Документация участника Fedora

Коллекции программ

Руководство по работе с коллекциями программ в Fedora

Petr Kovář

Документация участника Fedora Коллекции программ Руководство по работе с коллекциями программ в Fedora Редакция 1.0

Автор Petr Kovář *pkovar@redhat.com*

Данное руководство вводит понятие коллекций программ и дает детальное объяснение, как создавать для них пакеты. Разработчики и системные администраторы, имеющие общее представление о сборке RPM-пакетов, но не знакомые с концепцией коллекций, могут использовать данное руководство как введение в создание и использование коллекций программ.

Введение

1. Соглашения документа

В этом руководстве используются различные стили для выделения текста.

В PDF и бумажной версиях руководства используются шрифты семейства *Liberation*[1]. Эти же шрифты будут использоваться для отображения HTML-версии, если они установлены в вашей системе. В противном случае будут использоваться аналогичные шрифты. Red Hat Enterprise Linux 5 и более поздние версии включают в свой состав комплект Liberation по умолчанию.

1.1. Типографические соглашения

Для выделения текста используются четыре стиля, которые будут перечислены далее.

`Моноширинный жирный шрифт`

Используется для выделения вводимого текста, включая команды оболочки, а также имен файлов, путей и комбинаций клавиш. Пример:

> Чтобы просмотреть содержимое файла **`my_next_bestselling_novel`** в текущем каталоге, в строке приглашения оболочки введите **`cat my_next_bestselling_novel`** и нажмите **`Enter`** для выполнения этой команды.

Приведенный текст содержит имя файла, команду оболочки и имя клавиши, которые выделены моноширинным жирным шрифтом.

Для разделения клавиш в составе комбинаций используется дефис. Пример:

> Нажмите **`Enter`** для исполнения команды.
>
> Нажмите **`Ctrl`**+**`Alt`**+**`F2`** для перехода в первый виртуальный терминал. Нажмите **`Ctrl`**+**`Alt`**+**`F1`** , чтобы вернуться в сессию X-Windows.

В первом примере жирным шрифтом выделено название отдельной клавиши, во втором — комбинаций клавиш.

Этим же шрифтом выделяются имена классов, методов, функций, переменных и возвращаемые ими значения. Пример:

> Классы файлов включают **`filesystem`** для файловых систем, **`file`** для файлов, **`dir`** для каталогов. Каждому классу соответствует набор разрешений.

Пропорциональный жирный

Выделяет системные слова и фразы, что включает имена приложений, текст диалогов, названия меню, текст кнопок, флажков и других элементов графического интерфейса. Пример:

> В главном меню выберите **Система** → **Параметры** → **Мышь** для запуска утилиты **Настройки мыши**. На вкладке **Кнопки** установите флажок **Настроить мышь под левую руку** и нажмите кнопку **Закрыть**, чтобы настроить мышь для левши.

[1] https://fedorahosted.org/liberation-fonts/

> Чтобы вставить специальный символ в файл **gedit**, выберите **Приложения** → **Стандартные** → **Таблица символов**. Затем в меню выберите **Поиск** → **Поиск…**, введите имя символа и нажмите кнопку **Найти следующее**. Найденный символ будет выделен в **таблице символов**. Дважды щелкните на этом символе, чтобы вставить его в поле **Текст для копирования** и нажмите кнопку **Копировать**. Теперь вернитесь к вашему документу и в меню выберите **Правка** → **Вставить**.

Приведенный выше текст содержит имя приложения, названия меню, кнопок и текста элементов графического интерфейса.

Моноширинный жирный курсив или ***пропорциональный жирный курсив***

Оба типа выделения обозначают изменяемый или заменяемый текст. Курсив сообщает о том, что не следует вводить приведенный текст напрямую, а изменить в соответствии с вашими настройками. Пример:

> Для подключения к удаленной машине с помощью SSH в строке приглашения выполните **ssh *имя_пользователя*@*имя_домена***. Скажем, имя удаленной машины – **example.com**, а ваше имя пользователя – john, тогда команда будет выглядеть так: **ssh john@example.com**.
>
> Команда **mount -o remount *файловая_система*** повторно подключит заданную файловую систему. Например, для **/home** команда будет выглядеть так: **mount -o remount /home**.
>
> Чтобы просмотреть версию установленного пакета, выполните команду **rpm -q *пакет***. Результат команды будет представлен в формате ***пакет-версия-выпуск***.

В приведенных примерах жирным курсивом выделяются имя пользователя, имя домена, файловой системы, пакет, его версия и выпуск.

Также курсивом выделяются термины, которые встречаются в тексте документа впервые. Пример:

> Publican — система публикации *DocBook*.

1.2. Соглашения по выделению текста

Вывод экрана и листинг исходного кода будут отделены от окружающего текста.

Для отображения текста, который вы увидите на экране, используется **моноширинный шрифт**:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Для отображения содержимого исходного кода используется **моноширинный шрифт**:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
```

```
    {
        InitialContext iniCtx = new InitialContext();
        Object         ref    = iniCtx.lookup("EchoBean");
        EchoHome       home   = (EchoHome) ref;
        Echo           echo   = home.create();

        System.out.println("Created Echo");

        System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
    }
}
```

1.3. Примечания и предупреждения

Наконец, чтобы привлечь внимание читателя к важной информации, используются три стиля.

Примечание

Примечания обычно содержат дополнительную информацию. Если вы их проигнорируете, это не критично, но вы можете пропустить совет, который, возможно, поможет сэкономить время при выполнении задания.

Важно

На информацию, отмеченную как важную, следует обратить особое внимание. Она может включать изменения настроек текущего сеанса или, например, перечень служб, которые нужно запустить, прежде чем обновления вступят в силу. Ознакомление с важной информацией значительно облегчит вашу работу.

Предупреждение

Не стоит игнорировать предупреждения, так как они содержат важную информацию, которая позволит избежать потери данных.

2. Нам нужны ваши отзывы!

Если вы нашли опечатку в этом руководстве, или у вас есть идеи по его усовершенствованию, мы будем рады их выслушать. Пожалуйста, оставьте сообщение в Bugzilla по адресу *http://bugzilla.redhat.com/bugzilla/*, выбрав компонент **Fedora Documentation.**

Не забудьте указать в запросе идентификатор данного руководства: *software-collections-guide*.

Если у вас есть предложения по улучшению этого руководства, постарайтесь описать их как можно подробнее. Если же вы нашли ошибку, пожалуйста, укажите номер раздела и окружающий текст, чтобы мы могли быстро ее найти.

3. Благодарности

Авторы хотели бы выразить благодарность Jindřich Nový, Marcela Mašláňová, Bohuslav Kabrda, Florian Nadge, Stephen Wadeley, Douglas Silas и Vít Ondruch, а также многим другим людям за их ценный вклад в написание этой книги.

Введение в коллекции программ

В этой главе вводится понятие коллекций программ.

1.1. Достоинства RPM

Red Hat Package Manager (RPM) облегчает распространение, управление и обновление программного обеспечения для операционной системы Fedora. Многие производители программного обеспечения распространяют свои продукты в форме обычных архивных файлов (например, tar-файлов). Однако, их сборка в виде RPM-пакетов дает несколько преимуществ.

С помощью RPM можно:

Устанавливать, переустанавливать, удалять, обновлять и проверять пакеты.

Пользователям доступны стандартные инструменты (например, **Yum** или **PackageKit**) для установки, переустановки, удаления, обновления или проверки целостности RPM-пакетов.

Запрашивать данные о пакетах и проверять их целостность.

Поскольку RPM поддерживает базу данных инсталлированных пакетов и принадлежащих им файлов, пользователи могут легко узнать информацию о пакетах, установленных в системе, и проверить их целостность.

Использовать метаданные для описания пакетов, инструкций по их установке и т.п.

Каждый RPM-пакет включает в себя метаданные, описывающие компоненты пакета, версию, выпуск, размер, URL-адрес проекта, инструкции по установке и т.п.

Упаковывать оригинальные исходные тексты программ в исходные и двоичные пакеты.

RPM дает возможность взять оригинальные исходные тексты программы и упаковать их в исходные и двоичные пакеты пригодные для конечных пользователей. При этом, в состав исходных пакетов включаются оригинальные исходные тексты, все произведенные исправления и полный набор инструкций по сборке. Такое устройство облегчает сопровождение пакетов по мере появления новых версий программ.

Размещать пакеты в Yum-репозиториях.

Пакет может быть размещен в **Yum**-репозитории, позволяющем клиентам легко находить и устанавливать требуемые программы.

Подписывать пакеты с помощью цифровой подписи.

Пакет может содержать цифровую подпись, созданную с помощью подписывающего GPG-ключа. Цифровая подпись позволяет пользователям проверять аутентичность пакета.

1.2. Концепция коллекций программ

Коллекции программ дают возможность собирать и устанавливать в систему одновременно несколько версий одного и того же RPM-пакета. Коллекции программ не влияют на системные версии пакетов, установленные обычным менеджером RPM-пакетов.

Коллекции программ:

Не изменяют системные файлы.

Коллекции программ распространяются как наборы компонентов, которые реализуют все функциональные возможности программ без изменения системных файлов.

Спроектированы для избежания конфликтов с системными файлами.

Коллекции программ используют особую иерархию в файловой системе для избежания возможных конфликтов с файлами основной системы.

Не требуют изменений менеджера RPM-пакетов.
: Коллекции программ не требуют изменения менеджера RPM-пакетов, используемого основной системой.

Требуют лишь небольших изменений в spec-файле.
: Для преобразования обычного пакета в пакет коллекции программ надо произвести лишь небольшие изменения в spec-файле пакета.

Позволяют собирать обычные пакеты и пакеты коллекций из одних и тех же spec-файлов.
: Используя один и тот же spec-файл, можно собрать обычный пакет и пакет, входящий в коллекцию программ.

Позволяют использовать spec-файл из одной коллекции для сборки в другой коллекции.
: Один spec-файл может использоваться для сборки пакетов в разных коллекциях программ.

Обеспечивают уникальность именования входящих в коллекцию пакетов.
: Все пакеты, входящие в коллекцию программ, получают уникальные имена, зависящие от наименования данной коллекции.

Устраняют конфликты с пакетами обновлений.
: Механизм именования пакетов с учетом названия коллекции программ устраняет возможность конфликтов во время обновления системных пакетов.

Могут зависеть от других коллекций программ.
: Поскольку одна коллекция программ может зависеть от другой, можно определять многоуровневые зависимости.

1.3. Обеспечение поддержки коллекций программ

Для поддержки коллекций программ, то есть для того, чтобы их можно было подключать к текущей среде исполнения программ и осуществлять сборку коллекций программ, в системе должны быть установлены пакеты *scl-utils* и *scl-utils-build*.

Если пакеты *scl-utils* и *scl-utils-build* еще не установлены, то их нужно инсталлировать с помощью следующей команды, выполняемой с правами суперпользователя root:

```
yum install -y scl-utils scl-utils-build
```

Пакет *scl-utils* предоставляет утилиту **scl**, которая служит для работы с коллекциями программ (см. *Раздел 1.7, «Использование коллекций программ»*).

Пакет *scl-utils-build* предоставляет макросы необходимые для сборки коллекций программ (см. *Раздел 2.1, «Локальная сборка коллекций программ»*).

1.4. Иерархическая структура файловой системы

Для избежания конфликтов между коллекциями программ и файлами основной системы корневой каталог коллекций программ обычно располагается в **`/opt/`**. Использование каталога **`/opt/`** рекомендуется стандартом FHS[1].

Расположение корневого каталога можно изменить, установив значение макропеременной **`%_scl_prefix`** в spec-файле пакета. Например:

[1] Filesystem Hierarchy Standard (FHS) — стандарт на иерархическую структуру каталогов файлой системы.

```
%_scl_prefix /opt/provider
```

где *provider* — это наименование поставщика или разработчика программного обеспечения, зарегистрированное в Linux Foundation и подчиненной ей организации Linux Assigned Names and Numbers Authority (LANANA) и удовлетворяющее требованиям стандарта FHS.

Каждый разработчик программного обеспечения, занимающийся сборкой и распространением коллекций программ, должен использовать имя, удовлетворяющее требованиям стандарта FHS, и избегать возможные конфликты между коллекциями программ и установками основной системы.

Рекомендуется, чтобы иерархия файловой системы соответствовала следующему шаблону:

```
/opt/поставщик/приложение-версия/
```

Информация о стандарте на иерархическую структуру каталогов файловой системы доступна по адресу *http://www.pathname.com/fhs/*.

Информация об организации Linux Assigned Names and Numbers Authority доступна по адресу *http://www.lanana.org/*.

Ниже приведен пример схемы каталогов файловой системы двух коллекций программ — *Коллекция 1* и *Коллекция 2*:

```
opt
`-- provider
    |-- Коллекция 1
    |   |-- корневой каталог коллекции
    |   `-- скриплеты коллекции
    |
    `-- Коллекция 2
        |-- корневой каталог коллекции
        `-- скриплеты коллекции
```

Как показано в приведенном выше примере, каждая коллекция программ содержит два каталога: корневой каталог коллекции и каталог скриплетов коллекции (см. *Раздел 1.5, «Скриплеты коллекций программ»*).

1.5. Скриплеты коллекций программ

Скриплеты коллекции программ представляют собой небольшие командные файлы, которые меняют текущее состояние системной среды таким образом, что группа пакетов, входящая в коллекцию, получает предпочтение перед соответствующей группой обычных пакетов, установленных в системе.

Скриплеты коллекций программ используются утилитой **scl** (см. *Раздел 1.7, «Использование коллекций программ»*).

1.6. Схема организации пакетов

Каждая коллекция программ состоит из метапакета, устанавливающего некоторое подмножество других пакетов, и нескольких пакетов коллекции, названия которых зависят от наименования коллекции программ.

1.6.1. Метапакет

Каждая коллекция программ включает в себя метапакет, который устанавливает минимальное подмножество наиболее важных пакетов. Например, в число важных пакетов может входить интерпретатор языка Perl, но не входить модули расширений Perl. Метапакет содержит базовую иерархию файловой системы и предоставляет некоторые скриплеты коллекции программ.

Назначение метапакета состоит в том, чтобы гарантировать, что все важные пакеты коллекции программ установлены должным образом, и коллекцию можно использовать.

Метапакет создает следующие пакеты, которые тоже являются частью коллекции программ:

Главный пакет: %scl
: Главный пакет коллекции программ содержит зависимости основных пакетов, входящих в состав коллекции. Главный пакет не содержит каких-либо файлов.

  Например, если название коллекции программ — **`ruby-1.8.7`**, то макропеременная, определяющая название главного пакета, расширяется в значение:

```
ruby-1.8.7
```

Подпакет рабочей среды: *имя_коллекции*-runtime
: Подпакет рабочей среды в коллекции программ содержит ее файловую систему и скриплеты.

  Например, если название коллекции программ — **`ruby-1.8.7`**, то макропеременная, определяющая имя подпакета рабочей среды, расширяется в значение:

```
ruby-1.8.7-runtime
```

Подпакет сборки: *имя_коллекции*-build
: Подпакет сборки в коллекции программ предоставляет конфигурацию для ее сборки. Подпакет сборки не является обязательным.

  Например, если название коллекции программ — **`ruby-1.8.7`**, то макропеременная, определяющая имя подпакета сборки, расширяется в значение:

```
ruby-1.8.7-build
```

Пример метапакета

Приведенный ниже пример дает представление о том, как выглядит типичный метапакет коллекции программ:

```
%{!?scl:%global scl example}
%scl_package %scl

Summary: Package that installs %scl
Name: %scl_name
Version: 1
Release: 1%{?dist}
BuildArch: noarch
License: GPLv2+
Requires: %{scl_prefix}less
BuildRequires: scl-utils-build
```

```
%description
This is the main package for %scl Software Collection.

%package runtime
Summary: Package that handles %scl Software Collection.
Requires: scl-utils

%description runtime
Package shipping essential scripts to work with %scl Software Collection.

%package build
Summary: Package shipping basic build configuration

%description build
Package shipping essential configuration macros to build %scl Software Collection.

%install
rm -rf %{buildroot}
mkdir -p %{buildroot}%{_scl_scripts}/root
cat >> %{buildroot}%{_scl_scripts}/enable << EOF
export PATH=%{_bindir}:\$PATH
EOF
%scl_install

%files

%files runtime
%scl_files

%files build
%{_root_sysconfdir}/rpm/macros.%{scl}-config

%changelog
* Thu Jan 07 2012 John Doe <jdoe@example.com> 1-1
- Initial package
```

1.6.2. Именование пакетов внутри коллекции программ

Помимо минимального подмножества пакетов, поставляемых метапакетом, в состав коллекции программ могут входить и другие пакеты.

Пакет коллекции программ: *имя_коллекции-имя_пакета*

Коллекция программ может содержать пакеты, имена которых включают в себя название коллекции, а файлы при установке помещаются в иерархию файловой системы, отведенную данной коллекции.

Например, если коллекция программ называется **`ruby-1.8.7`**, то пакет *ruby-1.8.7.352-2.1.fc17.x86_64.rpm* при включении в эту коллекцию будет именоваться так:

```
ruby-1.8.7-ruby-1.8.7.352-2.1.fc17.x86_64.rpm
```

1.7. Использование коллекций программ

Утилита **scl** служит для переключения текущей системной среды на использование указанной коллекции программ и для запуска приложений в среде коллекций программ.

В общем виде использование утилиты **scl** выглядит так:

```
scl action software_collection_1 software_collection_2 command
```

1.7.1. Запуск приложения

Например, чтобы запустить **Perl** с опцией **`--version`** в среде коллекции программ **software_collection_1**, надо выполнить следующую команду:

```
scl enable software_collection_1 'perl --version'
```

1.7.2. Запуск интерпретатора команд в среде нескольких коллекций программ

Запуск интерпретатора команд **Bash** в среде нескольких коллекций программ осуществляется следующим образом:

```
scl enable software_collection_1 software_collection_2 bash
```

Приведенная выше команда задействует две коллекции программ — **software_collection_1** и **software_collection_2**.

1.7.3. Запуск команд из файла

Для выполнения в среде коллекции программ нескольких команд, хранящихся в файле, используется конвейер:

```
cat cmd | scl enable software_collection_1 -
```

В приведенном выше примере команды, хранящиеся в командном файле **`cmd`**, выполняются в среде коллекции программ **software_collection_1**.

Создание пакетов для коллекций программ

В этой главе описывается процесс создания пакетов коллекций программ и пакетов приложений, использующих коллекции программ.

2.1. Локальная сборка коллекций программ

Для сборки коллекции программ на локальной системе используется следующая команда:

```
rpmbuild -bb package.spec --define 'scl name'
```

Отличие между показанной выше командой и стандартной командой для сборки обычных пакетов (**rpmbuild -bb *package.spec***) состоит в том, что при сборке коллекции программ к команде **rpmbuild** добавляется опция **--define**.

Опция **--define** определяет значение макропеременной **scl**, которое используется как название коллекции внутри spec-файла пакета, подготовленного для включения в коллекцию программ.

2.2. Макросы для поддержки коллекций программ

Макропеременная **scl**, используемая при сборке пакетов коллекции программ, обеспечивает перемещение файловой стуктуры пакетов коллекции в каталог файловой системы, отведенный исключительно для данной коллекции программ.

Кроме этого, макропеременная **scl** используется в метапакете коллекции и служит для определения значений других макропеременных в среде сборки пакетов коллекции программ.

Когда макросы коллекций программ используются в spec-файле, перед ними необходим префикс — **%{?scl:*макровыражение*}**. Например:

```
%{?scl:Requires:%scl_runtime}
```

В приведенном выше примере макропеременная **%scl_runtime** задает значение тега **Requires**. Префикс **%{?scl:** должен стоять перед макровыражением, включающим тег и макропеременную.

2.2.1. Макросы специфичные для коллекции программ

В представленной ниже таблице приведен полный перечень макросов, значения которых специально определяются для каждой коллекции программ.

Таблица 2.1. Макросы со специфичными значениями для коллекции программ

Макрос	Описание	Пример значения
%scl_name	Название коллекции программ	**software_collection_1**
%pkg_name	Имя оригинального пакета	**perl**
%_scl_prefix	Корневой каталог коллекции программ (не путать с корневым каталогом для установки пакета)	**/opt/provider/**

Макрос	Описание	Пример значения
%_scl_scripts	Местоположение скриплетов коллекции программ	**/opt/provider/software_collection_1/**
%_scl_root	Корневой каталог для установки пакета	**/opt/provider/software_collection_1/root/**

2.2.2. Неспецифичные макросы для коллекции программ

В приведенной ниже таблице перечислены макросы, которые не являются специфичными для какой-либо коллекции программ. Их значения не указывают в область файловой системы, выделенную для коллекции программ. В названиях этих макросов используется префикс **_root**.

Таблица 2.2. Макросы, имеющие неспецифические значения для коллекции программ

Макрос	Описание	Перемещаемый	Пример значения
%_root_prefix	Макрос **%_prefix** для коллекции программ	нет	**/usr/**
%_root_exec_prefix	Макрос **%_exec_prefix** для коллекции программ	нет	**/usr/**
%_root_bindir	Макрос **%_bindir** для коллекции программ	нет	**/usr/bin/**
%_root_sbindir	Макрос **%_sbindir** для коллекции программ	нет	**/usr/sbin/**
%_root_datadir	Макрос **%_datadir** для коллекции программ	нет	**/usr/share/**
%_root_sysconfdir	Макрос **%_sysconfdir** для коллекции программ	нет	**/etc/**
%_root_libexecdir	Макрос **%_libexecdir** для коллекции программ	нет	**/usr/libexec/**
%_root_sharedstatedir	Макрос **%_sharedstatedir** для коллекции программ	нет	**/usr/com/**
%_root_localstatedir	Макрос **%_localstatedir** для коллекции программ	нет	**/usr/var/**
%_root_includedir	Макрос **%_includedir** для коллекции программ	нет	**/usr/include/**

Макрос	Описание	Перемещаемый	Пример значения
%_root_infodir	Макрос **%_infodir** для коллекции программ	нет	**/usr/share/info/**
%_root_mandir	Макрос **%_mandir** для коллекции программ	нет	**/usr/share/man/**
%_root_initddir	Макрос **%_initddir** для коллекции программ	нет	**/etc/rc.d/init.d/**
%_**root_libdir**	Макрос **%_libdir** для коллекции программ. Этот макрос не работает, если метапакет коллекции программ является платформно-независимым	нет	**/usr/lib/**

2.3. Преобразование обычного spec-файла

Ниже приведена последовательность шагов для трансформации обычного spec-файла в spec-файл для коллекции программ. Полученный в результате spec-файл может использоваться для сборки обычного пакета и пакета коллекции программ.

Процедура 2.1. Трансформация обычного spec-файла в spec-файл для коллекции программ

1. В начале spec-файла перед его преамбулой следует поместить макроопределение **%scl_package** :

```
%{?scl:%scl_package имя_пакета}
```

2. В spec-файле рекомендуется определить макропеременную %pkg_name на тот случай, когда пакет собирается не для коллекции программ:

```
%{!?scl:%global pkg_name %{name}}
```

   Затем макропеременную %pkg_name можно использовать везде, где в spec-файле требуется сослаться на оригинальное имя пакета. Такой spec-файл можно использовать при сборке обычного пакета и коллекции программ.

3. В преамбуле spec-файла тег **Name** надо изменить следующим образом:

```
Name: %{?scl_prefix}имя_пакета
```

4. Для того, чтобы сделать все существенно важные пакеты коллекции программ зависимыми от главного метапакета коллекции, в их spec-файлы после тегов **BuildRequires** или **Requires** надо добавить такое макровыражение:

```
%{?scl:Requires: %scl_runtime}
```

5. Следующее изменение spec-файла касается раздела **%prep**. В нем надо отредактировать обращение к макросу **%setup** таким образом, чтобы макрос мог работать с измененным именем пакета при сборке в среде коллекции программ:

```
%setup -q -n %{pkg_name}-%{version}
```

Пример трансформированного spec-файла

Следующий пример показывает изменения в spec-файле, преобразованном для коллекции программ, по сравнению с обычным spec-файлом:

```
--- a/less.spec
+++ b/less.spec
@@ -1,10 +1,13 @@
+%{?scl:%scl_package less}
+%{!?scl:%global pkg_name %{name}}
+
 Summary: A text file browser similar to more, but better
-Name: less
+Name: %{?scl_prefix}less
 Version: 444
 Release: 7%{?dist}
 License: GPLv3+
 Group: Applications/Text
-Source: http://www.greenwoodsoftware.com/less/%{name}-%{version}.tar.gz
+Source: http://www.greenwoodsoftware.com/less/%{pkg_name}-%{version}.tar.gz
 Source1: lesspipe.sh
 Source2: less.sh
 Source3: less.csh
@@ -19,6 +22,7 @@ URL: http://www.greenwoodsoftware.com/less/
 Requires: groff
 BuildRequires: ncurses-devel
 BuildRequires: autoconf automake libtool
+%{?scl:Requires: %scl_runtime}

 %description
 The less utility is a text file browser that resembles more, but has
@@ -31,7 +35,7 @@ You should install less because it is a basic utility for viewing text
 files, and you'll use it frequently.

 %prep
-%setup -q
+%setup -q -n %{pkg_name}-%{version}
 %patch1 -p1 -b .Foption
 %patch2 -p1 -b .search
 %patch4 -p1 -b .time
@@ -51,16 +55,16 @@ make CC="gcc $RPM_OPT_FLAGS -D_GNU_SOURCE -D_LARGEFILE_SOURCE -
D_LARGEFILE64_SOU
 %install
 rm -rf $RPM_BUILD_ROOT
 make DESTDIR=$RPM_BUILD_ROOT install
-mkdir -p $RPM_BUILD_ROOT/etc/profile.d
+mkdir -p $RPM_BUILD_ROOT%{_sysconfdir}/profile.d
 install -p -c -m 755 %{SOURCE1} $RPM_BUILD_ROOT/%{_bindir}
-install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT/etc/profile.d
-install -p -c -m 644 %{SOURCE3} $RPM_BUILD_ROOT/etc/profile.d
-ls -la $RPM_BUILD_ROOT/etc/profile.d
+install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{_sysconfdir}/profile.d
+install -p -c -m 644 %{SOURCE3} $RPM_BUILD_ROOT%{_sysconfdir}/profile.d
+ls -la $RPM_BUILD_ROOT%{_sysconfdir}/profile.d

 %files
 %defattr(-,root,root,-)
 %doc LICENSE
```

```
-/etc/profile.d/*
+%{_sysconfdir}/profile.d/*
 %{_bindir}/*
 %{_mandir}/man1/*
```

2.4. Использование коллекций программ в приложениях

Для использования коллекции программ в приложении нужно изменить теги **`BuildRequires`** и **`Requires`** в spec-файле приложения таким образом, чтобы они должным образом отражали зависимости от коллекций программ.

Например, чтобы указать зависимости от двух коллекций программ — **software_collection_1** и **software_collection_2**, надо добавить в spec-файл приложения три строки:

```
BuildRequires: scl-utils-build
Requires: %scl_require software_collection_1
Requires: %scl_require software_collection_2
```

2.5. Установка коллекции программ

Для установки коллекции программ надо проинсталлировать ее главный метапакет. При этом можно использовать обычные средства управления пакетами, такие как **Yum** или **PackageKit**, поскольку коллекции программ полностью совместимы с менеджером RPM-пакетов. Например, для установки коллекции программ **`software_collection_1`** с помощью **Yum** надо выполнить следующую команду:

```
yum install -y software_collection_1
```

Эта команда автоматически проинсталлирует все пакеты, входящие в данную коллекцию программ. Кроме того, при установке приложения, зависящего от коллекции программ, эта коллекция будет установлена наряду с другими зависимостями данного приложения.

Подробная информация о приложениях **Yum** и **PackageKit** представлена в *Руководстве системного администратора Fedora 17*[1].

2.6. Вывод списка установленных коллекций программ

Для получения списка коллекций программ, которые в настоящее время установлены в системе, служит команда

```
scl --list
```

2.7. Поддержка служб в коллекциях программ

Необходимо предпринять специальные меры, чтобы с помощью обычных системных средств, таких как утилиты **service** и **chkconfig**, можно было управлять службами, предоставляемыми коллекциями программ или связанными с ними приложениями.

Для избежания возможных конфликтов имен между системными службами и службами, входящими в состав коллекций программ, в spec-файлах соответствующих пакетов нужно внести изменения в раздел **`%install`**:

[1] http://docs.fedoraproject.org/en-US/Fedora/17/html/System_Administrators_Guide/index.html

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?scl:
%_sysconfdir}/rc.d/%{?scl_prefix}имя_службы
```

При такой конфигурации к тем службам, которые относятся к коллекции программ, можно обращаться следующим образом:

```
%{?scl_prefix}имя_службы
```

2.8. Поддержка библиотек в коллекциях программ

Если библиотеки, поставляемые в пакете коллекции программ, предназначены для использования только в этой коллекции и дополняют уже имеющиеся в системе библиотеки, то в spec-файле метапакета коллекции надо установить значение переменной среды `LD_LIBRARY_PATH`:

```
export LD_LIBRARY_PATH=%{_libdir}:\$LD_LIBRARY_PATH
```

Такая конфигурация при подключении коллекции программ к текущей среде исполнения приложений отдает предпочтение библиотекам, входящим в коллекцию.

Если в пакет включены библиотеки, которые могут использоваться вне среды коллекции программ, то ссылки на их расположение должны быть добавлены в каталог **`/etc/ld.so.conf.d/`**.

Предупреждение

Не следует использовать каталог **`/etc/ld.so.conf.d/`** для системных библиотек. Каталог **`/etc/ld.so.conf.d/`** рекомендуется использовать только для тех библиотек, которые недоступны в основной системе, так как в противном случае, вместо системных библиотек предпочтение получат версии библиотек, поставляемых с коллекцией программ, а это может привести к нежелательным результатам при работе системных приложений, включая их аварийную остановку и потерю данных.

Процедура 2.2. Использование **`/etc/ld.so.conf.d/`** с библиотеками из коллекции программ

1. Нужно создать файл **`libs.conf`** и внести соответствующие изменения в spec-файл:

   ```
   SOURCE2: %{?scl_prefix}libs.conf
   ```

2. В файле **`libs.conf`** надо перечислить каталоги, где располагаются версии библиотек, распространяемые вместе с коллекцией программ. Например:

   ```
   /opt/provider/software_collection_1/root/usr/lib64/
   ```

   В приведенном примере в списке указан каталог **`/usr/lib64/`**, входящий в коллекцию программ **software_collection_1**.

3. Раздел **%install** в spec-файле пакета надо отредактировать так, чтобы файл **libs.conf** инсталлировался следующим образом:

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?scl:
%_sysconfdir}/ld.so.conf.d/
```

2.9. Поддержка файлов .pc в коллекциях программ

Файлы .pc — это файлы метаданных для программы **pkg-config**, содержащие информацию о библиотеках, установленных в системе. При сборке пакета с файлами .pc, которые предполагается использовать только в среде коллекции программ или в дополнение к файлам .pc, уже установленным в системе, нужно изменить значение переменной среды PKG_CONFIG_PATH. В зависимости от того, что определено в файлах .pc, значение переменной PKG_CONFIG_PATH должно быть модифицировано с помщью макроса %{_libdir} (обычно указывающего путь к каталогу библиотек **/usr/lib/** или **/usr/lib64/**) или с помощью макроса %{_datadir} (обычно указывающего путь к каталогу **/usr/share/**).

Если в файле .pc указан путь к каталогу библиотек, то в spec-файле метапакета коллекции программ переменную среды PKG_CONFIG_PATH надо изменить следующим образом:

```
export PKG_CONFIG_PATH=%{_libdir}/pkgconfig:\$PKG_CONFIG_PATH
```

Если в файле .pc указан путь к каталогу с данными, независящими от архитектуры процессора, то в spec-файле метапакета коллекции программ переменную среды PKG_CONFIG_PATH надо изменить так:

```
export PKG_CONFIG_PATH=%{_datadir}/pkgconfig:\$PKG_CONFIG_PATH
```

При такой конфигурации файлы .pc, включенные в коллекцию программ, будут пользоваться предпочтением при подключении коллекции к текущей среде исполнения программ.

При некоторых обстоятельствах в составе коллекции программ бывает полезно иметь скрипт-обертку, который устанавливается в системный каталог, например, в **/usr/bin/**, и позволяет использовать средства, поставляемые в коллекции программ. В этом случае надо принять меры, чтобы файлы .pc были видны системе, даже если коллекция программ не подключена.

Чтобы система могла использовать файлы .pc из неподключенной коллекции программ, в переменной среды PKG_CONFIG_PATH нужно указать путь доступа к этим файлам. В зависимости от содержания файлов .pc значение переменной PKG_CONFIG_PATH определяется с помощью макроса %{_libdir} (указывающего положение каталога библиотек) или с помощью макроса %{_datadir} (указывающего положение каталога данных, независящих от архитектуры процессора).

Процедура 2.3. Изменение переменной среды PKG_CONFIG_PATH с помощью макроса %{_libdir}

1. Чтобы измениеть значение переменной среды PKG_CONFIG_PATH с помощью макроса %{_libdir}, надо создать скрипт **/etc/profile.d/*имя.sh***. Этот скрипт исполняется всякий раз при запуске интерпретатора команд.

   Например, можно создать файл

   ```
   %{?scl_prefix}pc-libdir.sh
   ```

2. Скрипт **pc-libdir.sh** изменяет переменную PKG_CONFIG_PATH так, чтобы она указывала на файлы .pc в коллекции программ:

```
export PKG_CONFIG_PATH=%{_libdir}/pkgconfig:/opt/provider/имя_коллекции/путь/к/файлам_pc
```

3. Данный файл надо добавить в spec-файл пакета коллекции программ:

```
SOURCE2: %{?scl_prefix}pc-libdir.sh
```

4. Раздел **%install** в spec-файле пакета коллекции программ надо изменить так, чтобы данный файл устанавливался в каталог **/etc/profile.d/**:

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?scl:
%_sysconfdir}/profile.d/
```

Процедура 2.4. Изменение переменной среды PKG_CONFIG_PATH с помощью макроса %{_datadir}

1. Чтобы изменить значение переменной среды PKG_CONFIG_PATH с помощью макроса %{_datadir}, надо создать скрипт **/etc/profile.d/*имя.sh***. Этот скрипт исполняется всякий раз при запуске интерпретатора команд.

   Например, можно создать файл

```
%{?scl_prefix}pc-datadir.sh
```

2. Скрипт **pc-datadir.sh** изменяет переменную PKG_CONFIG_PATH так, чтобы она указывала на файлы .pc в коллекции программ:

```
export PKG_CONFIG_PATH=%{_datadir}/pkgconfig:/opt/provider/имя_коллекции/путь/к/
файлам_pc
```

3. Данный файл надо добавить в spec-файл пакета коллекции программ:

```
SOURCE2: %{?scl_prefix}pc-datadir.sh
```

4. Раздел **%install** в spec-файле пакета коллекции программ надо изменить так, чтобы данный файл устанавливался в каталог **/etc/profile.d/**:

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?scl:
%_sysconfdir}/profile.d/
```

2.10. Поддержка переменной MANPATH в коллекциях программ

Чтобы команда **man** могла показывать справочные страницы из подключенной коллекции программ, надо добавить путь доступа к этим страницам в переменную среды MANPATH.

Для изменения переменной среды MANPATH в spec-файл метапакета коллекции программ должна быть добавлена следующая строка:

```
export MANPATH=${MANPATH}:%{_mandir}
```

При этом значение макроса **%{_mandir}** указывает путь доступа к справочным страницам внутри коллекции программ. Таким образом, они остаются невидимыми, пока коллекция программ не подключена.

В некоторых случаях в составе коллекции программ может поставляться скрипт-обертка, устанавливаемый в системный каталог, например, в **/usr/bin/**, который позволяет использовать средства коллекции программ без ее подключения. В этом случае надо принять меры, чтобы справочные страницы оставались доступны, даже если коллекция программ не подключена.

Чтобы команда **man** могла показывать справочные страницы из неподключенной коллекции программ, надо указать в переменной среды MANPATH путь доступа к справочным страницам этой коллекции.

Процедура 2.5. Установка значения переменной среды MANPATH для неподключенной коллекции программ.

1. Чтобы изменить значение переменной среды MANPATH, надо создать скрипт **/etc/profile.d/*имя.sh***. Этот скрипт исполняется всякий раз при запуске интерпретатора команд.

   Например, можно создать файл

   ```
   %{?scl_prefix}manpage.sh
   ```

2. Скрипт **manpage.sh** изменяет переменную MANPATH так, чтобы она указывала на каталог со справочными страницами:

   ```
   export MANPATH=${MANPATH}:/opt/provider/имя_коллекции/путь/к/справочным_страницам
   ```

3. Данный файл надо добавить в spec-файл пакета коллекции программ:

   ```
   SOURCE2: %{?scl_prefix}manpage.sh
   ```

4. Раздел **%install** в spec-файле пакета коллекции программ надо изменить так, чтобы данный файл устанавливался в каталог **/etc/profile.d/**:

   ```
   %install
   install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?scl:
   %_sysconfdir}/profile.d/
   ```

2.11. Поддержка SystemTap в коллекциях программ

SystemTap — это инструментальная платформа для зондирования процессов и ядра Linux. При сборке пакета, содержащего каталог для кэширования данных SystemTap, которые предполагается использовать только в среде коллекции программ или как дополнение к общесистемным данным SystemTap, в spec-файле метапакета коллекции программ надо изменить переменную среды XDG_DATA_DIRS следующим образом:

```
export XDG_DATA_DIRS=%{_datadir}:\$XDG_DATA_DIRS
```

Такая конфигурация при подключении коллекции программ к текущей среде исполнения приложений отдает предпочтение файлам данных SystemTap, входящим в коллекцию, перед системыми файлами SystemTap.

В некоторых случаях в составе коллекции программ может поставляться скрипт-обертка, устанавливаемый в системный каталог, например, в **`/usr/bin/`**, который позволяет использовать средства коллекции программ без ее подключения. В этом случае надо принять меры, чтобы данные SystemTap оставались доступны, даже если коллекция программ не подключена.

Чтобы система могла использовать файлы данных SystemTap из неподключенной коллекции программ, надо указать в переменной среды `XDG_DATA_DIRS` путь доступа к каталогу с файлами SystemTap из этой коллекции.

Процедура 2.6. Установка значения переменной среды `XDG_DATA_DIRS` для неподключенной коллекции программ.

1. Чтобы изменить значение переменной среды `XDG_DATA_DIRS`, надо создать скрипт **`/etc/profile.d/`*`имя.sh`***. Этот скрипт исполняется всякий раз при запуске интерпретатора команд.

   Например, можно создать файл

   ```
   %{?scl_prefix}systemtap.sh
   ```

2. Скрипт **`systemtap.sh`** изменяет переменную `XDG_DATA_DIRS` так, чтобы она указывала на каталог с файлами данных SystemTap:

   ```
   export XDG_DATA_DIRS=${XDG_DATA_DIRS}:/opt/provider/имя_коллекции/путь/к/
   данным_systemtap
   ```

3. Данный файл надо добавить в spec-файл пакета коллекции программ:

   ```
   SOURCE2: %{?scl_prefix}systemtap.sh
   ```

4. Раздел **`%install`** в spec-файле пакета коллекции программ надо изменить так, чтобы данный файл устанавливался в каталог **`/etc/profile.d/`**:

   ```
   %install
   install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?scl:
   %_sysconfdir}/profile.d/
   ```

2.12. Поддержка заданий службы cron в коллекциях программ

Для регулярного запуска задач в среде коллекции программ можно использовать специально выделенную службу или задания службы `cron`. *Раздел 2.7, «Поддержка служб в коллекциях программ»* дает представление об организации работы со службами и скриптами для их запуска, поставляемыми с коллекциями программ.

Процедура 2.7. Запуск регулярных задач с помощью заданий службы `cron`.

1. Для регулярного запуска задач надо поместить файл **`crontab`**, относящийся к коллекции программ, в каталог **`/etc/cron.d/`** с именем коллекции в качестве префикса.

Например, можно создать файл

```
%{?scl_prefix}crontab
```

2. Содержание файла **`crontab`** должно соответствовать стандартному формату файла заданий службы `cron`. Например:

```
0 1 * * Sun root /opt/provider/имя_коллекции/архитектура/usr/bin/имя_задания_cron
```

3. Данный файл надо добавить в spec-файл пакета коллекции программ:

```
SOURCE2: %{?scl_prefix}crontab
```

4. Раздел **`%install`** в spec-файле пакета коллекции программ надо изменить так, чтобы данный файл устанавливался в системный каталог **`/etc/cron.d/`**

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?scl:
%_sysconfdir}/cron.d/
```

2.13. Поддержка ротации журналов в коллекциях программ

Ротация журнальных файлов в коллекции программ или в приложении, связанном с коллекцией программ, может осуществляться с помощью утилиты **logrotate**.

Процедура 2.8. Управление журнальными файлами с помощью **logrotate**.

1. Для управления журнальными файлами с помощью утилиты **logrotate** надо поместить конфигурационный файл **`logrotate`**, относящийся к коллекции программ, в системный каталог **`/etc/logrotate.d/`**.

   Например, можно создать файл

```
%{?scl_prefix}logrotate
```

2. Содержание файла **`logrotate`** должно соответствовать стандартному формату файла заданий **logrotate**:

```
/opt/provider/имя_коллекции/var/log/имя_приложения.log {
      missingok
      notifempty
      size 30k
      yearly
      create 0600 root root
  }
```

3. Данный файл надо добавить в spec-файл пакета коллекции программ:

```
SOURCE2: %{?scl_prefix}logrotate
```

4. Раздел **%install** в spec-файле пакета коллекции программ надо изменить так, чтобы данный файл устанавливался в системный каталог **/etc/logrotate.d/**

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?scl:
%_sysconfdir}/logrotate.d/
```

2.14. Поддержка файлов блокировок в коллекциях программ

Для устранения потенциальных конфликтов с системными версиями приложений или служб файлы блокировок, относящиеся к коллекции программ, следует создавать в иерархии файловой системы **/opt/*provider*/*имя_коллекции*/**.

Если необходимо предотвратить запуск приложений или служб коллекции программ в то время, пока работают соответствующии им системные версии, то приложения или службы, требующие блокировки, должны создавать файлы блокировок в системном каталоге **/var/lock/**, а не в каталоге коллекции программ **/opt/*provider*/*имя_коллекции*/var/lock/**. Таким образом, файлы блокировок, принадлежащие коллекции программ, не будут игнорироваться системными приложениями или службами. Файлы блокировок не надо переименовывать, их имена должны оставаться теми же, что и у системных версий.

Если желательно, чтобы версии приложений или служб коллекции программ могли работать одновременно с соответствующими системными версиями (когда эти версии не конфликтуют), то файлы блокировок, относящиеся к приложениям или службам коллекции программ, надо создавать в каталоге коллекции программ **/opt/*provider*/*имя_коллекции*/var/lock/**.

2.15. Поддержка конфигурационных файлов в коллекциях программ

Для избежания всевозможных конфликтов с системными версиями конфигурационных файлов соответствующие конфигурационные файлы, относящиеся к коллекции программ, следует создавать в иерархии файловой системы **/opt/*provider*/*имя_коллекции*/**.

Если по каким-либо причинам конфигурационные файлы не могут быть внутри **/opt/*provider*/*имя_коллекции*/**, то необходимо должным образом подготовить для них альтернативное местоположение. Для многих программ это можно сделать во время сборки или при инсталляции.

2.16. Поддержка модулей ядра в коллекциях программ

Поскольку модули ядра Linux обычно привязаны к определенной версии ядра, то следует проявлять особую осторожность при включении модулей ядра в пакет коллекции программ, так как система управления пакетами Fedora не производит автоматическое обновление или установку обновленных версий модулей ядра при установке обновленной версии ядра Linux. Следующие рекомендации позволяют упростить включение модулей ядра в коллекцию программ:

1. Имя пакета с модулями ядра должно включать в себя версию ядра.

2. Тег **Requires**, используемый в spec-файле пакета с модулями ядра, должен включать версию и номер выпуска пакета ядра (в формате **kernel-*версия*-*выпуск***).

2.17. Поддержка SELinux в коллекциях программ

Поскольку коллекции программ предназначены для установки пакетов в нетрадиционные каталоги, в этих каталогах надо расставить метки SELinux, позволяющие контролировать работу программ.

Если иерархическая структура файловой системы пакета коллекции программ имитирует структуру файловой системы соответствующего обычного пакета, то для установки меток SELinux можно использовать команды **`semanage fcontext`** и **`restorecon`**.

Например, если каталог коллекции программ **`/opt/provider/software_collection_1/x86_64/root/usr/`** имитирует каталог **`/usr/`** обычного пакета, то поставить метки SELinux можно так:

```
semanage fcontext -a -e /usr /opt/provider/software_collection_1/x86_64/root/usr
```

```
restorecon -R -v /opt/provider/software_collection_1/x86_64/root/usr
```

Приведенные выше команды обеспечивают установку меток SELinux на все каталоги и файлы в **`/opt/provider/software_collection_1/x86_64/root/usr/`**, как если бы они располагались в каталоге **`/usr/`**.

Для использования команд **`semanage fcontext`** и **`restorecon`** в коллекции программ их надо добавить в раздел **`%post`** в spec-файле пакета коллекции.

Дополнительные ресурсы

Ниже перечислены дополнительные информационные ресурсы, посвященные коллекциям программ.

Системная документация

- **scl**(1) – справочная страница по утилите **scl**, используемой для подключения коллекций программ и запуска приложений в их среде.
- **scl --help** – общая информация об использовании утилиты **scl** для работы с коллекциями программ и запуска приложений в их среде.
- **rpmbuild**(8) – справочная страница по утилите **rpmbuild**, используемой для сборки как двоичных, так и исходных пакетов программного обеспечения.

Документация в сети

- *Fedora 17 Installation Guide*[1] – *Руководство по установке Fedora 17* предоставляет детальную информацию о загрузке установочных носителей, а так же о процессе инсталляции и обновлении системы.
- *Fedora 17 System Administrator's Guide*[2] — *Руководство системного администратора* Fedora 17 предоставляет важную информацию о установке, конфигурации и администрировании операционной системы Fedora 17.

[1] http://docs.fedoraproject.org/en-US/Fedora/17/html/Installation_Guide/index.html
[2] http://docs.fedoraproject.org/en-US/Fedora/17/html/System_Administrators_Guide/index.html

Приложение A. История изменений

Издание 1-0 **Tue Jun 19 2012** **Petr Kovář** *pkovar@redhat.com*

Software Collections Guide, издание 1-0.

Издание 0-0 **Thu Feb 23 2012** **Petr Kovář** *pkovar@redhat.com*

Первоначальное создание книги.