ООО «БАЗАЛЬТ СПО»

АЛЬТ ПЛАТФОРМА

Руководство пользователя

Ред. 1.0

МОСКВА 2023

Содержание

Данное руководство предназначено для пользователей «Альт Платформа» и содержит информацию о технологии сборки пакетов как в целом, для Linux, так и специфическую для дистрибутивов «Альт».

Для команд, встречающихся в тексте, используется следующая нотация:

- команды без административных привилегий начинаются с символа «$»;
- команды с административными привилегиями начинаются с символа «#».

Примечание. По умолчанию `sudo` отключено. Для получения административных привилегий используется команда `su -`. Для включения `sudo` в стандартном режиме можно использовать команду:

```
# control sudowheel enabled
```

1 РАБОТА С ПАКЕТАМИ

Для работы с пакетами в «Альт Платформа» используются следующие инструменты:

- пакетный менеджер rpm;
- система управления пакетами apt.

1.1 Пакетный менеджер RPM

Все пакеты в «Альт Платформа» собираются в формате RPM.

RPM (RPM Package Manager) – это семейство пакетных менеджеров, применяемых в различных дистрибутивах GNU/Linux, в том числе и в проекте Sisyphus (Сизиф) и в дистрибутивах «Альт». Практически каждый крупный проект, использующий RPM, имеет свою версию пакетного менеджера, отличающуюся от остальных.

Между представителями семейства RPM могут иметься следующие различия:

- наборы макросов, используемых в spec-файлах;
- различное поведение RPM при сборке «по умолчанию» – при отсутствии каких-либо указаний в spec-файлах;
- формат строк зависимостей;
- мелкие отличия в семантике операций (например, в операциях сравнения версий пакетов);
- мелкие отличия 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:

- обширный набор макросов для сборки различных типов пакетов;
- отличающееся поведение «по умолчанию» для уменьшения количества шаблонного кода в spec-файлах;
- наличие механизмов для автоматического поиска межпакетных зависимостей;
- наличие так называемых `set-version` зависимостей (начиная с 4.0.4-alt98.46), обеспечивающих дополнительный контроль над 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 осуществлен частичный переход на rpm 4.13).

1.2 Утилита командной строки RPM

RPM – это низкоуровневая утилита командной строки, используемая для установки, удаления, обновления, выполнения запросов и проверки целостности пакетов программного

обеспечения. Все остальные утилиты управления пакетами в конечном итоге работают через RPM. RPM ничего не знает о репозиториях и оперирует только файлами, пакетами и их зависимостями, для чего использует собственную базу данных (БД). Информация, хранимая в этой БД, в идеале должна соответствовать фактической картине внутри файловой системы. В дистрибутивах «Альт» RPM хранит свою БД в /var/lib/rpm. В один момент времени в системе должен быть запущен лишь один процесс, обращающийся к БД RPM, другие запросы блокируются.

Пакеты могут предоставлять (провайдить) что-либо, требовать (запрашивать) что-либо и конфликтовать с чем-либо, образуя, таким образом, систему межпакетных зависимостей (dependencies). В дистрибутивах «Альт» можно установить пакет, если удовлетворены все его зависимости и нет конфликтов с другими уже установленными пакетами и объектами файловой системы. Из этого следует, что никакими системными файлами нельзя манипулировать непосредственно (вручную), никакое программное обеспечение не стоит устанавливать в обход штатного пакетного менеджера.

Пакеты всегда содержат определенную мета-информацию, на которую ориентируется утилита rpm. Пакеты также могут содержать какие-то файлы, каталоги и символические ссылки. Так называемые мета-пакеты никогда не содержат объектов файловой системы, в них перечисляются только зависимости на другие пакеты.

Основные режимы работы утилиты rpm:

- Install: установка пакетов;
- Remove: удаление пакетов;
- Upgrade: обновление пакетов;
- Query: выполнение запросов;
- Verify: проверка целостности пакетов.

Примечание. Справку по ключам команды rpm можно получить, выполнив команду `rpm --help`

В качестве примера в данной главе используется пакет Yodl-docs (файл yodl-docs-4.03.00-alt2.noarch.rpm).

1.2.1 Вывод информации о пакете

Для вывода информации о пакете, который еще не установлен в систему, используется ключ `-qip (Query|Install|Package)`:

```
# rpm -i package.rpm
```

где `package.rpm` – файл пакета.

Например:

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

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

This package contais documentation for Yodl.
```

Примечание. Ключ `-p (Package)` работает не с базой RPM-пакетов, а с конкретным пакетом.

Для вывода информации об установленном в систему пакете используется команда:

```
$ rpm -qi package
```

где `package` – установленный пакет.

```
$ rpm -qi bash
Name        : bash
Version     : 4.4.23
Release     : alt1
DistTag     : sisyphus+221902.500.4.1
Architecture: noarch
Install Date: Ср 29 ноя 2023 11:03:45
Group       : Shells
Size        : 0
License     : None
Signature       : DSA/SHA1, Вт 19 фев 2019 16:40:44, Key ID
95c584d5ae4ae412
Source RPM  : bash-defaults-4.4.23-alt1.src.rpm
Build Date  : Вт 19 фев 2019 16:40:42
Build Host  : ldv-sisyphus.hasher.altlinux.org
Relocations : (not relocatable)
Packager    : Dmitry V. Levin <ldv@altlinux.org>
Vendor      : ALT Linux Team
Summary     : The GNU Bourne Again SHell (/bin/bash)
Description :
This package provides default setup for the GNU Bourne Again SHell
(/bin/bash).
```

1.2.2 Установка пакета из файла

П р и м е ч а н и е . В команде должен быть указан файл пакета или полный путь к нему.

Для установки RPM-пакета используется ключ `-i` (`Install`):

```
# rpm -i package.rpm
```

где `package.rpm` – файл пакета.

Пример выполнения команды:

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

В команде можно указать дополнительные ключи `-vh` (`Verbose|Hash`). Ключи `-v` и `-h` не влияют на установку, а служат для вывода наглядного процесса сборки в консоль. Ключ `-v` выводит детальные значения. Ключ `-h` выводит символ «#» по мере установки пакета.

Пример выполнения команды:

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

1.2.3 Обновление пакета

Для обновления пакета используется ключ `-U` (если пакет не установлен, то будет установлен):

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

1.2.4 Просмотр файлов пакета

Чтобы получить список файлов, находящихся в пакете, который не установлен в систему, используются ключи `-qpl` (`Query|Package|List`):

```
$ rpm -qpl package.rpm
```

Например:

```
$ rpm -qpl yodl-docs-4.03.00-alt2.noarch.rpm
/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
```

Для просмотра файлов пакета, установленного в систему, используется команда:

```
$ rpm -ql package
```

Например:

```
$ rpm -qpl yodl-docs
```

1.2.5 Список недавно установленных пакетов

Для получения списка пакетов, которые были недавно установлены в систему, используется команда:

```
$ rpm -qa -last
```

Например:

```
$ rpm -qa --last
usbids-20231116-alt1.noarch                   Ср 29 ноя 2023 20:45:28
pciids-20231116-alt1.noarch                   Ср 29 ноя 2023 20:45:28
ethtool-6.5-alt3.x86_64                       Ср 29 ноя 2023 20:45:28
tree-2.0.2-alt1.x86_64                        Ср 29 ноя 2023 11:04:32
shim-signed-15.4-alt2.x86_64                  Ср 29 ноя 2023 11:04:32
pv-1.6.6-alt1.x86_64                          Ср 29 ноя 2023 11:04:32
mailx-8.1.2-alt8.x86_64                       Ср 29 ноя 2023 11:04:32
lsof-4.93.2-alt1.x86_64                       Ср 29 ноя 2023 11:04:32
…
```

Чтобы список прокручивался можно указать параметр `less`:

```
# rpm -qa -last | less
```

1.2.6 Поиск пакета в системе

Проверить установлен ли пакет в систему можно, выполнив команду:

```
$ rpm -q пакет
```

Например:

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

Вывод:

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

или:

```
пакет yodl-docs не установлен
```

Вывести список всех установленных пакетов можно, выполнив команду:

```
$ rpm -qa
```

Определить, установлен ли пакет в системе или нет, также можно, выполнив следующую команду:

```
$ rpm -qa | grep yodl-docs
yodl-docs-4.03.00-alt2.noarch
```

1.2.7 Узнать пакет по файлу

Узнать к какому пакету относится файл можно, выполнив команду:

```
$ rpm –qf /путь/к/файлу
```

Например:

```
$ rpm -qf /usr/share/doc/yodl-doc
yodl-docs-4.03.00-alt2.noarch
```

1.2.8 Зависимости пакетов

Чтобы узнать от каких пакетов зависит указанный пакет, используется конструкция:

```
$ rpm -q --requires пакет
```

Пример:

```
$ rpm -q --requires yodl-doc
rpmlib(PayloadIsLzma)
```

Узнать, какие установленные пакеты зависят от указанного можно, выполнив команду:

```
$ rpm -q --whatrequires пакет
```

Пример:

```
# rpm -q --whatrequires yodl-docs
ни один из пакетов не требует yodl-docs eepm-3.28.1-alt1.noarch
```

Узнать, какие зависимости предоставляет указанный пакет можно, выполнив команду:

```
# rpm -q --whatprovides пакет
```

Пример:

```
# rpm -q --whatprovides yodl-docs
yodl-docs-4.03.00-alt2.noarch
```

1.3 Система управления пакетами APT

Утилита RPM делает атомарными (одношаговыми) операции с отдельными пакетами: вместо копирования множества файлов и запуска нескольких сценариев пользователь вводит одну

команду «установить/удалить пакет». Однако атомарная с точки зрения пользователя операция – добавление в систему одного нового компонента может состоять из нескольких (и даже многих) операций над пакетами. Чтобы сделать процедуру установки, удаления и обновления компонента системы атомарной, были разработаны системы управления пакетами.

Используемая в «Альт» усовершенствованная система управления программными пакетами APT, является системой управления пакетами с простым пользовательским интерфейсом, позволяющим производить установку, обновление и повседневные «хозяйственные» работы с установленными на машине программами без необходимости изучения тонкостей используемого в дистрибутиве менеджера программных пакетов.

В распоряжении APT находятся две базы данных: одна описывает установленные в системе пакеты, вторая – внешний репозиторий. APT отслеживает целостность установленной системы и, в случае обнаружения противоречий в зависимостях пакетов, руководствуется сведениями о внешнем репозитории для разрешения конфликтов и поиска корректного пути их устранения.

Система APT состоит из нескольких утилит. Чаще всего используется утилита управления пакетами apt-get: она автоматически определяет зависимости между пакетами и строго следит за их соблюдением при выполнении любой из следующих операций: установка, удаление или обновление пакетов.

APT хранит кэш скачанных из сети пакетов в `/var/cache/apt/archives`. Кэш скачанных индексов можно найти в `/var/lib/apt/lists`. Вся конфигурация APT хранится в `/etc/apt`, главный конфигурационный файл: `/etc/apt/apt.conf`, списки подключенных репозиториев хранятся в `/etc/apt/sources.list` и `/etc/apt/sources.list.d/*`. Посмотреть текущую конфигурацию APT можно командой:

```
$ apt-config dump
```

1.3.1 Репозитории

Репозитории, с которыми работает APT, отличаются от обычного набора пакетов наличием метаинформации – индексов пакетов, содержащихся в репозитории, и сведений о них. Поэтому, чтобы получить всю информацию о репозитории, APT достаточно получить его индексы.

APT может работать с любым количеством репозиториев одновременно, формируя единую информационную базу обо всех содержащихся в них пакетах. При установке пакетов APT обращает внимание только на название пакета, его версию и зависимости, а расположение в том или ином репозитории не имеет значения. Если потребуется, APT в рамках одной операции установки группы пакетов может пользоваться несколькими репозиториями.

Примечание. Подключая одновременно несколько репозиториев, нужно следить за тем, чтобы они были совместимы друг с другом по пакетной базе, т. е. отражали один определенный

этап разработки. Например, совместимыми являются основной репозиторий дистрибутива и репозиторий обновлений по безопасности к данному дистрибутиву. В то же время смешение среди источников APT репозиториев, относящихся к разным дистрибутивам, или смешение стабильного репозитория с нестабильной веткой разработки (Sisyphus) чревато различными неожиданными трудностями при обновлении пакетов.

APT позволяет взаимодействовать с репозиторием с помощью различных протоколов доступа. Наиболее популярные – HTTP и FTP, однако существуют и некоторые дополнительные методы.

Для того чтобы APT мог использовать тот или иной репозиторий, информацию о нем необходимо поместить в файл `/etc/apt/sources.list`, либо в любой файл `.list` (например, `mysources.list`) в каталоге `/etc/apt/sources.list.d`. Описания репозиториев заносятся в эти файлы в следующем виде:

```
rpm [подпись] метод: путь база название
rpm-src [подпись] метод: путь база название
```

Здесь:

- rpm или rpm-src – тип репозитория (скомпилированные программы или исходные тексты);
- [подпись] – необязательная строка-указатель на электронную подпись разработчиков. Наличие этого поля подразумевает, что каждый пакет из данного репозитория должен быть подписан соответствующей электронной подписью. Подписи описываются в файле /etc/apt/vendor.list;
- метод – способ доступа к репозиторию: ftp, http, file, cdrom, copy;
- путь – путь к репозиторию в терминах выбранного метода;
- база – относительный путь к базе данных репозитория;
- название – название репозитория.

Непосредственно после установки ОС «Альт» в файлах `/etc/apt/sources.list.d/*.list` обычно указывается интернет-репозиторий, совместимый с установленным дистрибутивом.

После редактирования списка репозиториев в `sources.list`, необходимо обновлять локальную базу данных APT о доступных пакетах. Это делается командой `apt-get update`.

Если в `sources.list` присутствует репозиторий, содержимое которого может изменяться (как происходит с любым постоянно разрабатываемым репозиторием, в частности, обновлений по безопасности), то прежде чем работать с APT, необходимо синхронизировать локальную базу данных с удаленным сервером командой `apt-get update`. Локальная база данных создается заново каждый раз, когда в репозитории происходит изменение: добавление,

удаление или переименование пакета. Для репозиториев, находящихся на компакт-дисках и подключенных командой `apt-cdrom add`, синхронизация производится единожды в момент подключения.

При выборе пакетов для установки, APT руководствуется всеми доступными репозиториями вне зависимости от способа доступа к ним. Так, если в репозитории, доступном по сети Интернет, обнаружена более новая версия программы, чем на компакт-диске, то APT начнет загружать данный пакет из Интернет. Поэтому если подключение к Интернет отсутствует или ограничено низкой пропускной способностью канала или высокой стоимостью, то следует закомментировать те строчки в `/etc/apt/sources.list`, в которых говорится о ресурсах, доступных по Интернет.

1.3.1.1 Утилита apt-repo для работы с репозиториями

Для редактирования репозиториев можно воспользоваться скриптом `apt-repo`:

- просмотреть список активных репозиториев:

```
$ apt-repo
```

- показать все доступные репозитории (неактивные будут закомментированы «#»):

```
$ apt-repo -a
```

- добавить репозиторий в список активных репозиториев:

```
# apt-repo add репозиторий
```

- удалить или выключить репозиторий:

```
# apt-repo rm репозиторий
```

- удалить все существующие источники и добавить новый репозиторий:

```
# apt-repo set репозиторий
```

- удалить все источники типа cdrom и все хранилища задач (task):

```
# apt-repo clean
```

- обновить информацию о репозиториях (запустить команду `apt-get update`):

```
# apt-repo update
```

- справка о команде `apt-repo`:

```
$ man apt-repo
```

или

```
$ apt-repo --help
```

П р и м е ч а н и е . Для выполнения большинства команд необходимы права администратора.

Типичный пример использования: удалить все источники и добавить стандартный репозиторий P10 (архитектура выбирается автоматически):

```
# apt-repo rm all
# apt-repo add p10
```

Или то же самое одной командой:

```
# apt-repo set p10
```

Источник может быть указан в формате sources.list(5):

```
# apt-repo add "rpm http://git.altlinux.org/repo/39115/ i586 task"
```

Допустимы следующие типы репозиториев: rpm, rpm-dir и rpm-src. APT поддерживает протоколы file://, copy://, http://, ftp://, rsync:// и cdrom://. URL с обязательным указанием протокола может сопровождаться необязательным указанием архитектуры и одним или несколькими компонентами. Если в конце строки отсутствуют архитектура и компонент, будут добавлены две строки (текущая архитектура системы и noarch) с компонентом classic.

Пример добавления локального репозитория:

```
# apt-repo add file:/srv/public/mirror/p10/branch
```

1.3.1.2 Добавление репозитория на CD/DVD-носителе

Для добавления в `sources.list` репозитория на компакт-диске в APT предусмотрена специальная утилита – `apt-cdrom`. Чтобы добавить запись о репозитории на компакт-диске, достаточно вставить диск в привод и выполнить команду `apt-cdrom add`. После этого в `sources.list` появится запись о подключенном диске.

П р и м е ч а н и е . Если при выполнении команды `apt-cdrom add`, получена ошибка:

```
Не найдена точка монтирования /media/ALTLinux/ диска
```

Необходимо:

- в файл `/etc/fstab` добавить строку:

```
/dev/sr0 /media/ALTLinux udf,iso9660
ro,noauto,user,utf8,nofail,comment=x-gvfs-show  0 0
```

- создать каталог для монтирования:

```
# mkdir /media/ALTLinux
```

- затем использовать команду добавления носителя:

```
# apt-cdrom add
```

1.3.1.3 Добавление репозиториев вручную

Для изменения списка репозиториев можно отредактировать в любом текстовом редакторе файлы из каталога `/etc/apt/sources.list.d/`.

П р и м е ч а н и е . Для изменения этих файлов необходимы права администратора.

В файле `alt.list` может содержаться такая информация:

```
# ftp.altlinux.org (ALT Linux, Moscow)

# ALT Platform 10
#rpm [p10] ftp://ftp.altlinux.org/pub/distributions/ALTLinux
p10/branch/x86_64 classic
#rpm [p10] ftp://ftp.altlinux.org/pub/distributions/ALTLinux
p10/branch/x86_64-i586 classic
#rpm [p10] ftp://ftp.altlinux.org/pub/distributions/ALTLinux
p10/branch/noarch classic

rpm [p10] http://ftp.altlinux.org/pub/distributions/ALTLinux
p10/branch/x86_64 classic
rpm [p10] http://ftp.altlinux.org/pub/distributions/ALTLinux
p10/branch/x86_64-i586 classic
rpm [p10] http://ftp.altlinux.org/pub/distributions/ALTLinux
p10/branch/noarch classic

#rpm [p10] rsync://ftp.altlinux.org/ALTLinux p10/branch/x86_64 classic
#rpm [p10] rsync://ftp.altlinux.org/ALTLinux p10/branch/x86_64-i586 classic
#rpm [p10] rsync://ftp.altlinux.org/ALTLinux p10/branch/noarch classic
```

По сути, каждая строчка соответствует некому репозиторию. Не активные репозитории – строки, начинающиеся со знака «#». Для добавления нового репозитория, достаточно дописать его в этот или другой файл.

После обновления списка репозиториев следует обновить информацию о них (выполнить команду `apt-get update` или `apt-repo update`).

1.3.2 Поиск пакетов

Если точное название пакета неизвестно, то для его поиска можно воспользоваться утилитой `apt-cache`, которая позволяет искать не только по имени пакета, но и по его описанию.

Команда `apt-cache search` позволяет найти все пакеты, в именах или описании которых присутствует указанная подстрока. Например:

```
$ apt-cache search ^gimp
gimp - The GNU Image Manipulation Program
libgimp - GIMP libraries
libgimp-devel - GIMP plugin and extension development kit
gimp-help-en - English help files for the GIMP
```

```
gimp-help-ru - Russian help files for the GIMP
gimp-plugin-separateplus - Improved version of the CMYK Separation plug-in [...]
gimp-script-ISONoiseReduction - Gimp script for reducing sensor noise [...]
gimp-plugin-gutenprint - GIMP plug-in for gutenprint
gimp-plugin-ufraw - GIMP plugin for opening and converting RAW files [...]
```

Следует обратить внимание, что в данном примере в поисковом выражении используется символ ^, указывающий на то, что необходимо найти совпадения только в начале строки (в данном случае – в начале имени пакета).

Для того чтобы подробнее узнать о каждом из найденных пакетов и прочитать его описание, можно воспользоваться командой `apt-cache show`, которая покажет информацию о пакете из репозитория:

```
$ apt-cache show gimp-help-ru
Package: gimp-help-ru
Section: Graphics
Installed Size: 37095561
Maintainer: Alexey Tourbin <at@altlinux.ru>
Version: 2.6.1-alt2
Pre-Depends: rpmlib(PayloadIsLzma)
Provides: gimp-help-ru (= 2.6.1-alt2)
Obsoletes: gimp-help-common (< 2.6.1-alt2)
Architecture: noarch
Size: 28561160
MD5Sum: 0802d8f5ec1f78af6a4a19005af4e37d
Filename: gimp-help-ru-2.6.1-alt2.noarch.rpm
Description: Russian help files for the GIMP
Russian help files for the GIMP.
```

Утилита `apt-cache` позволяет осуществлять поиск и по русскому слову, однако в этом случае будут найдены только те пакеты, у которых есть описание на русском языке. К сожалению, русское описание на настоящий момент есть не у всех пакетов, хотя описания наиболее актуальных для пользователя пакетов переведены.

1.3.3 Установка или обновление пакета

Установка пакета с помощью APT выполняется командой:

```
# apt-get install имя_пакета
```

Примечание. Для установки пакетов требуются привилегии администратора.

Утилита `apt-get` позволяет устанавливать в систему пакеты, требующие для работы другие, пока еще не установленные. В этом случае он определяет, какие пакеты необходимо установить, и устанавливает их, пользуясь всеми доступными репозиториями.

Установка пакета gimp командой `apt-get install gimp` приведет к следующему диалогу с APT:

```
# apt-get install gimp
Чтение списков пакетов... Завершено
Построение дерева зависимостей... Завершено
Следующие дополнительные пакеты будут установлены:
icc-profiles libbabl libgegl libgimp libjavascriptcoregtk2 libopenraw
libspiro libwebkitgtk2 libwmf
Следующие НОВЫЕ пакеты будут установлены:
gimp icc-profiles libbabl libgegl libgimp libjavascriptcoregtk2
libopenraw libspiro libweb-kitgtk2 libwmf
0 будет обновлено, 10 новых установлено, 0 пакетов будет удалено и 0
не будет обновлено.
Необходимо получить 0B/24,6MB архивов.
После распаковки потребуется дополнительно 105MB дискового
пространства.
Продолжить? [Y/n] y
. . .
Получено 24,6MB за 0s (44,1MB/s).
Совершаем изменения...
Preparing... ######################################## [100%]
1: libbabl ######################################## [10%]
2: libwmf ######################################## [20%]
3: libjavascriptcoregtk2 ######################################## [30%]
4: libwebkitgtk2 ######################################## [40%]
5: icc-profiles ######################################## [50%]
6: libspiro ######################################## [60%]
```

```
7: libopenraw ######################################## [70%]
8: libgegl ######################################## [80%]
9: libgimp ######################################## [90%]
10: gimp ######################################## [100%]
Running /usr/lib/rpm/posttrans-filetriggers
Завершено.
```

Команда `apt-get install имя_пакета` используется и для обновления уже установленного пакета или группы пакетов. В этом случае `apt-get` дополнительно проверяет, не обновилась ли версия пакета в репозитории по сравнению с установленным в системе.

При помощи APT можно установить и отдельный бинарный rpm-пакет, не входящий ни в один из репозиториев (например, полученный из Интернет). Для этого достаточно выполнить команду `apt-get install путь_к_файлу.rpm`. При этом APT проведет стандартную процедуру проверки зависимостей и конфликтов с уже установленными пакетами.

Иногда, в результате операций с пакетами без использования APT, целостность системы нарушается, и `apt-get` отказывается выполнять операции установки, удаления или обновления. В этом случае необходимо повторить операцию, задав опцию `-f`, заставляющую `apt-get` исправить нарушенные зависимости, удалить или заменить конфликтующие пакеты. В этом случае необходимо внимательно следить за сообщениями, выдаваемыми `apt-get`. Любые действия в этом режиме обязательно требуют подтверждения со стороны пользователя.

1.3.4 Удаление установленного пакета

Для удаления пакета используется команда `apt-get remove имя_пакета`. Для того чтобы не нарушать целостность системы, будут удалены и все пакеты, зависящие от удаляемого: если отсутствует необходимый для работы приложения компонент (например, библиотека), то само приложение становится бесполезным. В случае удаления пакета, который относится к базовым компонентам системы, `apt-get` потребует дополнительного подтверждения производимой операции с целью предотвратить возможную случайную ошибку.

Примечание. Для удаления пакетов требуются привилегии администратора.

При попытке с помощью `apt-get` удалить базовый компонент системы, вы увидите следующий запрос на подтверждение операции:

```
# apt-get remove filesystem
Обработка файловых зависимостей... Завершено
Чтение списков пакетов... Завершено
Построение дерева зависимостей... Завершено
Следующие пакеты будут УДАЛЕНЫ:
```

```
basesystem filesystem ppp sudo
Внимание: следующие базовые пакеты будут удалены:
В обычных условиях этого не должно было произойти, надеемся, вы точно
представляете, чего требуете!
basesystem filesystem (по причине basesystem)
0 пакетов будет обновлено, 0 будет добавлено новых, 4 будет
удалено(заменено) и 0 не будет обновлено.
Необходимо получить 0B архивов. После распаковки 588kБ будет
освобождено.
Вы делаете нечто потенциально опасное!
Введите фразу 'Yes, do as I say!' чтобы продолжить.
```

Каждую ситуацию, в которой APT выдает такое сообщение, необходимо рассматривать отдельно. Однако, вероятность того, что после выполнения этой команды система окажется неработоспособной, очень велика.

1.3.5 Обновление всех установленных пакетов

Для обновления всех установленных пакетов используется команда `apt-get upgrade`. Она позволяет обновить те, и только те установленные пакеты, для которых в репозиториях, перечисленных в `/etc/apt/sources.list`, имеются новые версии; при этом из системы не будут удалены никакие другие пакеты. Этот способ полезен при работе со стабильными пакетами приложений, относительно которых известно, что они при смене версии изменяются несущественно.

Иногда, однако, происходит изменение в именовании пакетов или изменение их зависимостей. Такие ситуации не обрабатываются командой `apt-get upgrade`, в результате чего происходит нарушение целостности системы: появляются неудовлетворенные зависимости. Например, переименование пакета MySQL-shared, содержащего динамически загружаемые библиотеки для работы с системой управления базами данных MySQL, в libmysqlclient (отражающая общую тенденцию к наименованию библиотек в дистрибутиве) не приводит к тому, что установка обновленной версии libmysqlclient требует удаления старой версии MySQL-shared. Для разрешения этой проблемы существует режим обновления в масштабе дистрибутива – `apt-get dist-upgrade`.

Для обновления всех установленных пакетов необходимо выполнить команды:

```
# apt-get update
# apt-get dist-upgrade
```

Первая команда (`apt-get update`) обновит индексы пакетов. Вторая команда (`apt-get dist-upgrade`) позволяет обновить только те установленные пакеты, для которых в репозиториях, перечисленных в `/etc/apt/sources.list`, имеются новые версии.

В случае обновления всего дистрибутива APT проведет сравнение системы с репозиторием и удалит устаревшие пакеты, установит новые версии присутствующих в системе пакетов, а также отследит ситуации с переименованиями пакетов или изменения зависимостей между старыми и новыми версиями программ. Все, что потребуется поставить (или удалить) дополнительно к уже имеющемуся в системе, будет указано в отчете `apt-get`, которым APT предварит само обновление.

Примечание. Команда `apt-get dist-upgrade` обновит систему, но ядро ОС не будет обновлено.

1.3.6 Обновление ядра

APT в дистрибутивах «Альт» автоматом не обновляет ядра вместе с обновлением системы (см. настройки hold в apt.conf), поскольку обновление такого критичного компонента системы может привести к нежелательным последствиям. Вместо этого в систему могут быть поставлены пакеты нескольких ядер и модулей к разным ядрам одновременно.

Для обновления ядра ОС необходимо выполнить команду:

```
# update-kernel
```

Примечание. Если индексы сегодня еще не обновлялись перед выполнением команды `update-kernel` необходимо выполнить команду `apt-get update`.

Команда `update-kernel` обновляет и модули ядра, если в репозитории обновилось что-то из модулей без обновления ядра.

Новое ядро загрузится только после перезагрузки системы.

Если с новым ядром что-то пойдет не так, вы сможете вернуться к предыдущему варианту, выбрав его в начальном меню загрузчика.

После успешной загрузки на обновленном ядре можно удалить старое, выполнив команду:

```
# remove-old-kernels
```

2 ОСНОВЫ СБОРКИ RPM-ПАКЕТОВ

2.1 RPM-пакет

RPM-пакет состоит из архива cpio, содержащего файлы (скомпилированные исполняемые файлы, библиотеки, данные), и заголовка, содержащего метаданные о пакете (название, версия, группа и т.п.). Менеджер пакетов RPM использует эти метаданные для определения зависимостей, места установки файлов и другой информации.

Различают два вида пакетов RPM:

- пакет с исходным кодом – SRPM-пакет (имеет расширение .src.rpm). Такой пакет содержит архив (один или несколько) с исходным кодом, spec-файл и, возможно, разнообразные патчи и дополнения. Пакет src.rpm можно использовать только для сборки двоичных пакетов, но не установки. Сборка осуществляется командой:

```
rpmbuild --rebuild package...src.rpm
```

- собранный двоичный пакет – RPM-пакет (имеет расширение вида `<архитектура>.rpm`).

Такие пакеты можно устанавливать командой:

```
rpm -Uvh package...rpm
```

Однако для сборки через rpmbuild возникают очевидные сложности:

- необходимо вручную удовлетворить сборочные зависимости для сборки (поставить компилятор, включаемые файлы, библиотеки). При большом количестве собираемых пакетов система засоряется;
- для сборки пакета нужно сформировать .src.rpm из файлов, разбросанных по разным каталогам (по умолчанию, это подкаталоги SOURCE, SPECS и подкаталоги для сборки в ~/RPM);
- файлы с исходным кодом должны лежать в упакованном виде, что делает трудоемким процесс изготовления патчей;
- на рабочей системе можно упустить необходимое и достаточное количество зависимостей.

Чтобы избежать этих сложностей, в «Альт Платформа» используется две технологии:

- Hasher – для сборки в изолированном окружении. В chroot ставится базовый комплект пакетов и пакеты, необходимые для сборки (поле BuildRequires в спеке). Если какой-то пакет для сборки не указан в спеке, то появится ошибка. Так обеспечивается чистота сборки. Обратной стороной является необходимость иметь доступ к репозиторию, так как пакеты ставятся при каждой сборке в Hasher;
- Gear – для сборки пакетов из репозитория Git. В этом случае все файлы лежат в распакованном виде и в src.rpm упаковываются по правилам, определённым в .gear/rules.

Это позволяет работать сразу с содержимым, быстро делать патчи, вести историю изменений и обмениваться изменениями при коллективной разработке.

2.2 SPEC-файл

Spec-файл можно рассматривать как «инструкцию», которую утилита rpmbuild использует для фактической сборки RPM-пакета. Spec-файл определяет все действия, которые должны быть выполнены при сборке RPM-пакета, а также все действия, необходимые при установке/удалении пакета. Каждый src.rpm-пакет имеет в своем составе spec-файл.

Spec-файл – это текстовый файл. Соглашение об именовании предлагает называть spec-файл таким образом: имя_пакета.spec. Spec-файл 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.

Текст внутри spec-файла имеет специальный синтаксис. Синтаксические определения имеют значения, задающие порядок сборки, номер версии, информацию о зависимостях и вообще всю информацию о пакете, которая может быть впоследствии запрошена из БД RPM.

2.2.1 Пример spec-файла

Пример spec-файла:

```
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/*
%_man1dir/*

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

2.2.2 Директивы преамбулы

В Табл. 1 перечислены директивы, используемые в разделе преамбулы файла спецификации RPM.

Табл. 1 – Директивы преамбулы spec-файла

SPEC Директива	Определение
`Name`	Базовое имя пакета, которое должно совпадать с именем spec-файла.
`Version`	Версия upstream-кода.
`Release`	Релиз пакета используется для указания номера сборки пакета при данной версии upstream-кода. Для пакетов Sisyphus поле Release должно иметь вид в простых случаях – altN, а в сложных – altN[суффикс]. N начинается с 1 для каждой новой upstream-версии и увеличивается на 1 для каждой новой сборки: - 1.0-alt1 - 1.0-alt2
`Epoch`	Поле Epoch используется тогда, когда по какой-то причине требуется уменьшить версию или релиз пакета по сравнению с имеющимся в репозитории. При этом значение поля Epoch увеличивается на единицу по сравнению с предыдущим (отсутствие поля Epoch эквивалентно значению 0), версия и релиз устанавливаются в нужное значение. В имя RPM-файлов Epoch не входит, и поэтому необходимо избегать RPM с одинаковыми Version и Release и разными Epoch.
`Summary`	Краткое, однострочное описание пакета. Значение тэга Summary должно начинаться с заглавной буквы. В конце Summary не должно быть точки.
`License`	Лицензия на собираемое программное обеспечение. Лицензия должна быть указана в точности так, как сформулировано в upstream-пакете. При указании лицензии рекомендуется пользоваться макросами из пакета rpm-build-licenses, добавив его в список BuildRequires. Сам текст лицензии упаковывать в пакет нужно только в том случае, если соответствующий текст отсутствует в /usr/share/license (пакет common-licenses). Если же таковой файл присутствует, то достаточно указать название лицензии в тэге пакета.
`Group`	Используется для указания категории, к которой относится пакет. Указанная группа должна находиться в списке групп, известном RPM. Этот список располагается в файле /usr/lib/rpm/GROUPS, находящемся в пакете rpm.
`URL`	Полный URL-адрес для получения дополнительной информации о программе. В тэге URL настоятельно рекомендуется указывать действующий URL домашней страницы проекта, либо если таковой нет – любого другого места, где можно получить архив с исходным кодом.

	Для директивы URL можно использовать утилиту rpmurl, например, проверка доступности URL: `$ rpmurl -c пакет.spec`
`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 номер, 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 номер каждый раз, например: Patch1, Patch2, Patch3 и так далее.
`BuildArch`	Явное указание архитектуры, под которую собирается двоичный пакет. Если параметр не задан, то пакет автоматически наследует архитектуру машины, на которой он собран, например, x86_64. Если пакет не зависит от архитектуры, например, если он полностью написан на интерпретируемом языке программирования, следует установить для этой директивы значение noarch.
`BuildRequires`, `BuildPreReq`, `BuildRequires(pre)`	Разделенный запятыми или пробелами список пакетов, необходимых для сборки программы. Может быть несколько записей BuildRequires, каждая в отдельной строке. Тэг BuildRequires используется для хранения результатов работы утилиты buildreq. По этой причине дополнительные сборочные зависимости, не находящиеся buildreq, рекомендуется хранить в тэге BuildPreReq.
`Requires, PreReq`	Разделенный запятыми или пробелами список пакетов, необходимых программному обеспечению для запуска после установки. Это его зависимости. Может быть несколько записей Requires, каждая в отдельной строке.
`ExcludeArch`	Если программное обеспечение, для которого собирается двоичный пакет, не может работать на определенной архитектуре процессора, то можно исключить эту архитектуру данной директивой.
`Conflicts`	Разделенный запятыми или пробелами список пакетов, с которыми конфликтует данный пакет. Применяется для указания наличия конфликта (обязательно в случае файлового/RPC и желательно в случае существенного смыслового) между

	данным пакетом и указываемым.
`Provides`	Используется для указания того факта, что данный пакет предоставляет функциональность иного (переименованного устаревшего названия, широко известного по другим дистрибутивам либо же виртуального). Следует применять только в случае реальной необходимости и, как правило, в форме `Provides: something = %version-%release` При переименовании пакета обязательно сочетается с Obsoletes.
`Obsoletes`	Перечисляет пакеты/версии, объявленные устаревшими. Обычно применяется при переименовании пакета в сочетании с Provides и с указанием версии, меньшей или равной последней известной версии пакета под старым названием.

Каждая директива разделяется от ее значения символом «:». Между директивой и двоеточием не должно быть пробелов!

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.

Увидеть пример NAME-VERSION-RELEASE можно, 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. Исключением из этого правила является архитектурно-независимый пакет noarch.

2.2.3 Директивы основной части

В Табл. 2 перечислены директивы, используемые в основной части spec-файла, причем все они, кроме %check, являются обязательными.

Сколько бы двоичных пакетов не было описано в spec-файле, все директивы, кроме %files, должны быть указаны только один раз – разделение на разные двоичные RPM происходит именно в блоках %files согласно преамбуле.

Табл. 2 – Директивы основной части spec-файла

SPEC Директива	Определение
`%description`	Полное описание программного обеспечения, входящего в комплект поставки RPM. Это описание может занимать несколько строк и может быть разбито на абзацы. Длина каждой строки не должна превышать 72 символа. Данное описание учитывается при поиске пакета через apt-cache search и полностью выводится во время просмотра информации о пакете при помощи apt-cache show имя_пакета.
`%prep`	Команда или серия команд для подготовки программного обеспечения к сборке, например, распаковка архива, указанного в Source0. Эта директива может содержать сценарий оболочки (shell скрипт).
`%build`	Команда или серия команд для фактической сборки программного обеспечения в машинный код (для компилируемых языков) или байт-код (для некоторых интерпретируемых языков).
`%install`	Команды установки/копирования файлов из сборочного каталога в псевдо-корневой каталог. Во время сборки пакета этот раздел эмулирует конечные пути установки файлов в систему. Команда или серия команд для копирования требуемых артефактов сборки из %builddir (где происходит сборка) в %buildroot каталог (который содержит структуру каталогов с файлами, подлежащими сборке).
`%check`	Команда или серия команд для тестирования программного обеспечения. Обычно включает в себя модульные тесты.
`%files`	Список файлов, которые будут установлены в системе конечного пользователя.
`%changelog`	Запись изменений, произошедших в пакете между сборками разных версий или релизов.

2.3 RPM Макросы

Макросы RPM – это прямые текстовые подстановки, которые происходят путем замены определенных выражений и условий на соответствующий текст во время процесса сборки пакета. Иными словами, макросы являются псевдонимами для часто используемых фрагментов текста, применение которых сокращает не только объем ввода, но и делает спецификацию легче читаемой и воспринимаемой. Имена макросов начинаются с символа «%».

Список пакетов, содержащих макросы можно получить, выполнив команду:

```
$ apt-cache search rpm | grep '^rpm-[bm]' | sort -n
rpm-build-apache2 - Набор утилит для автоматической Web серверов и приложений
rpm-build-browser-plugins - Netscape Gecko Plug-in API common packaging files
rpm-build-compat - ALT Linux compatibility macros for backport purposes
```

```
rpm-build-dmd - RPM build enviroment to build D lang(dmd) packages
rpm-build-emacs - Helper scripts and RPM macros to build GNU Emacs extensions
rpm-build-erlang - RPM helper scripts to calculate Erlang dependencies
rpm-build-extra-targets - Build packages for other platforms
rpm-build-fedora-compat-fonts - Build-stage rpm automation for fonts packages
rpm-build-file - Утилита для определения типов файлов
rpm-build-firefox - RPM helper macros to rebuild firefox packages
rpm-build-fonts - RPM helper scripts for building font packages
…
```

Для использования данных макросов, необходимо добавить в spec-файл строку:

```
BuildRequires(pre): имя-пакета-с-макросами
```

Например:

```
BuildRequires(pre): rpm-build-java
```

Списки макросов находятся в каталоге `/usr/lib/rpm/macros.d/`.

Список доступных макросов и их значения:

```
$ rpm --showrc
```

Получить значение, раскрываемое макросом можно, выполнив команду:

```
$ rpm --eval <имя_макроса>
```

Например:

```
$ rpm --eval %_sysconfdir
/etc
```

Макросы можно использовать внутри других макросов. Так, например, если название архива исходных текстов проекта формируется из его имени и версии (директивы «Name» и «Version» транслируются в определенные макросы), то директива задания пути к файлу может выглядеть следующим образом:

```
Source0: %{name}-%{version}.tar.gz
```

П р и м е ч а н и е . Не следует использовать в спеках внутренние макросы RPM, которые начинаются с двух подчеркиваний (например, %__install или %__mkdir_p).

В Табл. 3 перечислены некоторые макросы.

Табл. 3 – Макросы путей системных каталогов

Макрос	Описание
`%homedir`	Домашний каталог пользователя, вызывающего этот макрос
`%_licensedir`	Каталог лицензий (/usr/share/license)
`%_controldir`	Каталог control (/etc/control.d/facilities)
`%_defaultdocdir`	Каталог документации (/usr/share/doc)

%_defattr	Атрибуты файлов и каталогов по умолчанию для каждой секции %files и для каждого файла, включаемого в таких секциях (-,root,root,755)
%_man1dir, %_man2dir, %_man3dir, %_man4dir, %_man5dir, %_man6dir, %_man7dir, %_man8dir, %_man9dir	Каталог man-файлов (/usr/share/man/manX)
%start_service()	Запуск установленного пакета как сервиса
%java_dir %_javadir	Каталог для некоторых jar файлов (/usr/share/java)
%_rpmmacrosdir	Каталог для установки сторонних макросов (/usr/lib/rpm/macros.d)

С целью сокращения написания часто требуемых команд при сборке пакета в блоках тела спецификации существует ряд полезных встроенных макросов. Некоторые из них рассмотрены ниже.

Макрос `%setup` – используется в блоке `%prep`. Этот макрос распаковывает архив с исходным кодом (с ключом -q подавляет подробный вывод при распаковке архива). По умолчанию распаковывается первый архив с исходным кодом (т. е. который имеет номер 0), для любых других необходимо использовать дополнительный параметр `-a X`, где `X` – номер, совпадающий с таковым у Source.

```
%prep
%setup -a1 -a100 -a101 -a102 -a103 -a104 -a105
%patch1 -p1
```

Макрос %setup в Sisyphus RPM использует ключ –q (подавляет подробный вывод при распаковке архива) по умолчанию. Запись %setup -q и %setup – полностью идентичны. Для включения отладочной информации следует использовать конструкцию с ключом -v.

`%patch[X]` – используется в блоке `%prep` и выполняет применение указанного патча (`X` – номер патча, такой же, как и при их описании в преамбуле, если патчей несколько). Применение патчей производится от текущего сборочного каталога, при необходимости обрезать часть пути к изменяемому файлу, можно использовать ключ `-p` (как и у самой утилиты `patch`). Например:

```
%patch3 -p1
```

Макрос `%autopatch` позволяет применить все патчи, описанные в основной секции spec-файла, в порядке возрастания их номера в имени. Макрос поддерживает использование ключей `-p` и `-F`, аналогичные таким же опциям директивы `%patch`.

Макрос %configure – применяется в блоке %build и используется для упрощения выполнения ./configure с соответствующими параметрами данной платформы. Почти всегда вполне достаточно выполнить %configure без параметров.

```
%build
%configure
%make_build
```

Если скрипта configure в архиве исходных текстов нет (обычное явление для исходников из git или иных SCM), но есть configure.ac – следует добавить перед вызовом %configure макрос %autoreconf.

Макрос %make_build – используется в блоке %build для выполнения команды make. По умолчанию поддерживает при сборке использование нескольких процессоров/ядер.

Макрос %makeinstall_std – применяется в блоке %install. Этот макрос рекомендуется применять вместо макроса %make_install с ключами install и DESTDIR=%buildroot:

```
%make_install DESTDIR=%buildroot install
```

Макрос %make_install – применяется в блоке %install. Используется для упрощения установки ПО и представляет собой запуск утилиты make с ключом install, и указанием каталога для установки (DESTDIR=%buildroot) и рядом других ключей:

```
%make_install DESTDIR=%buildroot install
```

или

```
%make_install DESTDIR=%buildroot %_make_install_target
```

Макрос %makeinstall – применяется в блоке %install. Это редко используемый макрос, предназначенный для установки ПО, Makefile которого не умеет использовать параметр DESTDIR.

Макрос %dir – макрос, используемый в блоке %files, указывающий, что путь является каталогом, который должен принадлежать этому RPM. Это указание важно для того, чтобы RPM точно знал, какие каталоги нужно очистить при удалении пакета.

Макрос %doc – макрос, используемый в блоке %files, обеспечивающий создание каталога документации (%_defaultdocdir/%name-%version) и копирование в него указанных в качестве аргументов файлов. Пути к файлам строятся относительно каталога сборки проекта, например:

```
%doc Examples
```

3 ИНСТРУМЕНТ GEAR

Gear (Get Every Archive from git package Repository) – система для работы с произвольными архивами программ. В качестве хранилища данных `gear` использует `git`, что позволяет работать с полной историей проекта.

Gear поддерживает полный цикл организации репозиториев:

- создание репозитория или импорт существующих src.rpm-пакетов;
- обновление upstream-кода в репозиториях;
- наложение патчей и пакетирование;
- экспорт pkg.tar и src.rpm, сборка бинарных RPM-пакетов.

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».

3.1 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 репозиториев, которые надо обработать, много. Но можно использовать fetch/push в цикле.

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, все равно придется исправить.

Форматированный changelog

В changelog релизного коммита имеет смысл включать соответствующий текст из changelog пакета, как это делают утилиты `gear-commit` (обертка 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 spec-файл самого пакета.

3.2 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-репозиториев, 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:

- любой файл из дерева, соответствующего коммиту;
- любой каталог из дерева, соответствующего коммиту в виде `tar-` или `zip-архива`;
- `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.

3.3 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: .
```

Все 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 следующего правила позволяет избежать коллизий:

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

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.

3.4 Быстрый старт Gear

3.4.1 Создание gear-репозитория путем импорта созданного ранее srpm-пакета

Исходные данные: srpm-пакет foobar-1.0-alt1.src.rpm со следующим содержимым:

```
$ rpm -qpl foobar-1.0-alt1.src.rpm
foobar-1-fix.patch
foobar-2-fix.patch
foobar.icon.png
foobar-1.0.tar.bz2
foobar-plugins.tar.gz
```

Для того чтобы сделать из этого srpm-пакета gear-репозиторий необходимо:

- создать каталог, в котором будет располагаться архив:

```
$ mkdir foobar
$ cd foobar
```

- создать новый git-репозиторий:

```
$ git init
Initialized empty Git repository in .git/
```

Получившийся пустой git-репозиторий будет выглядеть примерно следующим образом:

```
$ ls -dlog .*
drwxr-xr-x 4 4096 Aug 12 34:56 .
drwxr-xr-x 6 4096 Aug 12 34:56 ..
drwxr-xr-x 8 4096 Aug 12 34:56 .git
```

Таким образом, git-репозиторий готов для импорта srpm-пакета.

- импортировать srpm-пакет в git-репозиторий, воспользовавшись утилитой `gear-srpmimport`:

```
$ gear-srpmimport foobar-1.0-alt1.src.rpm
Committing initial tree deadbeefdeadbeefdeadbeefdeadbeefdeadbeef
```

```
gear-srpmimport: Imported foobar-1.0-alt1.src.rpm
gear-srpmimport: Created master branch
```

После выполнения импорта git-репозиторий будет выглядеть следующим образом:

```
$ ls -Alog
drwxr-xr-x 1 4096 Aug 12 34:56 .gear
drwxr-xr-x 1 4096 Aug 12 34:56 .git
-rw-r--r-- 1 6637 Aug 12 34:56 foobar.spec
drwxr-xr-x 3 4096 Aug 12 34:56 foobar
drwxr-xr-x 3 4096 Aug 12 34:56 foobar-plugins
-rw-r--r-- 1  791 Aug 12 34:56 foobar-1-fix.patch
-rw-r--r-- 1 3115 Aug 12 34:56 foobar-2-fix.patch
-rw-r--r-- 1  842 Aug 12 34:56 foobar.icon.png
```

- при необходимости в файл правил можно вносить изменения. Например, можно убрать сжатие исходников (соответствующие изменения следует вносить и в `.gear/rules`).

3.4.2 Создание gear-репозитория на основе готового git-репозитория

Для того чтобы создать gear-репозиторий на основе git-репозитория необходимо:

- создать и добавить в git-репозиторий spec-файл;
- создать и добавить в git-репозиторий файл с правилами `.gear/rules`.

3.4.3 Сборка пакета из gear-репозитория

Сборка пакета при помощи `hasher` осуществляется командой `gear-hsh`:

```
$ gear-hsh
```

Чтобы собрать пакет, который не содержит определения тега `Packager` в spec-файле, следует отключить соответствующую проверку:

```
$ gear-hsh --no-sisyphus-check=gpg,packager
```

Сборка пакета при помощи `rpmbuild(8)` осуществляется командой `gear-rpm`:

```
$ gear-rpm -ba
```

3.5 Фиксация изменений в репозитории

Для того чтобы сделать коммит очередной сборки пакета, имеет смысл воспользоваться утилитой `gear-commit`, которая помогает сформировать список изменений на основе записи в spec-файле:

```
$ gear-commit -a
```

Прежде чем сделать первый коммит, необходимо сконфигурировать адрес. Это можно сделать глобально, например, прописав соответствующие значения в `~/.gitconfig`:

```
$ git config --global user.name 'Your Name'
$ git config --global user.email '<login>@altlinux.org'
```

Для отдельно взятого git-репозитория можно сконфигурировать адрес, прописав соответствующие значения в `.git/config` этого git-репозитория:

```
$ git config user.name 'Your Name'
$ git config user.email '<login>@altlinux.org'
```

4 ИНСТРУМЕНТ HASHER

Hasher – это инструмент безопасной и воспроизводимой сборки пакетов. Все пакеты репозитория Сизиф собираются с его помощью.

4.1 Принцип действия

Hasher – инструмент для сборки пакетов в «чистой» и контролируемой среде. Это достигается с помощью создания в chroot минимальной сборочной среды, установки туда указанных в source-пакете сборочных зависимостей и сборке пакета в свежесозданной среде. Для сборки каждого пакета сборочная среда создается заново.

Такой принцип сборки имеет несколько следствий:

- все необходимые для сборки зависимости должны быть указаны в пакете. Для облегчения поддержания сборочных зависимостей в актуальном состоянии в Sisyphus используется инструмент buildreq;
- сборка не зависит от конфигурации компьютера пользователя, собирающего пакет, и может быть повторена на другом компьютере;
- изолированность среды сборки позволяет с легкостью собирать на одном компьютере пакеты для разных дистрибутивов и веток репозитория – для этого достаточно лишь направить hasher на различные репозитории для каждого сборочного окружения.

4.2 Пакеты hasher

hasher в дистрибутиве «alt-platform-builder» располагается в пакетах `hasher`, `hasher-priv`, которые установлены по умолчанию.

4.3 Справочная информация по hasher

Для получения подробной справки по hasher можно воспользоваться командой:

```
$ man hsh
```

4.4 Настройка Hasher

4.4.1 Добавление пользователя

`hasher` использует специальных вспомогательных пользователей и группу `hashman` для своей работы, поэтому каждого пользователя, который будет использовать `hasher`, перед началом работы нужно зарегистрировать:

```
# hasher-useradd <имя_пользователя>
```

Эта команда создает вспомогательных пользователей `имя_пользователя_a` и `имя_пользователя_b` и добавляет пользователя в группы `hashman`, `имя_пользователя_a` и `имя_пользователя_b`.

Поскольку `hasher-useradd` изменяет список групп, в которых состоит пользователь, пользователю перед началом работы с `hasher` необходимо перелогиниться.

4.4.2 Настройка сборочной среды

Для работы `hasher` требуется создать каталог, в котором будет строиться сборочная среда:

```
$ mkdir ~/.hasher
```

Рабочий каталог (в данном случае `~/.hasher`) должен быть доступен на запись пользователю, запускающему сборку. Кроме того, его нельзя располагать на файловой системе, которая смонтирована с опциями `noexec` или `nodev` – в таких условиях `hasher` не сможет создать корректное сборочное окружение.

Команда создания сборочного окружения:

```
$ hsh --initroot-only ~/.hasher
```

Явное создание сборочного окружения необязательно – при необходимости оно будет произведено при первой сборке пакета.

`hasher` берет пакеты для установки из APT-источников. По умолчанию в сборочную среду копируется список источников, указанный в конфигурации APT хост-системы; также можно явно задать дополнительные репозитории, указав альтернативный файл конфигурации APT, например:

```
$ hsh --apt-config=.hasher/p10-apt.conf --initroot-only ~/.hasher
```

В таком файле конфигурации (в примере `p10-apt.conf`) необходимо указать расположение файла с APT-источниками, например:

```
Dir::Etc::SourceList "/home/user/.hasher/sources_p10.list";
```

Если необходимо создать сборочную среду, независимую по источникам от основной операционной системы, в вышеуказанный файл помимо строки с источником следует добавить следующую строку во избежание включения `/etc/apt/sources.list.d/*.list`:

```
Dir::Etc::SourceParts "/var/empty";
```

По умолчанию (без указания в команде ключа `--apt-config`) задействуется общесистемная конфигурация репозитория из /etc/apt/. Чтобы не указывать каждый раз ключ `--apt-config` можно указать его в файле конфигурации `~/.hasher/config`, например:

```
apt_config=/home/user/.hasher/p10-apt.conf
```

Пример файла `~/.hasher/p10-apt.conf`:

```
Dir::Etc::SourceList "/home/user/.hasher/sources_p10.list";
Dir::Etc::SourceParts "/var/empty";
```

Пример файла `~/.hasher/sources_p10.list` с локальным репозиторием:

```
rpm file:/srv/public/mirror p10/branch/x86_64 classic
rpm file:/srv/public/mirror p10/branch/ x86_64-i586classic
rpm file:/srv/public/mirror p10/branch/noarch classic
```

4.5 Сборка в hasher

Сборка происходит от обычного пользователя, добавленного с помощью `hasher-useradd`:

```
$ hsh ~/.hasher path/to/package-0.0-alt0.src.rpm
```

При удачной сборке полученные пакеты будут находиться в каталоге `~/.hasher/repo/<платформа>/RPMS.hasher/`, в противном случае на `stdout` будет выведена информация об ошибках сборки.

Для наблюдения за процессом сборки следует использовать ключ -v.

Создаваемый `hasher` репозиторий является обычным APT-репозиторием и может быть использован в `sources.list[3]`. Он также будет использован при дальнейшей сборке пакетов (это поведение можно регулировать ключом `--without-stuff`).

Если сборочная среда создана в `tmpfs` (см. ниже), каталог `~/.hasher/repo`, вероятно, не переживет перезагрузку системы. Репозиторий можно переместить в постоянное место, указав в файле конфигурации `hasher` (`~/.hasher/config`) параметр `def_repo=постоянное_хранилище` (или вызвав `hasher` с ключом `--repo`).

4.6 Сборочные зависимости

Сборочные зависимости RPM делятся на два вида:

- необходимые для корректного создания src.rpm из spec-файла (содержащие определения RPM-макросов, используемых в spec-файле);
- все остальные (необходимые для непосредственной сборки).

Поскольку `hasher` собирает пакеты из src.rpm (не считая поддержки `gear`), то для сборки в хост-системе необходимо иметь установленные сборочные зависимости первого типа. Большинство таких зависимостей (но пока не все) содержатся в пакетах с названием `rpm-build-*`.

Сборка `src.rpm` либо завершается неудачно (при отсутствии сборочной зависимости первого типа), либо корректно, поэтому собирать src.rpm-пакеты в хост системе можно с помощью параметра `--nodeps`:

```
rpm -bs --nodeps package.spec
```

hasher, в отличие от gear, не предъявляет никаких требований к разделению сборочных зависимостей на первый и второй тип. Однако для совместимости с gear и для улучшения документируемости spec-файла рекомендуется распределять их так:

- в поле BuildRequires(pre) помещать сборочные зависимости, требуемые для сборки src.rpm;
- в поле BuildRequires – все остальные.

Примечание. В поле BuildRequires(pre) нельзя использовать макросы.

4.7 Монтирование файловых систем внутри hasher

Некоторым приложениям для сборки требуется смонтированная файловая система (например, /proc). hasher поддерживает монтирование дополнительных файловых систем в сборочную среду.

Монтирование происходит при одновременном выполнении следующих четырех условий:

- файловая система описана в файле /etc/hasher-priv/fstab, либо является одной из предопределенных: /proc, /dev/pts, /sys;
- файловая система указана в опции allowed_mountpoints в конфигурации hasher-priv (/etc/hasher-priv/system);
- файловая система указана при запуске hasher в опции --mountpoints, либо указана в ключе known_mountpoints конфигурационного файла hasher (~/.hasher/config);
- файловая система указана сборочной зависимостью (например, BuildReq: /proc) собираемого пакета, прямой или косвенной (через зависимости сборочных зависимостей пакета).

Для монтирования /proc необходимо:

- в /etc/hasher-priv/system добавить строку:

```
allowed_mountpoints=/proc
```

- в ~/.hasher/config добавить строку (либо указывать опцию --mountpoints=/proc при сборке пакета):

```
known_mountpoints=/proc
```

- в spec-файле пакета указать BuildRequires: /proc

4.8 Использование нескольких сборочных окружений

`hasher` не ограничивает пользователей одним сборочным окружением. Первый параметр, передаваемый `hsh`, указывает на конкретную сборочницу, в которой необходимо производить работу:

```
$ hsh --apt-conf=.hasher-p10/apt.conf.p10 ~/.hasher-p10
package-0.0-alt0.src.rpm
…
$ hsh --apt-conf=.hasher-test/apt.conf.test ~/.hasher-test
package-1.0-alt0.src.rpm
…
```

По умолчанию используется каталог `~/hasher`.

4.9 Сборка пакетов на tmpfs

При наличии достаточного количества памяти на сборочной машине сборку пакетов можно производить на tmpfs – такая конфигурация заметно ускоряет сборку.

Можно взять уже смонтированный `/tmp`:

```
$ mkdir -p /tmp/.private/$USER/hasher
$ hsh --repo=$HOME/hasher-repo /tmp/.private/$USER/hasher
package-0.0-alt0.src.rpm
```

Каталог `/tmp/.private/$USER/hasher` нужно будет создавать после каждой перезагрузки (это можно сделать в файле `~/.hasher/config`, воспользовавшись тем, что это shell-скрипт). Указывать `--repo` нужно для каждой сборки (либо следует указать в `.hasher/config` параметр `def_repo=постоянное_хранилище`).

4.10 Отключение проверок sisyphus_check

По умолчанию `hasher` запускает утилиту `sisyphus_check` с полным набором тестов. `sisyphus_check` проверяет не только технические требования репозитория Sisyphus, но и организационные: сборочный хост, подпись PGP-ключом члена ALT Linux Team и т.д., так что в случае сборки пакета не для репозитория Sisyphus возникает необходимость отключить часть проверок.

Для отключения части или всех проверок используется ключ `--no-sisyphus-check[=LIST]` или, что эквивалентно, опция конфигурационного файла `no_sisyphus_check`.

Без аргумента этот ключ отключает запуск `sisyphus_check` вообще:

```
$ hsh --no-sisyphus-check ~/.hasher package-0.0-alt0.src.rpm
```

С аргументом, списком отключаемых тестов, отключает только эти тесты:

```
$ hsh --no-sisyphus-check=packager,gpg ~/.hasher
package-0.0-alt0.src.rpm
```

Список тестов можно увидеть в справке `sisyphus_check`:

```
$ sisyphus_check --help
…
Valid options are:
…
  --[no-]check-buildhost
  --[no-]check-buildtime
  --[no-]check-changelog
…
```

Более тонко запуск тестов можно настроить с помощью опций `--no-sisyphus-check-in` и `--no-sisyphus-check-out`, с описанием которых можно ознакомиться в man-странице `hsh(1)`.

4.11 Отладка в сборочном chroot

Для отладки сборки иногда полезно запустить `shell` в сборочном `chroot`. Для этого используется утилита `hsh-shell(1)`:

```
$ hsh-shell ~/.hasher
```

Можно запустить программу и с правами псевдо-root:

```
$ hsh-shell --rooter
```

4.12 Ограничение ресурсов

`hasher` позволяет ограничить ресурсы, выделяемые на сборку: CPU, память, общее время исполнения и другие. Ограничения указываются в конфигурационном файле `hasher-priv`.

Полный список ограничиваемых ресурсов можно найти в man-странице `hasher-priv.conf(5)`.

4.13 Пересборка пакета без пересоздания всего chroot

Развертывание всей сборочной среды и зависимостей идет небыстро.

Для того чтобы собирать один и тот же пакет до тех пор, пока он не соберется, нужно указать hasher не разворачивать заново всю сборочницу, либо работать в самой сборочнице.

4.13.1 Многократная сборка пакета в одном hasher

Если пакет не собрался можно воспользоваться `hsh-shell` (предварительно, установив текстовый редактор, shell и прочие инструменты разработчика):

```
$ hsh-install ~/.hasher vim-console less rpm-utils patchutils zsh
$ mkdir -p ~/.hasher/chroot/.in/src
$ cp -
a .vim* .zprofile .zsh_aliases .zshenv .zsh_bind .zshrc .dircolors
~/.hasher/chroot/.in/src
$ hsh-run -- cp -r /.in/src /usr
$ hsh-shell --shell=/bin/zsh
```

В дереве каталогов hasher есть каталог `~/hasher/chroot/.in`, в которой может писать сам пользователь.

Отсутствующие сборочные зависимости можно доставлять с помощью `hsh-install` (выйдя из chroot).

4.13.2 Многократная сборка пакета при работе с gear

Если используется `gear` и разработка ведется в базовой системе, вместо `hsh` можно использовать `hsh-rebuild` – программу, работающую с уже сформированным сборочным окружением.

Первоначальная сборка (даже если прошла успешно, окружение не удаляется):

```
$ gear --hasher -- hsh --lazy-cleanup
```

Повторная сборка:

```
$ gear --hasher -- hsh-rebuild
```

Примечание. По окончании работы необходимо выполнить `buildreq` и забрать spec:

```
$ hsh-install rpm-utils
$ echo buildreq '/usr/src/RPM/SPECS/*.spec' | hsh-shell
$ cp ~/hasher/chroot/usr/src/RPM/SPECS/*.spec .
```

5 ПРИМЕРЫ СБОРКИ ПАКЕТОВ

Для примера сборки пакета используется программа для вывода системных уведомлений о текущей дате и времени. Ссылки на github-репозитории с исходными текстами программ:

- C++: Notification – https://github.com/MakDaffi/notification;
- Python: DBusTimer_Example – https://github.com/danila-Skachedubov/DBusTimer_example.

Структура репозиториев для программ Notification и DBusTimer_Example идентична: главный файл (.cpp или .py) и два юнита systemd (.service и .timer):

- файл .timer – юнит systemd, который при истечении заданного времени вызывает скрипт .py, выводящий уведомление о дате и времени. После срабатывания таймер снова начинает отсчет времени до запуска скрипта;
- файл .service – содержит описание, расположение скрипта .py и интерпретатора, который будет обрабатывать скрипт.

5.1 Пакет с исходными текстами на Python

5.1.1 Подготовка пространства

В первую очередь необходимо склонировать репозиторий в рабочий каталог, используя команду `git clone`:

```
$ git clone https://github.com/danila-Skachedubov/DBusTimer_example.git

remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 9 (delta 1), reused 8 (delta 1), pack-reused 0
Receiving objects: 100% (9/9), done.
Resolving deltas: 100% (1/1), done
```

В рабочем каталоге появится каталог с названием проекта: DBusTimer_Example.

Перейти в каталог DBusTimer_Example и создать в нем каталог .gear:

```
$ cd DBusTimer_example
$ mkdir .gear
```

В каталоге .gear создать два файла: файл правила для gear – rules и spec-файл – dbustimer.spec

```
$ touch .gear/rules .gear/dbustimer.spec
```

Содержимое каталога DBusTimer_Example в результате проделанных действий:

```
$ ls -a1
.
```

```
..
.gear
.git
script_dbus.py
script_dbus.service
script_dbus.timer
```

5.1.2 Написание spec файла и правил Gear

Следующий этап сборки – это написание spec-файла и правил для gear.

Заполнить файл .gear/rules следующим содержимым:

```
tar: .
spec: .gear/dbustimer.spec
```

Первая строка указывает, что проект будет упакован в .tar архив. Вторая строка указывает путь к расположению spec-файла.

Далее необходимо заполнить spec-файл.

В заголовке spec-файла находятся секции `Name`, `Version`, `Release`, `Summary`, `License`, `Group`, `BuildArch`, `BuildRequires`, `Source0`.

Для данного примера можно заполнить секции заголовка следующим образом:

```
Name: dbustimer
Version: 0.4
Release: alt1

Summary: Display system time
License: GPLv3+
Group: Other
BuildArch: noarch

BuildRequires: rpm-build-python3
```

Стандартная схема `Name-Version-Release`, содержит имя пакета, его версию и релиз сборки. Поле `Summary` включает в себя краткое описание пакета. `License` – лицензия, под которой выпускается данное ПО. В данном случае – GPLv3. `Group` – категория, к которой относится пакет. Так как это тестовый пакет, можно указать группу «Other». `BuildRequires` – пакеты, необходимые для сборки. Так как исходный код написан на python3, необходимо указать пакет `rpm-build-python3`, который содержит макросы для сборки скриптов Python. `Source0` – путь к архиву с исходниками (%name-%version.tar).

В теле, или основной части spec-файла, описываются процесс сборки и инструкции к преобразованию исходных файлов.

В секции `%description` находится краткое описание программы.

Секция `%prep` отвечает за подготовку программы к сборке. Макрос `%setup` распаковывает исходный код перед компиляцией. Пример заполнения полей `%description` и `%prep`:

```
%description
This program displays notifications about the system time with a frequency of one hour.

%prep
%setup -q
```

В секции `%install` описываются инструкции, куда в систему конечного пользователя будут скопированы файлы пакета. Чтобы не указывать пути установки файлов вручную, можно использовать предопределенные макросы: `%python3_sitelibdir_noarch` – раскрывается в путь `/usr/lib/python3/site-packages`. По этому пути будет создан каталог с именем пакета, в который будет помещен файл `script_dbus.py` с правами доступа 755. Аналогичная операция будет проведена с файлами `script_dbus.timer` и `script_dbus.service`. Они должны быть скопированы в каталог `/etc/xdg/systemd/user`. Так как макроса, раскрывающегося в данный каталог нет, можно использовать макрос `%_sysconfdir`, который раскрывается в путь `/etc`. Пример заполнения секции `%install`:

```
%install


mkdir -p \
        %buildroot%python3_sitelibdir_noarch/%name/
install -Dm0755 script_dbus.py \
        %buildroot%python3_sitelibdir_noarch/%name/

mkdir -p \
        %buildroot%_sysconfdir/xdg/systemd/user/
cp script_dbus.timer script_dbus.service \
        %buildroot%_sysconfdir/xdg/systemd/user/
```

Команда `mkdir -p \ %buildroot%python3_sitelibdir_noarch/%name/` создает каталог `dbustimer` в окружении `buildroot` по пути `/usr/lib/python3/site-packages`.

Следующим действием происходит копирование файла `script_dbus.py` с правами 755 в каталог `/usr/lib/python3/site-packages/dbustimer/` в окружении `buildroot`.

Аналогично создается каталог `%buildroot%_sysconfdir/xdg/systemd/user/`, в который копируются файлы `.service` и `.timer`.

В секции `%files` описывается, какие файлы и каталоги с соответствующими атрибутами должны быть скопированы из дерева сборки в rpm-пакет, а затем будут копироваться в целевую систему при установке этого пакета. Все три файла из пакета будут распакованы по путям, описанным в секции `%install`. Пример заполнения секции `%files`:

```
%files
%python3_sitelibdir_noarch/%name/script_dbus.py
/etc/xdg/systemd/user/script_dbus.service
/etc/xdg/systemd/user/script_dbus.timer
```

В секции `%changelog` описываются изменения, внесенные в ПО, патчи, изменения методологии сборки. Пример секции `%changelog`:

```
%changelog
* Thu Apr 13 2023 Danila Skachedubov <dan@altlinux.org> 0.4-alt1
- Update system
- Changed access rights
```

Итоговый spec-файл может выглядеть следующим образом:

```
Name: dbustimer
Version: 0.4
Release: alt1

Summary: Display system time
License: GPLv3+
Group: Other
BuildArch: noarch

BuildRequires: rpm-build-python3

Source0: %name-%version.tar

%description
```

```
This program displays notifications about the system time with a frequency of one
hour.

%prep
%setup

%install

mkdir -p \
        %buildroot%python3_sitelibdir_noarch/%name/
install -Dm0755 script_dbus.py \
        %buildroot%python3_sitelibdir_noarch/%name/

mkdir -p \
        %buildroot%_sysconfdir/xdg/systemd/user/
cp script_dbus.timer script_dbus.service \
        %buildroot%_sysconfdir/xdg/systemd/user/



%files
%python3_sitelibdir_noarch/%name/script_dbus.py
/etc/xdg/systemd/user/script_dbus.service
/etc/xdg/systemd/user/script_dbus.timer

%changelog
* Thu Apr 13 2023 Danila Skachedubov <dan@altlinux.org> 0.4-alt1
- Update system
- Changed access rights
```

После заполнения файлов данными необходимо добавить эти файлы на отслеживание `git`. Сделать это можно с помощью команды:

```
$ git add .gear/rules .gear/dbustimer.spec
```

После добавление файлов на отслеживание, необходимо запустить сборку с помощью инструментов `gear` и `hasher`:

```
$ gear-hsh ~/.hasher --no-sisyphus-check --commit –v
```

П р и м е ч а н и е . `Hasher` и `git` должны быть предварительно настроены.

Если сборка прошла успешно, собранный пакет `dbustimer-0.4-alt1.noarch.rpm` будет находиться в каталоге `~/.hasher/repo/x86_64/RPMS.hasher/`.

5.2 Пакет с исходными текстами на C++

Программа Notification выводит системное уведомление о текущей дате и времени в формате: День недели, месяц, число, чч:мм:сс, год.

В репозитории находятся следующие файлы:

- .gear – каталог с правилами gear и spec-файлом;
- Makefile – набор инструкций для программы make, которая собирает данный проект;
- notify.cpp – исходный код программы;
- notify.service – юнит данной программы для systemd;
- notify.timer – юнит systemd, запускающий вывод уведомления о дате и времени с периодичностью в один час.

В каталоге .gear находятся два файла:

- rules – правила для упаковки архива для gear;
- notify.spec – файл спецификации для сборки пакета.

Содержимое файла rules:

```
tar: .
spec: .gear/notify.spec
```

Первая строка – указания для gear, в какой формат упаковать файлы для последующей сборки. В данном проекте архив будет иметь вид name-version.tar. Вторая строка – путь к spec-файлу с инструкциями по сборке текущего пакета.

Содержимое файла notify.spec:

```
Name: notify
Version: 0.1
Release: alt1

Summary: Display system time every hour
License: GPLv3+
Group: Other

BuildRequires: make
BuildRequires: gcc-c++
BuildRequires: libsystemd-devel

Source0: %name-%version.tar

%description
```

```
This test program displays system date and time every hour via notification

%prep
%setup -q

%build
%make_build

%install

mkdir -p \
        %buildroot/bin/
install -Dm0644 %name %buildroot/bin/

mkdir -p \
        %buildroot%_sysconfdir/xdg/systemd/user/
cp %name.timer %name.service \
        %buildroot%_sysconfdir/xdg/systemd/user/

%files
/bin/%name
/etc/xdg/systemd/user/%name.service
/etc/xdg/systemd/user/%name.timer

%changelog
* Thu Apr 13 2023 Sergey Okunkov <sok@altlinux.org> 0.1-alt1
- Finished my task
```

В заголовке spec-файла описаны следующие поля:

- `Name`, `Version`, `Release` – стандартная схема Name-Version-Release, содержащая в себе имя пакета, его версию и релиз сборки;
- `Summary` – краткое описание пакета;
- `License` – лицензия, под которой выпускается данное ПО. В данном случае – GPLv3;
- `Group` – категория, к которой относится пакет. В примере указана группа «Other»;
- `BuildRequires` – пакеты, необходимые для сборки. Так как исходный код написан на `c++`, необходим компилятор `g++`, система сборки программы – `make` и библиотека для работы с модулями systemd – `libsystemd-devel`.

- `Source0` – путь к архиву с исходниками (`%name-%version.tar`).

В теле spec-файла описываются процесс сборки и инструкции к преобразованию исходных файлов:

- секция `%description` содержит описание того, что делает программа. В данном примере – вывод системного уведомления с датой и временем;
- секция `%prep` отвечает за подготовку программы к сборке. Макрос `%setup` с флагом `-q` распаковывает архив, описанный в секции `Source0`;
- секция `%build` описывает сборку исходного кода. Так как в примере присутствует Makefile для автоматизации процесса сборки, то в секции указан макрос `%make_build`, использующий Makefile для сборки программы;
- секция `%install` эмулирует конечные пути при установке файлов в систему. Так как файла три, для каждого должен быть прописан конечный путь:
  - notify – скомпилированный бинарный файл. В Unix-подобных системах бинарные файлы располагаются в каталоге `/bin`. `mkdir -p %buildroot/bin` – строка, в которой создается каталог bin в окружении buildroot. Следующая строка – `install -Dm0644 %name %buildroot/bin/` – установка бинарного файла notify в каталог `%buildroot/bin/` с правами 644;
  - %name.timer, %name.service – юниты systemd. Это пользовательские юниты, которые должны находиться в каталоге `/etc/xdg/systemd/user/`. В окружении buildroot создается каталог:
    `mkdir -p %buildroot%_sysconfdir/xdg/systemd/user/`. В пути использован макрос %_sysconfdir, который заменяется путем /etc. Следующая строка: `cp %name.timer %name.service %buildroot %_sysconfdir/xdg/systemd/user/` – переносит данные файлы по заданному пути в окружении `buildroot`;
- секция `%files` – описывает файлы и каталоги, которые будут скопированы в систему при установке пакета.

6 СБОРКА ОБРАЗОВ С ПОМОЩЬЮ MKIMAGE-PROFILES

Система генерации дистрибутивов использует все преимущества банка пакетов и позволяет получить установочные образы. Для сборки образа используется утилита mkimage, которая использует для сборки «профиль», представляющий из себя набор файлов Makefile. В результате из пакетов репозитория создается установочный диск CD/DVD. Целостность репозитория и его непротиворечивость позволяют с легкостью генерировать новые образы при необходимости.

mkimage-profiles (m-p) – система управления конфигурацией семейств дистрибутивов свободного программного обеспечения из репозиториев ALT для различных платформ. Целостность репозитория и его непротиворечивость позволяют с легкостью генерировать новые образы.

Концепция:

- конфигурация, как и образ – объект постадийной сборки;
- метапрофиль служит репозиторием для построения индивидуального профиля, по которому создается итоговый образ.

Особенности:

- метапрофиль при сборке может быть доступен только на чтение;
- для сборки выбирается предпочтительно tmpfs;
- в профиль копируются только нужные объекты (он автономен относительно метапрофиля).

Стадии работы:

- инициализация сборочного профиля;
- сборка конфигурации образа;
- наполнение сборочного профиля;
- сборка образа.

Примечание. Сборка образов может занимать большой объем дискового пространства (например, порядка 100 Гб).

Предварительные настройки:

- пользователь с правом запуска hasher и подключения /proc к нему (см. «Настройка Hasher»);
- смонтированный tmpfs на несколько гигабайт (можно указать в переменной BUILDDIR):
  - например, в `/tmp` или `/home/USER/hasher`;
  - каталог из `prefix` в `/etc/hasher-priv/system`;
- настроенный `~/.gitconfig`.

Объекты:

- дистрибутивы и виртуальные среды/машины:
  - описываются в `conf.d/*.mk`;
  - могут основываться на предшественниках, расширяя их;
  - дистрибутивы могут включать один или более субпрофилей;
  - следует избегать множественного наследования;
- субпрофили:
  - список собирается в `$(SUBPROFILES)`;
  - базовые комплекты помещены в подкаталоги sub.in/;
  - наборы скриптов базовых комплектов могут расширяться фичами (features);
- фичи (features):
  - законченные блоки функциональности (или наборы таковых);
  - описываются в индивидуальных `features.in/*/config.mk`;
  - могут требовать другие фичи, а также субпрофили;
  - накопительный список собирается в `$(FEATURES)`;
  - при сборке `$(BUILDDIR)` содержимое фич добавляется в профиль;
- списки пакетов `(*_LISTS)`:
  - не следует создавать фичу, если достаточно списка пакетов;
  - следует по возможности избегать дублирования (см. bin/pkgdups);
- индивидуальные пакеты `(*_PACKAGES)`: см. conf.d/README.

Результат:

- при успешном завершении сборки образ называется по имени цели и укладывается в `$(IMAGEDIR)`:
  - указанный явно;
  - `~/out/` (если возможно);
  - `$(BUILDDIR)/out/`;
- формируются отчеты, если запрошены (REPORT).

При запуске на сборку принимается ряд переменных (см. profiles.mk.sample). Переменные могут быть заданы, как в команде сборки в качестве аргументов, так и в файле настроек `$HOME/.mkimage/profiles.mk`. Список переменных приведен в Табл. 4.

Табл. 4 – Переменные, принимаемые при сборке

Переменная	Описание	Значение
APTCONF	Задает путь к apt.conf	Пусто (по умолчанию системный) либо строка
ARCH	Задает целевую архитектуру образов	Пусто (по умолчанию авто) либо строка
ARCHES	Задает набор целевых архитектур при параметрическом задании APTCONF	Пусто (по умолчанию авто) либо список через пробел
AUTOCLEAN	Включает уборку (distclean) после успешной сборки образа	Пусто (по умолчанию нет) либо любая строка
BELL	Подает сигнал после завершения сборки	Пусто (по умолчанию нет) либо любая строка
BRANCH	Указывает, для какого бранча производится сборка	-не определено – пытается определиться автоматически; -пусто – присваивается значение sisyphus; -имя бранча (sisyphus, p10).
BUILDDIR	Задает каталог генерируемого профиля и сборки	Пусто (по умолчанию авто) либо строка
BUILDDIR_PREFIX	Задает префикс каталога генерируемого профиля и сборки	Строка; по умолчанию выбирается алгоритмически
BUILDLOG	Задает путь к файлу журнала сборки/очистки	$(BUILDDIR)/build.log (по умолчанию) либо строка
CHECK	Включает режим проверки сборки конфигурации (без сборки образа)	-пусто (по умолчанию) – проверка не осуществляется; -0 – проверяется только конфигурация, списки пакетов не проверяются; -другое значение – полная проверка
CLEAN	Экономия RAM+swap при сборке в tmpfs. Очистка рабочего каталога после успешной сборки очередной стадии	Пусто, 0, 1, 2; по умолчанию пусто при DEBUG, иначе 1
DEBUG	Включает средства отладки, может отключить зачистку после сборки	Пусто (по умолчанию), 1 или 2
DISTRO_VERSION	Задает версию дистрибутива, если применимо	Пусто (по умолчанию) либо любая строка
HOMEPAGE,	Указывают адрес, название и таймаут	Корректный URL, строка, целое

HOMENAME, HOMEWAIT	перехода для домашней страницы	неотрицательное число
IMAGEDIR	Указывает путь для сохранения собранного образа	Равно $HOME/out, если существует, иначе $(BUILDDIR)/out (по умолчанию), либо другой путь
ISOHYBRID	Включает создание гибриднго ISO-образа	Пусто (по умолчанию) либо любая строка
LOGDIR	Указывает путь для сохранения логов сборки	Равно $(IMAGEDIR) (по умолчанию), либо другой путь
MKIMAGE_PREFIX	Указывает путь до mkimage. Если параметр не указан, то используется системный mkimage	
NICE	Понижает нагрузку системы сборочной задачей	Пусто (по умолчанию) либо любая строка
NO_SYMLINK	Не создавать символические ссылки на собранный образ	Пусто (по умолчанию) либо любая строка
QUIET	Отключает поясняющие сообщения при сборке (например, под cron)	Пусто (по умолчанию) либо любая строка
REPORT	Запрашивает создание отчетов о собранном образе. Требует включения DEBUG и отключения CHECK	-пусто (по умолчанию) – создание отчета выключено -2 – создать архив из каталога отчета -любая другое непустое значение – создать отчет в виде каталога
ROOTPW	Устанавливает пароль root по умолчанию для образов виртуальных машин	Пусто (по умолчанию root) либо строка
SAVE_PROFILE	Сохраняет архив сгенерированного профиля в .disk/	Пусто (по умолчанию) либо любая строка
SORTDIR	Дополнительно структурирует каталог собранных образов	Пусто (по умолчанию) либо строка (например, $(IMAGE_NAME)/$(DATE))
SQUASHFS	Определяет характер сжатия squashfs для stage2	-пусто (по умолчанию) либо normal: xz -tight: xz с -Xbcj по платформе (лучше, но дольше – подбор в два прохода) -fast: gzip/lzo (быстрее запаковывается и распаковывается, меньше степень)

STATUS	Добавляет в имя образа указанный префикс	Пусто (по умолчанию) либо строка (например, "alpha", "beta")
STDOUT	Выводит сообщения, при включенном DEBUG, одновременно в лог и на экран	1 – включить вывод на экран, если включен DEBUG
USE_QEMU	Использовать qemu, если архитектура не совпадает	1 (по умолчанию), для отключения любое другое значение
VM_SAVE_TARBALL	Указывает, что нужно сохранить промежуточный тарбол, из которого создается образ виртуальной машины, в заданном формате	tar tar.gz tar.xz
VM_SIZE	Задает размер несжатого образа виртуальной машины в байтах	Пусто (по умолчанию двойной размер чрута) или целое

Примечание. Для того чтобы при указании переменной BRANCH сборка осуществлялась для целевого бранча, требуется:

- прописать в ~/.mkimage/profiles.mk:

```
APTCONF = ~/apt/apt.conf.$(BRANCH).$(ARCH)
```

- создать целевые конфигурационные файлы apt по указанным путям.

Помимо этого переменная BRANCH, если определена, заменяет в имени собираемой цели слово «regular» на «alt-$BRANCH». Таким образом, достигается сборка стартеркитов из профиля регулярок под заданный бранч.

Список доступных целей:

```
$ make -C /usr/share/mkimage-profiles help
```

Список доступных образов:

```
$ make -C /usr/share/mkimage-profiles help/distro
```

Пример команды сборки образа:

```
$ make -C /usr/share/mkimage-profiles syslinux.iso
```

Пример команды сборки образа с указанием переменных:

```
$ make -C /usr/share/mkimage-profiles DEBUG=1 BRANCH=p10
APTCONF=/etc/apt/apt.conf.local alt-server.iso
```

Содержимое apt.conf.local может выглядеть так:

```
Dir::Etc::main "/dev/null";
Dir::Etc::parts "/ver/empty";
Dir::Etc::SourceParts "/var/empty";
```

```
Dir::Etc::sourcelist "/etc/apt/sources.list.d/alt-local.list";
APT::Acquire::Retries "3";
APT::Cache-Limit 201326592;
//Acquire::http::AllowRedirect "true";

RPM
{
        Allow-Duplicated {
                // Old-style kernels.
                   "^(NVIDIA_)?(kernel|alsa)[0-9]*(-adv|-linus)?($|-up|-smp|-secure|-
custom|-enterprise|-BOOT|-tape|-aureal)";
                // New-style kernels.
                "^kernel-(image|modules)-.*";
        };
        Hold {
                // Old-style kernels.
                "^(kernel|alsa)[0-9]+-source";
        };
};
```

7 JOIN

7.1 Разработка в ALT Linux Team

ALT Linux Team – команда независимых разработчиков свободного программного обеспечения, которые совместно работают над поддержкой наборов программ в репозитории Сизиф (Sisyphus).

У каждого пакета в Sisyphus есть один или несколько мейнтейнеров. Мейнтейнер – это человек, который собирает пакет для установки программы из централизованного репозитория, исправляет обнаруженные в программе ошибки, общается с разработчиками программы, старается реагировать на нужды пользователей (багрепорты, вопросы). Мейнтейнер должен быть членом ALT Linux Team (команды ALT).

Join – это процесс вступления в ALT Linux Team, результатом которого является возможность непосредственно участвовать в разработке репозитория Сизиф. После прохождения Join кандидат (человек, вступающий в ALT Linux Team) становится мейнтенером.

Для принятия человека в Team создается небольшая команда:

- Секретарь команды. Секретарь – это административная должность в ALT Linux Team. В его обязанность входит отслеживать стадии принятия и выполнять административные действия.
- Ментор вступающего. Задача ментора – способствовать кандидату со вступлением в Team. Он помогает кандидату со вступлением, отвечает на его вопросы и принимает решение о готовности кандидата.
- Рецензент работы вступающего. Задача рецензента – оценить готовность кандидата к вступлению в Team. Рецензент проводит независимую оценку готовности вступающего по результатам его работы и подтверждает его готовность.
- Кандидат – вступающий в ALT Linux Team человек.

При взаимодействии друг с другом можно использовать определенный номер стадии, до которой дошел кандидат. Например, секретарь регистрирует ключ кандидата и переводит процесс на стадию 2.2; или ментор решает, что его подопечный готов отправлять пакеты в Сизиф, и переводит процесс на стадию 4.0.

Официальное взаимодействие происходит в Bugzilla в содержимом особым образом оформленной ошибки (бага). Открытый баг означает заявку на вступление в тим, закрытый – само вступление или отказ в нем.

7.2 Создание заявки на вступление в ALT Linux Team

Для принятия в Team необходима следующая информация:

- имя ментора – участника команды, имеющего желание помогать в процессе приема в Team. Менторов можно искать в списках рассылки или канале Telegram;
- псевдоним (имя пользователя) участника. Псевдоним выбирается самим участником. Имя должно начинаться с буквы, содержать только строчные латинские буквы и цифры, быть не короче трех символов;
- адрес почты, на который будет производиться пересылка с адреса псевдоним@altlinux.org;
- SSH-ключ (ED25519 или RSA с размером не менее 4096 бит). Принимающему нужна публичная часть ключа. Этот ключ будет использоваться для SSH-доступа на ресурсы Sisyphus (git.alt и другие);
- GPG-ключ (RSA с размером не менее 4096 бит). В ключе должны быть указаны имя в формате <First name> <Last name> и uid вида псевдоним@altlinux.org. Принимающему нужна публичная часть ключа. Этот ключ будет использоваться для подписи пакетов и для удостоверения личности в почте.

Создание SSH и GPG-ключей описано в разделе «Работа с ключами разработчика».

Примечание. Псевдоним – это фактически второе имя человека в команде. Псевдоним лучше выбирать короткий, запоминающийся и не отягощенный мусором. Например, yoda – удобный псевдоним, а travellingwilburys1998 – неудобный. Список уже занятых имен можно посмотреть в пакете alt-gpgkeys.

Заявка на принятие в ALT Linux Team создается кандидатом в Bugzilla (https://bugzilla.altlinux.org/). В данном случае Bugzilla выступает площадкой для вступления в ALT Linux Team. Здесь ведется официальный диалог с кандидатами. Посмотреть прогресс других кандидатов и узнать, какие задачи они взяли можно, по ключевому слову «join».

Примечание. Чтобы сообщать о новых ошибках или оставлять комментарии к существующим в Bugzilla, необходимо иметь учетную запись. Учетная запись позволяет другим пользователям идентифицировать автора, оставившего комментарии к ошибкам или изменившего их состояние. Чтобы зарегистрироваться, достаточно указать адрес электронной почты. По этому адресу будет отправлено сообщение с подтверждением.

Регистрация заявки происходит следующим образом:

- на главной странице ALT Bugzilla выбрать ссылку «Зарегистрировать ошибку» или «Новая ошибка»;
- выбрать раздел «Team Accounts: ALT Linux Team: присоединение»;
- зарегистрировать заявку.

Заявка (баг) должна быть оформлена следующим образом:

- в поле «Компонент» необходимо выбрать «join» (в поле «Продукт» должен быть указан «Team Accounts»);
- поле «Аннотация» предназначено для краткого описания сути ошибки, здесь можно указать ключевое слово «join» с указанием предполагаемой почты: `join user@;`
- значение поля «Серьезность» можно оставить со значением по умолчанию: «normal»;
- платформа определяется автоматически, ее можно изменить, если планируется собирать пакеты для другой платформы (в том числе указать «all» – все платформы);
- в теле заявки (поле «Подробности») нужно указать псевдоним (имя пользователя) нового участника, адрес пересылки почты, имя ментора, а также несколько слов о том, чем кандидат намерен заняться в ALT Linux Team на момент подачи заявки на вступление (например, «собрать для начала такой-то пакет, а потом пакеты из такой-то области», «помочь со сборкой чего-нибудь», «научиться собирать пакеты» и т. п.). От заявленной кандидатом цели могут зависеть, в частности, требования ментора и рецензента к обретаемым кандидатом навыкам и их пристрастие. Пример комментария к открытой заявке на вступление в ALT Linux Team:

```
Псевдоним: sova
Почта: Valentin Sokolov <sova@altlinux.org>
Адрес пересылки почты: mail@mail.com
Имя ментора: Grigory Ustinov
Почта ментора: grenka@altlinux.org

Хочу научиться собирать пакеты.
```

- приложить к заявке публичный SSH-ключ и публичный GPG-ключ в виде отдельных приложений (attachments) обычными файлами. GPG-ключ необходимо приложить в экспортированном виде (`gpg --export --armor <id ключа>`). Файлы можно прикладывать уже после создания заяки. Необходимо убедиться, что экспортирован единственный ключ, а не более (gpg %путь-до-файла%);
- после создания заявки следует добавить e-mail ментора в поле «Подписчики», чтобы он мог должным образом подтвердить свое менторство.

7.3 Обработка заявки

После получения необходимой информации секретарь проверяет приложенные ключи, создает e-mail адрес и выдает ограниченный доступ в git.alt (без возможности сборки пакетов).

Секретарь ведет процесс обработки заявки по регламенту. При переходе на новый этап секретарь обычно указывает номер этапа в открытой кандидатом заявке.

После успешного завершения процедуры «Join» секретарь выдает полный доступ в git.alt. Начиная с этого момента, кандидат становится полноправным членом команды.

7.4 Работа с ключами разработчика

При приеме в Team участник предоставляет два криптографических ключа, по которым он идентифицируется в дальнейшем. Необходимо хранить ключи подписи в недоступном для других людей месте.

При утере одного из ключей участник может заменить его, заверив вторым. При утрате обоих ключей участник обязан незамедлительно известить об этом принимающих. Его доступ в git.alt прекращается до восстановления ключей.

Два ключа могут быть восстановлены либо посредством личной встречи с одним из принимающих, либо посылкой их письмом, заверенным ключом одного из участников ALT Linux Team. В последнем случае всю ответственность за дальнейшую безопасность репозитория несет участник, заверивший ключи.

7.4.1 Создание SSH-ключа

Создать SSH-ключа можно, например, следующей командой:

```
$ ssh-keygen [-t <тип ключа>] [-b <размер ключа (для RSA)>]
```

Рекомендуется указывать тип ключа ED25519 или RSA с размером не менее 4096 бит. Ключ настоятельно рекомендуется сделать с паролем.

Пример создания SSH-ключа:

```
$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/user/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_ed25519.
Your public key has been saved in /home/user/.ssh/id_ed25519.pub.
The key fingerprint is:
SHA256:7gUUwuricYRlF2mQxl4429Y5Dl2xECRSxcX67yPJrdU user@platform
The key's randomart image is:
+--[ED25519 256]--+
|   .o*=**+o.     |
|    O.*o.oo.     |
|   * O o.+.      |
|  . = +.*        |
|   o . oSo       |
```

```
|  o o  ....  .   |
| . +    ...+. E  |
|  .    . .+.+    |
|        . .+..   |
+----[SHA256]-----+
```

На вопрос о файле сохранения ключа можно ничего не отвечать, а принять путь файла ключа по умолчанию, нажав <Enter>. В этом случае публичная часть ключа – файл `~/.ssh/id_ed25519.pub` для ED25519-ключа или `~/.ssh/id_rsa.pub` для RSA-ключа.

Примечание. Файлы `~/.ssh/id_ed25519` и `~/.ssh/id_rsa` – это секретные (закрытые) ключи. Никогда и никому не следует пересылать секретный ключ.

Для упрощения работы с ключами с паролями можно воспользоваться SSH-агентом.

7.4.2 Создание GPG-ключа

Создать новый GPG-ключ можно, выполнив команду:

```
$ gpg --gen-key
```

В процессе ответа на вопросы следует выбрать:

- тип ключа – RSA и RSA;
- размер – не менее 4096 бит;
- срок действия ключа – без ограничения срока действия;
- имя – имя в формате <Имя> <Фамилия>;
- email – псевдоним@altlinux.org (желательно только латинские буквы нижнего регистра);
- комментарий – лучше оставить пустым.

Все входные данные для gpg --gen-key должны быть указаны в ASCII.

Пример создания GPG-ключа:

```
$ gpg --gen-key
Выберите тип ключа:
   (1) RSA и RSA (по умолчанию)
   (2) DSA и Elgamal
   (3) DSA (только для подписи)
   (4) RSA (только для подписи)
Ваш выбор? 1
длина ключей RSA может быть от 1024 до 4096 бит.
Какой размер ключа Вам необходим? (2048) 4096
Запрошенный размер ключа - 4096 бит
Выберите срок действия ключа.
         0 = без ограничения срока действия
      <n>  = срок действия - n дней
      <n>w = срок действия - n недель
```

```
      <n>m = срок действия - n месяцев
      <n>y = срок действия - n лет
Срок действия ключа? (0) 0
Срок действия ключа не ограничен
Все верно? (y/N) y

Для идентификации Вашего ключа необходим ID пользователя. Программа создаст его
из Вашего имени, комментария и адреса электронной почты в виде:
    "Baba Yaga (pensioner) <yaga@deepforest.ru>"

Ваше настоящее имя: Valentin Sokolov
Адрес электронной почты: sova@altlinux.org
Комментарий:
Вы выбрали следующий ID пользователя:
    "Valentin Sokolov <sova@altlinux.org>"

Сменить (N)Имя, (C)Комментарий, (E)адрес или (O)Принять/(Q)Выход? O
Для защиты секретного ключа необходима фраза-пароль.
```

Экспорт публичной части ключа из связки выполнятся командой:

```
$ gpg --armor --export псевдоним@altlinux.org
```

Примечание. Может быть экспортировано несколько ключей с одним uid (email), а требуется один ключ. В подобном случае (например, после редактирования) следует удалить лишние ключи из связки перед экспортом:

```
$ gpg --delete-key старый/ключ
```

Для копирования публичной части ключа в файл можно использовать следующую команду:

```
$ gpg --armor --export псевдоним@altlinux.org > public.key
```

7.4.3 Модификация GPG-ключа

Необходимо убедиться, что тип ключа – RSA, а размер не менее 4096 бит. Добавить идентификатор в ключ можно с помощью команд

```
$ gpg --edit-key псевдоним@altlinux.org
gpg> adduid
```

Далее следует указать имя (в формате <Имя> <Фамилия>) и uid (email) вида псевдоним@altlinux.org, после чего сохранить изменения:

```
gpg > save
```

7.4.4 Обновление GPG-ключа в пакете alt-gpgkeys

Для замены просроченного или недействительного ключа, используемого для подписи пакетов, необходимо клонировать последнюю актуальную сборку пакета alt-gpgkeys.git, обновить в нем свой ключ и выложить модифицированную версию на git.alt в свое личное пространство (private repo) – этим подтверждается аутентичность модификации. Далее необходимо (пере-)открыть заявку на пакет (компонент) alt-gpgkeys для продукта Sisyphus в Bugzilla и ждать реакции мэйнтейнера.

Можно также приложить новый ключ, экспортированный в текстовый файл, к заявке.