Визитка

ДЕНИС СИПАКОВ, к. ф.-м. н., старший архитектор ЗАО «РОСА», занимается автоматизацией разработки ОС «РОСА»

Что такое дистрибутив Linux?

Разработка дистрибутива Linux на примере РОСЫ. Часть 2

В первой части статьи [1] были рассмотрены организационные вопросы, связанные с разработкой РОСЫ. Во второй части мы затронем различные технические аспекты, многие из них являются общими с другими дистрибутивами Linux

Управление ПО в Linux

Современный дистрибутив Linux общего назначения представляет собой согласованный и самодостаточный набор ПО, включающий не только системные компоненты и библиотеки, но и различные приложения, средства настройки системы и другие полезные компоненты. Для управления всем этим множеством программных составляющих (установки, удаления и обновления ПО при сохранении целостности системы) большинство дистрибутивов использует специализированные инструменты управления ПО.

В РОСЕ используется система, основанная на распространении приложений и прочих программных компонентов в виде прекомпилированных пакетов – файлов специального формата, включающих в себя архив со скомпилированными файлами приложения, а также различные метаданные:

- описание, имя и версию пакета;
- цифровую подпись;
- зависимости – перечень компонентов системы, которые необходимо установить для корректного функционирования приложения (вообще зависимость – это просто текстовая строка, она может содержать имена других пакетов, имена библиотек либо просто некоторое абстрактное название, главное, чтобы в системе нашелся пакет, предоставляющий такую зависимость);
- список зависимостей, которые можно удовлетворить установкой этого пакета;
- …и другую вспомогательную информацию.

Наиболее распространенными форматами пакетов являются Deb и RPM. В РОСЕ используется RPM5 – ответвление «классического» RPM, используемого в таких популярных системах, как Red Hat, Fedora и SUSE.

Каждый программный компонент, входящий в дистрибутив, должен быть оформлен в виде пакета. Все пакеты дистрибутива помещаются в сетевые хранилища – репозитории, с которыми работают инструменты управления ПО дистрибутива, позволяющие пользователям просмотреть список доступных приложений с подробными описаниями и установить их. Подготовка пакетов входит в задачи мэйнтейнеров (англ. maintainer), занимающихся интеграцией ПО в систему.

Сборка пакетов

Формирование (сборка) пакета фактически сводится к сборке соответствующего программного компонента и размещению собранных файлов в структуре каталогов, имитирующих корневую файловую систему. Далее эти файлы архивируются, архив снабжается необходимыми метаданными, подписывается и выдается пользователю.

Сборка RPM-пакетов осуществляется инструментом rpmbuild на основе исходного кода приложения, патчей от мэйнтейнеров (специфичных для конкретной системы) и так называемого spec-файла, содержащего служебную информацию для rpmbuild. Именно в spec-файле указываются практически все метаданные пакета – имя, версия, зависимости и прочее. Также важным атрибутом spec-файла является список пакетов, которые должны быть установлены в системе для того, чтобы собрать данное приложение, – так называемые сборочные зависимости. Наличие сборочных зависимостей позволяет системе управления пакетами дистрибутива автоматически поставить все необходимое для сборки того или иного приложения.

Один из главных вопросов, встающих перед разработчиками дистрибутива: где именно производить сборку пакетов? Есть несколько основных вариантов.

Локальная сборка

Самый простой способ создать пакет – это собрать его в собственной системе, установленной на реальной или виртуальной машине, непосредственно запустив rpmbuild на соответствующем наборе входных данных. Для подготовки пакетов рекомендуется использовать ту же ОС, для которой собирается пакет, хотя нередко в одном дистрибутиве можно подготавливать пакеты и для других ОС (особенно если целевая система близка к текущей в плане инструментария и настроек). Однако даже при использовании «родной» системы возможно возникновение ряда проблем.

Во-первых, при локальной сборке непосредственно в работающей системе используются штатный инструментарий и библиотеки. При этом на сборку могут влиять настройки конкретной системы, которые пользователь сделал

«под себя», например, флаги компиляции. Поэтому у разных разработчиков из одних и тех же исходных данных могут получаться разные пакеты.

Другим недостатком такой сборки является отсутствие проверки полноты набора сборочных зависимостей пакета. Если этот перечень не полон, то у одних пользователей пакет собираться не будет (поскольку нужные пакеты в системе отсутствуют, а rpmbuild не просит их установить), а у других может и собраться – в случае если нужные пакеты уже были установлены ранее. Нарушение полноты сборочных зависимостей не критично, но может серьезно усложнить работу.

Наконец, неаккуратность и ошибки при составлении spec-файлов могут приводить к печальным последствиям, ведь rpmbuild работает от лица пользователя и имеет доступ ко всем его файлам и каталогам. Небольшая опечатка в скрипте сборки может привести к удалению файлов пользователя, замусориванию файловой системы и прочим неприятным вещам.

Подобные проблемы можно решить, собирая пакеты в некоторой минималистской виртуальной среде. Например, в качестве такого окружения можно использовать виртуальную машину с дистрибутивом, установленным в минимальной конфигурации, и перед сборкой каждого пакета производить откат состояния виртуальной машины к снимку с изначальной минимальной конфигурацией.

Однако использование виртуальной машины требует достаточно много ресурсов, и более популярным способом является применение минимального chroot-окружения. Chroot – это простой способ организовать в Linux (и других UNIX-подобных системах) песочницу, имитирующую ОС. Способ заключается в смене корневого каталога, так что программа, запущенная в chroot-окружении, имеет доступ только к файлам и каталогам самого окружения. Если при этом в новом корневом каталоге воссоздать всю структуру системных каталогов и файлов, то программа будет полагать, что имеет дело с полноценной ОС, но все операции станут производиться внутри chroot-каталога. В частности, внутри chroot можно использовать средства управления пакетами для установки приложений непосредственно в chroot.

Для целей сборки пакетов формируется chroot-окружение, в которое устанавливаются базовые системные компоненты, необходимый инструментарий, а также все сборочные зависимости собираемого пакета. После этого внутри chroot запускается rpmbuild, и происходит «чистая» сборка пакета, на которую не влияют настройки основной системы.

Ручное формирование chroot для сборки пакета – достаточно нудное занятие, однако существуют инструменты, полностью автоматизирующие процесс сборки пакетов в chroot. В РОСЕ для этого используется программа mock-urpm, являющаяся модификацией mock (применяемой в Fedora и родственных ей системах). Отмечу, что операция смены корневого каталога может быть выполнена только пользователем с правами root (аналог Администратора в Windows). Однако работать внутри chroot-окружения с такими правами не рекомендуется, поскольку выйти за пределы этого окружения, имея права root, не так уж и сложно (см., например, http://www.bpfh.net/simes/computing/chroot-break.html). Поэтому после перехода в новое окружение рекомендуется понижать права до минимально необходимых для работы. В частности, mock и mock-urpm собирают пакеты в chroot-окружении от лица обычного пользователя вместо root.

Хотя сборка пакетов в chroot-окружении более удобна, чем в виртуальной машине, она все-таки остается ресурсоемкой. Формирование окружения «с нуля» может занимать достаточно много времени (иногда существенно больше, чем собственно сборка), особенно при большом числе сборочных зависимостей. Для частичного решения этой проблемы mock и mock-urpm позволяют кэшировать chroot-окружения и использовать для сборки пакетов заранее подготовленные копии chroot. Недостатком такого подхода является шанс допустить неполноту сборочных зависимостей, а также возможность проявления ошибок из-за отсутствия в сохраненных копиях обновлений, вышедших после их создания. Однако выигрыш по производительности от кэширования настолько велик, что вероятностью возникновения таких ошибок обычно пренебрегают. Соответствующие проверки оставляют на финальную стадию подготовки пакета, которая происходит уже на машине разработчика.

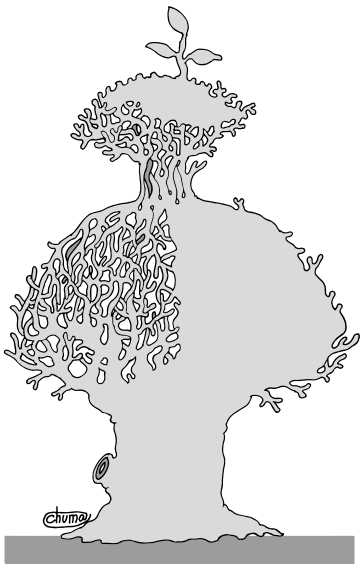
Сборочные фермы

В большинстве дистрибутивов все пакеты, попадающие в репозитории, собираются с помощью специальных сборочных ферм. Ферма представляет собой пул серверов, на которых осуществляется изолированная сборка пакетов – либо в отдельных виртуальных машинах, либо в chroot-окружениях. Использование централизованной сборочной среды обеспечивает:

- единое окружение для всех пакетов;
- сборку для всех аппаратных архитектур (ведь дистрибутивы обычно поддерживают несколько аппаратных платформ, и не всегда у разработчиков есть возможность собрать пакеты для каждой из них);
- автоматизированные проверки собранных пакетов на предмет следования правилам дистрибутива. Конечно, подобные проверки мэйнтейнер может производить и локально, но сборочная ферма позволяет запретить выпуск пакетов, имеющих серьезные нарушения;
- автоматическое помещение пакета в репозиторий.

В РОСЕ в роли сборочной фермы используется среда собственного производства – Automatic Build Farm, ABF – http://abf.rosalinux.ru. Стоит отметить, что ABF – это не просто система для сборки пакетов, но полноценная инфраструктура для ведения разработки приложений. В частности, ABF предоставляет программистам Git-репозиторий для хранения исходного кода, легковесный трекер ошибок и запросов, проектную вики и ряд других полезных инструментов. Разработка многих собственных программных компонентов РОСЫ ведется именно в ABF. Непосредственно сборочная инфраструктура ABF включает в себя следующие компоненты:

- подсистема сборки – компонент, отвечающий непосредственно за сборку пакетов из исходных кодов;
- CKB – система контроля версий (в настоящее время используется Git) для хранения патчей и spec-файлов;
- файловое хранилище для хранения архивов с исходным кодом приложений;
- подсистема сборки установочных iso-образов дистрибутивов;
- веб-интерфейс, представляющий доступ ко всем основным функциям системы;
- консольный клиент, который может быть использован для удаленного доступа к ABF из различных скриптов.



Разработка дистрибутива Linux — сложное и интересное занятие, решающее как организационные, так и технические вопросы

Подсистема сборки реализована с использованием клиент-серверной технологии. Центром подсистемы является ядро (Build core), принимающее задачи на сборку, распределяющее их по сборочным клиентам, отслеживающее статус каждой сборки и перемещающее собранные пакеты в репозитории. Сборочные клиенты (Build clients) соответственно занимаются непосредственной сборкой пакетов. В качестве входного параметра клиенты получают имя пакета и имя ветки в Git, которую необходимо использовать для сборки. Все остальные задачи (извлечение необходимой информации из Git, создание сборочного окружения и так далее) выполняются уже клиентом.

Исходный код ПО, входящего в пакеты, обычно хранится не в системе контроля версий, а размещается в архивированном виде на файловом сервере. Так происходит потому, что исходный код для большинства пакетов берется в виде архивов от соответствующих разработчиков. Помещение этих бинарных архивов в СКВ не имеет большого смысла, но приводит к дополнительной нагрузке, ведь при клонировании Git-репозитория пользователь вместе со всей историей получит все версии архивов.

В принципе ABF позволяет получать файлы по сети с сайта разработчика непосредственно в начале сборки, однако такой подход не является надежным, ведь нет гарантий, что сторонний сервер будет доступен в нужный момент времени или файлы на этом сервере не будут удалены или перемещены. Поэтому архивы с исходным кодом копируются в файловое хранилище ABF, а в СКВ соответствующих пакетов помещается специальный файл с указанием, какие архивы необходимы для их сборки.

Контроль качества репозиториев

При работе большого количества людей над общей массой пакетов важным аспектом является обеспечение согласованности результатов этой работы. Ведь пакеты в дистрибутиве сильно взаимосвязаны, и изменение одного из них может затронуть десятки других, и не всегда мэйнтейнер в состоянии оценить все последствия манипуляций с подконтрольными ему приложениями. В помощь ему предоставляются различные автоматические анализаторы репозиториев, производящие постоянный мониторинг пакетной базы. Рассмотрим основные проблемы, на борьбу с которыми нацелен такой мониторинг (полный список отчетов можно посмотреть на сайте http://fba.rosalinux.ru).

Замкнутость репозиториев по зависимостям

Чтобы некоторый пакет можно было успешно установить, в системе должны присутствовать все его зависимости. Зависимые пакеты могут поддерживаться разными мэйнтейнерами, несогласованность действий которых может привести к исчезновению или изменению зависимостей и невозможности установить пакет. Например, если пакет A требует наличия пакета B версии 1, то обновление B до версии 2 приведет к невозможности установить A. В этом случае надо обновлять зависимости пакета A (а, возможно, и сам пакет A, ведь старая версия может и не заработать с новой версией B).

Следить за зависимостями пакета – одна из основных задач мэйнтейнера. Часть зависимостей пакета выставляется автоматически инструментами rpmbuild на основе анализа файлов пакета (например, в зависимости пакета помещаются имена библиотек, используемых в его бинарных файлах, имена модулей Perl, Python и других языков, используемых в его скриптах, и так далее). Однако полностью избавиться от необходимости прописывать зависимости вручную пока не представляется возможным, и при большом количестве пакетов неизбежно возникают ошибки, связанные с человеческим фактором. Да и автоматические средства иногда допускают огрехи, и за ними также нужно «приглядывать». Репозитории РОСЫ подвергаются постоянному мониторингу на предмет нарушения зависимостей, и полнота зависимостей является одним из главных критериев при оценке готовности очередной версии к выпуску.

Более продвинутым вариантом анализа является проверка корректности зависимостей пакета, то есть попытка определить, действительно ли список зависимостей полон и ПО, входящее в пакет, будет корректно функционировать после их установки. Конечно, полностью автоматизировать такие проверки практически невозможно и без ручного

тестирования не обойтись, однако некоторые аспекты могут быть возложены и на компьютер. В частности, можно проверить, что устанавливаемые по зависимостям пакеты предоставляют все необходимые приложению библиотеки и функции – такая проверка производится на основе анализа бинарных файлов приложений и библиотек.

Возможность пересборки пакета

С выходом релиза работа над пакетной базой выпущенной версии не прекращается – как минимум в течение всего срока поддержки для нее предоставляются обновления, устраняющие различные ошибки в ПО. Обновление ПО подразумевает сборку обновленной версии соответствующего пакета. Для поддержки обновлений необходимо обеспечить возможность пересборки любого пакета на текущей пакетной базе. Эта задача включает в себя отслеживание зависимостей, но не сводится к нему. Например, нередко при небольшом обновлении библиотеки перестают собираться зависящие от нее приложения, поскольку в новой версии перенесли часть деклараций из одного заголовочного файла в другой, изменили имена каких-либо переменных или внесли другие модификации, нарушающие обратную совместимость с предыдущей версией на уровне исходного кода.

Будет печально, если при подготовке обновления пакета мэйнтейнер обнаружит, что пакет больше не собирается, ведь обновление может быть срочным (например, связанным с устранением бреши в безопасности), и тратить время на починку сборки – непозволительная роскошь, подвергающая излишнему риску пользователей системы. Поэтому при обновлении любого пакета важно убедиться, что все зависящие от него пакеты могут быть пересобраны. В идеале следует выполнить эту проверку рекурсивно, то есть проверить, что собираются пакеты, зависящие от пересобранных, и т.д.

ABF позволяет запустить такую пересборку всех зависимых пакетов. Однако в реальности процесс пересборки может оказаться слишком ресурсоемким (в частности, обновление системных библиотек может повлечь пересборку всего репозитория). В ситуациях, когда ПО обновляется очень часто (например, при работе над новой версией системы), используется другой подход. При обновлении какого-либо пакета мэйнтейнеры не трогают зависящие от него (за исключением случаев, когда необходимость обновить зависимый пакет очевидна), вместо этого проводятся регулярные массовые пересборки всех пакетов репозитория и последующая починка всех несобравшихся пакетов.

Возможность пересборки любого пакета – обязательное условие выпуска релиза в РОСЕ. В завершающей стадии подготовки релиза итерации массовых пересборок и последующих исправлений проводятся до тех пор, пока не будет достигнута полная пересобираемость всей массы пакетов.

Анализ альтернатив

Нередко случается, что одна и та же зависимость может быть удовлетворена несколькими различными пакетами. Иногда такие альтернативы легитимны (например, если приложению нужна Java, то она может быть предоставлена посредством OpenJDK либо Oracle Java), однако в ряде случаев могут являться следствием некорректной работы автоматических скриптов, определяющих зависимости пакета. При установке пакетов, зависимости которых могут быть разрешены несколькими способами, система спрашивает пользователя, какой именно вариант он предпочитает. Появление «неожиданных» альтернатив смущает и раздражает, а их установка может привести к неработоспособности пакета (хотя все зависимости вроде как удовлетворены). Возможен и автоматический выбор системой произвольного пакета для удовлетворения той или иной зависимости, и при неудачном выборе установленная у пользователя программа опять же окажется неработоспособной.

Таким образом, отслеживание зависимостей, предоставляемых сразу несколькими пакетами, и своевременное удаление некорректных альтернатив – важный аспект мониторинга репозиториев дистрибутива.

Поиск устаревших пакетов

Эволюция ПО не всегда сводится к обновлению версий. Бывает и так, что одни проекты постепенно умирают, а на смену им приходят другие, со схожей функциональностью. Нередко оказывается, что в репозиториях системы одновременно оказываются и устаревшие пакеты, и их более современные аналоги. Зачастую это вынужденная мера, так как от устаревшего пакета могут зависеть другие, и до их обновления удалить устаревший пакет из репозитория нельзя.

Имея несколько тысяч пакетов, отслеживать текущую ситуацию с зависимостями и удалять ставшие ненужными пакеты непросто. Однако подобный анализ легко автоматизируется благодаря возможности указывать непосредственно в пакете, какие другие пакеты при его установке должны быть объявлены устаревшими. На основе подобной информации скрипты, осуществляющие мониторинг, способны автоматически вычислять ставшие ненужными компоненты.

Обеспечение актуальности версий ПО

Одной из задач мэйнтейнера является отслеживание выхода новых версий приложений и своевременная их сборка для дистрибутива. При небольшом количестве пакетов эта задача не отнимает много времени, особенно если мэйнтейнер сам является активным пользователем соответствующих программ. Однако в репозиториях РОСЫ содержится более десяти тысяч пакетов. Один человек может эффективно отслеживать обновления десятка приложений, но уже с сотней программ будут некоторые затруднения (особенно если заниматься этим на досуге, в свободное от основной работы время). И если использовать только ручной труд мэйнтейнеров, то для поддержки пакетной базы дистрибутива понадобится несколько десятков, а лучше сотен человек.

Конечно, далеко не все приложения обновляются часто, но и мэйнтейнеры-добровольцы имеют тенденцию то появляться, то исчезать, а сопровождать дистрибутив и выпускать обновления и новые релизы надо постоянно. Поэтому для поддержки пакетной базы дистрибутива используются различные средства автоматизации, позволяющие повысить эффективность работы мэйнтейнеров, избавив их от многих рутинных задач, а заодно несколько нивелировать человеческий фактор.

В частности, для отслеживания выхода новых версий ПО можно использовать соответствующие автоматизированные системы, осуществляющие мониторинг сайтов, на которые выкладывается исходный код того или иного приложения. При обнаружении новой версии такие систе-

мы могут уведомлять об этом пользователя посредством электронной почты или веб-интерфейса.

В РОСЕ в качестве такой системы используется Upstream Tracker (http://upstream-tracker.org). Помимо отслеживания выхода новых версий ПО, этот инструментарий позволяет следить за обновлениями пакетов в других дистрибутивах. Ведь разработчики этих операционных систем способны добавлять свои собственные патчи, которые могут оказаться полезными и другим дистрибутивам.

К достоинствам Upstream Tracker можно отнести то, что он не просто уведомляет о выходе новой версии ПО, но предоставляет визуализированный отчет об изменениях, произошедших в новой версии по сравнению с предыдущей. Для библиотек Upstream Tracker производит анализ обратной совместимости их публичных интерфейсов. В случае если обратная совместимость сохраняется, библиотеку можно обновлять безболезненно, в противном случае может потребоваться обновление программ, использующих эту библиотеку.

Также мэйнтейнерам РОСЫ доступен инструмент updates_builder, связывающий Upstream Tracker и ABF. Он отслеживает появление новых версий приложений на сайтах разработчиков и при их обнаружении пытается пересобрать соответствующий пакет РОСЫ с использованием новых архивов (модифицировав соответствующим образом spec-файл). В случае успеха собранные пакеты помещаются в специальные репозитории.

Мэйнтейнеру отправляется письмо с результатами сборки. Если новая версия собралась успешно, то остается проверить ее работоспособность и отправить в официальные репозитории. Такой подход позволяет существенно сократить затраты на отслеживание и сборку минорных обновлений, ведь многие проекты предпочитают выпускать новые версии, даже если накопившиеся изменения незначительны.

***

Процесс разработки дистрибутива Linux – сложное и интересное занятие, требующее решения как организационных, так и технических вопросов. В этом процессе найдется место как опытным разработчикам, так и начинающим программистам, а также нетехническим специалистам, например, дизайнерам и переводчикам.

В наше время многие выпускники вузов при поиске работы сталкиваются с проблемой отсутствия опыта, для многих студентов участие в разработке Linux и других свободных продуктов может послужить отличным решением этой проблемы. Ведь для участия в открытых проектах достаточно только желания, в то же время результаты работы являются общедоступными и могут быть продемонстрированы потенциальным работодателям. Вклад в реальный проект скажет о вас гораздо больше, чем список прослушанных курсов и выполненных учебных заданий. Добавьте к этому моральное удовлетворение от проделанной работы и от процесса совместной работы в большой международной команде и, возможно, вы приблизитесь к ответу на вопрос, почему так много людей тратят свободное время на разработку свободного ПО.

В следующей, заключительной, части статьи мы рассмотрим, как организован процесс контроля за качеством ПО в РОСЕ. EOF

1. Силаков Д. Что такое дистрибутив Linux? Разработка дистрибутива Linux на примере РОСЫ. //«Системный администратор», №1-2, 2013 г. – С. 120-124.