависимости . 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 RPM-пакета. Эти три директивы часто называют **N-V-R** или **NVR**, поскольку имена RPM-пакета имеют формат **NAME-VERSION-RELEASE**.

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

```
$ rpm -q rpmdevtools
rpmdevtools-8.10-alt2.noarch
```

Здесь **rpmdevtools** - это имя пакета, **8.10** - версия, а **alt2** - релиз. Последний маркер **noarch** - сведения об архитектуре. В отличие от NVR, маркер архитектуры не находится под прямым управлением сборщика, а определяется средой сборки **rpmbuild**. Исключением из этого правила является архитектурно-независимый пакет **noarch**.

Составляющие основной части

В этой таблице перечислены элементы, используемые в теде файла спецификации RPM-пакета:

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

Инструмент Gear

Вступление

Основной смысл хранения исходного кода пакетов в `git-репозитории` заключается в более эффективной и удобной совместной разработке, а также в минимизации используемого дискового пространства для хранения архива репозитория за длительный срок и минимизации трафика при обновлении исходного кода.

Идея gear заключается в том, чтобы с помощью одного файла с простыми правилами (для обработки которых достаточно `sed` и `git`) можно было бы собирать пакеты из произвольно устроенного `git-репозитория`, по аналогии с `hasher`, который был задуман как средство для сборки пакетов из произвольных 'srpm-пакетов'.

Структура репозитория

Хотя **gear** и не накладывает ограничений на внутреннюю организацию git-репозитория (не считая требования наличия файла с правилами), есть несколько соображений о том, как более эффективно и удобно организовывать git-репозитории, предназначенные для хранения исходного кода пакетов.

Одна сущность — один репозиторий

Не стоит помещать в один репозиторий несколько разных пакетов, за исключением случаев, когда у этих пакетов есть общий пакет-предок.

- Плюсы: Соблюдение этого правила облегчает совместную работу над пакетом, поскольку неперегруженный репозиторий легче клонировать и в целом инструментарий git больше подходит для работы с такими репозиториями.
- Минусы: Несколько сложнее выполнять операции **fetch** и **push** в случае, когда репозитория, которые надо обработать, много. Впрочем, **fetch/push** в цикле выручает.

Несжатый исходный код

Сжатый разными средствами (**gzip**, **bzip2** и т.п.) исходный код лучше хранить в git-репозитории в несжатом виде.

- Плюсы: Изменение файлов, которые помещены в репозиторий в сжатом виде, менее удобно отслеживать штатными средствами (**git diff**). Поскольку git хранит объекты в сжатом виде, двойное сжатие редко приводит к экономии дискового пространства. Наконец, алгоритм, применяемый для минимизации трафика при обновлении репозитория по протоколу git, более эффективен на несжатых данных.
- Минусы: Поскольку некоторые виды сжатия одних и тех же данных могут приводить к разным результатам, может уменьшиться степень первозданности (нативности) исходного кода.

Распакованный исходный код

Исходный код, запакованный архиваторами (**tar**, **cpio**, **zip** и т.п.), лучше хранить в git-репозитории в распакованном виде.

- Плюсы: Существенно удобнее вносить изменения в конечные файлы и отслеживать изменения в них, заметно меньше трафик при обновлении.
- Минусы: Поскольку **git** из информации о владельце, правах доступа и дате модификации файлов хранит только исполняемость файлов, любой архив, созданный из репозитория, будет по этим параметрам отличаться от первозданного. Помимо потери нативности, изменение прав доступа и даты модификации может теоретически повлиять на результат сборки пакета. Впрочем, сборку таких пакетов, если они будут обнаружены, всё равно придётся исправить.

Форматированный changelog

В changelog релизного **commit** имеет смысл включать соответствующий текст из **changelog** пакета, как это делают утилиты **gear-commit** (обёртка к **git commit**, специально предназначенная для этих целей) и **gear-srpmimport**. В результате можно будет получить представление об изменениях в очередном релизе пакета, не заглядывая в спес-файл

самого пакета.

Правила экспорта

С одной стороны, для того, чтобы `srpm`-пакет мог быть импортирован в `git`-репозиторий наиболее удобным для пользователя способом, язык правил, согласно которым производится экспорт из коммита репозитория (в форму, из которой можно однозначно изготовить `srpm`-пакет или запустить сборку), должен быть достаточно выразительным.

С другой стороны, для того, чтобы можно было относительно безбоязненно собирать пакеты из чужих `gear`-репозиториях, этот язык правил должен быть достаточно простым.

Файл правил экспорта (по умолчанию в `.gear/rules`) состоит из строк формата:

```
директива: параметры
```

Параметры разделяются пробельными символами.

Директивы позволяют экспортировать:

1. Любой файл из дерева, соответствующего коммиту;
2. Любой каталог из дерева, соответствующего коммиту в виде `tar`- или `zip`-архива;
3. `nified diff` между любыми каталогами, соответствующими коммитам.

Файлы на выходе могут быть сжаты разными средствами (`gzip`, `bzip2` и т.п.). В качестве коммита может быть указан как целевой коммит (значение параметра `-t` утилиты `gear`), так и любой из его предков при соблюдении условий, гарантирующих однозначное вычисление полного имени коммита-предка по целевому коммиту.

(Правила экспорта из `gear`-репозитория описаны детально в `gear-rules`.)[\(ссылка под редактуру\)](#)

Основные типы устройства `gear`-репозитория

Правила экспорта реализуют основные типы устройства `gear`-репозитория следующим образом:

Архив с модифицированным исходным кодом

С помощью простого правила

```
tar: .
```

Всё дерево исходного кода экспортируется в один `tar`-архив. Если у проекта есть `upstream`, публикующий `tar`-архивы, то добавление релиза в имя `tar`-архива, например, с помощью правила:

```
tar: . name=@name@-@version@-@release@
```

позволяет избежать коллизий.

Архив с немодифицированным исходным кодом и патчем, содержащем локальные изменения

Если дерево с немодифицированным исходным кодом хранится в отдельном подкаталоге, а локальные изменения хранятся в gear-репозитории в виде отдельных патч-файлов, то правила экспорта могут выглядеть следующим образом:

```
tar: package_name  
copy: *.patch
```

Такое устройство репозитория получается при использовании утилиты `gear-srpmimport`, предназначенной для быстрой миграции от `srpm`-файла к `gear`-репозиторию.

Смешанные типы

Вышеперечисленные типы устройства gear-репозитория являются основными, но не исчерпывающими. Правила экспорта достаточно выразительны для того, чтобы реализовать всевозможные сочетания основных типов и создать полнофункциональный gear-репозиторий на любой вкус.