

# Руководство по сборке RPM-пакетов для дистрибутивов АЛЬТ

Валентин Соколов, Мария Фоканова и другие.

# Оглавление

Вступление .....	1
PDF Версия .....	1
Структура документации .....	1
Введение в пакетные менеджеры .....	2
Вклад в руководство .....	3
Система управления пакетами. Знакомство с APT .....	4
Установка необходимых пакетов для процесса сборки .....	5
Основные команды RPM .....	6
Программное обеспечение для упаковки .....	11
RPM-пакеты .....	11
Что такое RPM-пакет? .....	11
Инструменты для сборки RPM-пакетов .....	11
Рабочее пространство для сборки RPM-пакетов .....	11
Что такое SPEC-файл? .....	12
Пример .спес-файла .....	12
Инструмент Gear .....	16
Вступление .....	16
Структура репозитория .....	16
Правила экспорта .....	17
Основные типы устройства gear-репозитория .....	18
Быстрый старт Gear .....	18
Создание gear-репозитория путём импорта созданного ранее srpm-пакета .....	18
Создание gear-репозитория на основе готового git-репозитория .....	20
Сборка пакета из gear-репозитория .....	20
Фиксация изменений в репозитории .....	20
Примеры сборки пакетов с использованием инструментов Альт .....	22
Подготовка пространства .....	22
Написание спес файла и правил Gear .....	23
Описание пакета с исходными текстами на C++ .....	26
Join .....	31
Процедура принятия в Team .....	31
Сбор информации .....	31
Создание заявки .....	32
Создание SSH и GPG ключей .....	33
SSH Key .....	33
GPG Key .....	33

# Вступление

Уважаемый читатель! Данное руководство нацелено на подробное изучение сборки пакетов "с нуля". Для того, чтобы следовать ему, убедительная просьба внимательно читать текст, переходить по ссылкам, которые представлены в тексте. В них может быть предоставлена дополнительная информация, раскрывающая те темы, которые кратко описаны в данной документации.

## PDF Версия

Вы также можете скачать [PDF версию данного документа](#).

## Структура документации

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

- Отображение файловой структуры будет представлено следующим образом:

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

- В дальнейшем вывод команд будет продемонстрирован следующим образом:

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

- Темы, представляющие интерес, или словарные термины упоминаются либо как ссылки на соответствующую документацию или веб-сайт выделены **жирным** шрифтом, либо *курсивом*. Первые упоминания некоторых терминов ссылаются на соответствующую документацию.
- Названия утилит, команд и других элементов, обычно встречающихся в коде, написаны **моноширинным** шрифтом.

# Введение в пакетные менеджеры

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

**Различия между представителями семейства RPM выражаются в:**

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

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

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

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

# Вклад в руководство

Вы можете внести свой вклад в это руководство, отправив запрос на принятие изменений (Pull Request) в оригинальный [репозиторий](#).

# Система управления пакетами.

## Знакомство с АРТ

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

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

### ПРИМЕЧАНИЕ

Установка пакетов в АЛЪТ Линукс осуществляется с помощью утилиты АРТ

### ПРИМЕЧАНИЕ

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

- - команды без административных привилегий будут начинаться с символа “\$”
- - команды с административными привилегиями будут начинаться с символа “#”

### ПРИМЕЧАНИЕ

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

```
# control sudowheel enabled
```

# Установка необходимых пакетов для процесса сборки

Чтобы следовать данному руководству, Вам потребуется установить следующие пакеты:

## ПРИМЕЧАНИЕ

Некоторые из этих пакетов устанавливаются по умолчанию в [Альт](#). Установка проводится с правами суперпользователя.

```
# apt-get update
```

```
Получено: 1 http://ftp.altlinux.org p10/branch/x86_64 release [4223B]
Получено: 2 http://ftp.altlinux.org p10/branch/x86_64-i586 release [1665B]
Получено: 3 http://ftp.altlinux.org p10/branch/noarch release [2844B]
Получено 8732B за 0s (81,8kB/s).
Найдено http://ftp.altlinux.org p10/branch/x86_64/classic pkglist
Найдено http://ftp.altlinux.org p10/branch/x86_64/classic release
Найдено http://ftp.altlinux.org p10/branch/x86_64/gostcrypto pkglist
Найдено http://ftp.altlinux.org p10/branch/x86_64/gostcrypto release
Найдено http://ftp.altlinux.org p10/branch/x86_64-i586/classic pkglist
Найдено http://ftp.altlinux.org p10/branch/x86_64-i586/classic release
Найдено http://ftp.altlinux.org p10/branch/noarch/classic pkglist
Найдено http://ftp.altlinux.org p10/branch/noarch/classic release
Чтение списков пакетов... Завершено
Построение дерева зависимостей... Завершено
```

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

# Основные команды RPM

Для ознакомления с данным разделом потребуется пакет. В качестве примера мы будем использовать пакет [Yodl-docs](#).

## Как узнать информацию о RPM-пакете без установки?

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

### ПРИМЕЧАНИЕ

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

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

Вывод:

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

This package contains documentation for Yodl.



## Как установить RPM-пакет?

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

### ПРИМЕЧАНИЕ

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

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

Вывод:

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

## Проверка установки пакета в системе.

```
$ rpm -q () yodl-docs
```

Вывод:

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

## Просмотр файлов пакета, установленного в системе.

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

Вывод:

```
/usr/share/doc/yodl
/usr/share/doc/yodl-doc
/usr/share/doc/yodl-doc/AUTHORS.txt
/usr/share/doc/yodl-doc/CHANGES
/usr/share/doc/yodl-doc/changelog
/usr/share/doc/yodl-doc/yodl.dvi
/usr/share/doc/yodl-doc/yodl.html
/usr/share/doc/yodl-doc/yodl.latex
/usr/share/doc/yodl-doc/yodl.pdf
/usr/share/doc/yodl-doc/yodl.ps
```

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

### Просмотр недавно установленных пакетов.

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

Вывод:

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

### Поиск пакета в системе.

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

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

Вывод:

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

### Проверка файла, относящегося к пакету.

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

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

Вывод:

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

### Вывод информации о пакете.

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

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

Вывод:

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

### Обновление пакета.

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

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

Вывод:

```
Подготовка...
```

##### [100%]

пакет yodl-docs-1:4.03.00-alt2.noarch уже установлен

#### ПРИМЕЧАНИЕ

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

# Программное обеспечение для упаковки

## RPM-пакеты

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

### Что такое RPM-пакет?

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

Существует два типа RPM-пакетов:

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

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

### Инструменты для сборки RPM-пакетов

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

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

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

### Рабочее пространство для сборки RPM-пакетов

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

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

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

Созданные каталоги служат следующим целям:

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

## Что такое СПЕС-файл?

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

## Пример .спес-файла

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

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

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

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

Packager: Sample Packager <sample@altlinux.org>

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

%description
```

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

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

%prep
%setup
%patch0 -p1

%build
%configure
%make_build

%install
%makeinstall_std
%find_lang %name

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

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

## Пункты преамбулы

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

СРЕС Директива	Определение
<b>Name</b>	Базовое имя пакета, которое должно совпадать с именем спес-файла.
<b>Version</b>	Версия upstream-кода.
<b>Release</b>	Релиз пакета используется для указания номера сборки пакета при данной версии upstream-кода. Как правило, установите начальное <b>alt1</b> и увеличивайте его с каждым новым выпуском пакета, например: alt1, alt2, alt3 и т.д. Сбросьте значение до alt1 при создании новой версии программного обеспечения.
<b>Summary</b>	Краткое, в одну строку, описание пакета.
<b>License</b>	Лицензия на собираемое программное обеспечение.

Group	Используется для указания категории, к которой относится пакет. Указанная группа должна находиться в списке групп, известном RPM. Этот список располагается в файле <code>/usr/lib/rpm/GROUPS</code> , идущим вместе с пакетом <code>rpm</code> .
URL	Полный URL-адрес для получения дополнительной информации о программе. Чаще всего это ссылка на <b>GitHub</b> upstream-проекта для собираемого программного обеспечения.
Source0	Путь или URL-адрес к сжатому архиву исходного кода (не исправленный, исправления обрабатываются в другом месте). Этот раздел должен указывать на доступное и надежное хранилище архива, например, на upstream-страницу, а не на локальное хранилище сборщика. При необходимости можно добавить дополнительные исходные директивы, каждый раз увеличивая их количество, например: <code>Source1</code> , <code>Source2</code> , <code>Source3</code> и так далее.
Patch0	Название первого патча, который при необходимости будет применен к исходному коду. При необходимости можно добавить дополнительные директивы <code>PatchX</code> , увеличивая их количество каждый раз, например: <code>Patch1</code> , <code>Patch2</code> , <code>Patch3</code> и так далее.
BuildArch	Если пакет не зависит от архитектуры, например, если он полностью написан на интерпретируемом языке программирования, установите для этого значение <b>BuildArch: noarch</b> . Если этот параметр не задан, пакет автоматически наследует архитектуру компьютера, на котором он собран, например <code>x86_64</code> .
BuildRequires	Разделённый запятыми или пробелами список пакетов, необходимых для сборки программы. Может быть несколько записей <b>BuildRequires</b> , каждая в отдельной строке в SPEC файле.
Requires	Разделённый запятыми или пробелами список пакетов, необходимых программному обеспечению для запуска после установки. Это его <b>зависимости</b> . Может быть несколько записей <b>Requires</b> , каждая в отдельной строке в SPEC файле.
ExcludeArch	Если часть программного обеспечения не может работать на определенной архитектуре процессора, Вы можете исключить эту архитектуру здесь.

Директивы **Name**, **Version** и **Release** содержат имя RPM-пакета. Эти три директивы часто называют **N-V-R** или **NVR**, поскольку имена RPM-пакета имеют формат **NAME-VERSION-RELEASE**.

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

```
$ rpm -q rpmdevtools
```



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

### Составляющие основной части

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

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

### ПРИМЕЧАНИЕ

Конструкция `%setup` в Sisyphus RPM использует флаг `-q` (quiet) по умолчанию. Запись `%setup -q` и `%setup` полностью идентичны. Если использовать конструкцию с флагом `-v`, то будет выведена дополнительная информация в логах сборки

# Инструмент Gear

## Вступление

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.

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 'rpm-пакетов'.

## Структура репозитория

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

### Одна сущность — один репозиторий

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

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

### Несжатый исходный код

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

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

исходного кода.

## Распакованный исходный код

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

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

## Форматированный changelog

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

## Правила экспорта

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

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

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

```
директива: параметры
```

Параметры разделяются пробельными символами.

Директивы позволяют экспортировать:

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

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

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

## Основные типы устройства gear-репозитория

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

### Архив с модифицированным исходным кодом

С помощью простого правила

```
tar: .
```

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

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

позволяет избежать коллизий.

### Архив с немодифицированным исходным кодом и патчем, содержащем локальные изменения

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

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

Такое устройство репозитория получается при использовании утилиты [gear-srpmimport](#), предназначенной для быстрой миграции от srpm-файла к gear-репозиторию.

### Смешанные типы

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

## Быстрый старт Gear

### Создание gear-репозитория путём импорта созданного ранее srpm-пакета.

Пусть у нас есть srpm-пакет [foobar-1.0-alt1.src.rpm](#), и, к примеру, в нём находится

следующее:

```
$ 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 него gear-репозиторий, нам нужно:

1. Создать каталог, в котором будет располагаться наш архив:

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

2. Создать новый git-репозиторий:

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

Получившийся пустой git-репозиторий будет выглядеть примерно следующим образом:

```
$ ls -ldog .*
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.

3. В проекте **gear** есть утилита `gear-srpmimport`, предназначенная для автоматизации импортирования srpm-пакета в git-репозиторий:

```
$ 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
```

4. При необходимости в файл правил можно вносить изменения. Например, можно убрать сжатие исходников (соответствующие изменения следует вносить и в спес-файл).

## Создание gear-репозитория на основе готового git-репозитория

1. Создать и добавить в git-репозиторий спес-файл.
2. Создать и добавить в git-репозиторий файл с правилами .gear/rules.

## Сборка пакета из gear-репозитория

1. Сборка пакета при помощи hasher осуществляется командой gear-hsh:

```
$ gear-hsh
```

2. Чтобы собрать старый пакет, который не содержит определения тега Packager в спес-файле, следует отключить соответствующую проверку:

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

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

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

## Фиксация изменений в репозитории

1. Для того, чтобы сделать commit очередной сборки пакета, имеет смысл воспользоваться утилитой **gear-commit**, которая помогает сформировать список изменений на основе записи в спес-файле:

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

2. Прежде чем сделать первый commit, не забудьте сконфигурировать ваш адрес. Это можно сделать глобально несколькими способами, 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 сконфигурировать адрес можно, прописав соответствующие значения в `.git/config` этого git-репозитория:

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

# Примеры сборки пакетов с использованием инструментов Альт

Для примера сборки пакета будем использовать программу для вывода системных уведомлений о текущей дате и времени. Ссылка на github-репозиторий с исходными текстами программ на языках C++ ([Notification](#)) и Python ([DBusTimer\\_Example](#))

Структура репозитория для данных программ идентична: Главный файл (.cpp или .py) и два юнита systemd (.service и .timer)

Вдаваться в подробности написания кода мы не будем, так как основная цель - сборка пакета, а не разработка приложения.

Файл .timer - юнит systemd, который при истечении заданного времени будет вызывать скрипт .py, который выводит уведомление о дате и времени. После срабатывания таймер снова начинает отсчёт до запуска скрипта.

Файл .service - содержит описание, расположение скрипта .py и интерпретатора, который будет обрабатывать скрипт.

## Подготовка пространства

Первым шагом Вам необходимо клонировать репозиторий в Вашу рабочую директорию, используя команду `git clone` (адрес репозитория `DBusTimer_Example` из ссылки выше):

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

```
Cloning into 'DBusTimer_example'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 5 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

В рабочей директории появится каталог с названием проекта: `DBusTimer_Example`.

Создадим каталог `.gear` и перейдём в него.

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

В каталоге `.gear` создадим два файла: правила для `gear` - `rules` и `spec` файл - `dbustimer.spec`

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



После всех изменений содержание каталога проекта примет следующий вид:

```
DBusTimer_example $ ls -la
итого 28
drwxr-xr-x  4 sova domain users 4096 апр 20 14:20 .
drwxr-xr-x 10 sova domain users 4096 апр 20 14:06 ..
drwxr-xr-x  2 sova domain users 4096 апр 20 14:24 .gear
drwxr-xr-x  8 sova domain users 4096 апр 20 14:06 .git
-rwxr-xr-x  1 sova domain users  413 апр 20 14:06 script_dbus.py
-rw-r--r--  1 sova domain users  186 апр 20 14:06 script_dbus.service
-rw-r--r--  1 sova domain users  106 апр 20 14:06 script_dbus.timer
DBusTimer_example $ ls .gear/
dbustimer.spec  rules
```

## Написание spec файла и правил Gear

Следующим этапом сборки будет написание **spec** файла и правил для gear.

В каталоге .gear откроем файл **rules**. Заполним его следующим содержимымЖ

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

Первая строка указывает, что проект будет упакован в **.tar** архив. Вторая строка указывает путь к расположению **.spec** файла. На этом этапе редактирование **rules** заканчивается.

Перейдём к написанию **.spec** файла.

В заголовке или шапке спек файла находятся секции Name, Version, Release, Summary, License, Group, BuildArch, BuildRequires, Source0.

Заполнив данные секции, заголовок spec файла примет вид:

```
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, содержащая в себе 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. Группа - категория, к которой

относится пакет. Так как это тестовый пакет для примера, выставим группу "Other". BuildRequires - пакеты, необходимые для сборки. Так как исходный код написан на python3, нам необходим пакет `rpm-build-python3` с макросами для сборки скриптов Python. Source0 - путь к архиву с исходниками (%name-%version.tar). На этом заголовок .spec файла заканчивается.

Далее - тело, или основная часть .spec файла. В ней описывается сам процесс сборки и инструкции к преобразованию исходных файлов.

Начнём с 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 `%description` находится краткое описание программы. Секция `%prep` отвечает за подготовку программы к сборке. Макрос `%setup` распаковывает исходный код перед компиляцией.

В секции `%install` описаны инструкции, как установить файлы пакета в систему конечного пользователя.

Вместо того, чтобы писать пути установки файлов вручную, будем использовать предопределённые макросы: `%python3_sitelibdir_noarch` будет раскрываться в путь `/usr/lib/python3/site-packages`. По этому пути будет создан каталог с именем пакета, в который будет помещён файл `script_dbus.py` с правами доступа 755.

Аналогичная операция будет проведена с файлами `script_dbus.timer` и `script_dbus.service`. Они должны быть установлены по пути `/etc/xdg/systemd/user`. Так как макроса, раскрывающегося в данный путь нет, будет использован макрос `%_sysconfdir`, который раскрывается в путь `/etc`.

```
%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/` создаёт каталог `dbustimer` в окружении `buildroot` по пути `/usr/lib/python3/site-packages`

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

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

Секция `%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 `%files` описано, 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`.

Секция `%changelog`. Здесь описаны изменения внесённые в ПО, патчи, изменения методологии сборки

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

После всех манипуляций Ваш `.spec` файл будет 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

```

Сохраним файл и перейдём в основную директорию нашего проекта.

Теперь необходимо добавить созданные нами файлы на отслеживание git. Сделать это можно с помощью команды:

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

После добавление файлов на отслеживание, запустим сборку с помощью инструментов gear и hasher следующей командой:

```
$ gear-hsh --no-sisyphus-check --commit -v
```

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

## Описание пакета с исходными текстами на C++

Ссылка на GitHub репозиторий: [Notification](#).

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

В репозитории находятся следующие файлы:

1. .gear - каталог с правилами gear и .спес файлом
2. Makefile — набор инструкций для программы make, которая собирает данный проект.
3. notify.cpp - исходный код программы
4. notify.service - юнит данной программы для systemd
5. notify.timer - юнит systemd, запускающий вывод уведомления о дате и времени с периодичностью в один час.

В каталоге .gear находятся два файла:

1. rules - правила для упаковки архива для gear
2. notify.спес - файл спецификации для сборки пакета

Остановимся подробнее на этих двух файлах.

Перейдём к содержанию файла **rules**

```
tar: .  
спес: .gear/notify.спес
```

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 .спес файлу с инструкциями по сборке текущего пакета.

```
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 Работа с ключами разработчика.  
  
Создание заявки  
  
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 или "шапке" .spec файла описаны следующие поля:

```

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

```

Стандартная схема Name-Version-Release, содержащая в себе 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. Группа - категория, к которой относится пакет. Так как это тестовый пакет для примера, выставим группу "Other". BuildRequires - пакеты, необходимые для сборки. Так как исходный код написан на c++, нам 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). На этом заголовок .spec файла заканчивается.

Тело .spec файла, или же его основная часть.

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

%prep
%setup -q

%build
%make

%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
```

Секция `%description` - описание того, что делает программа. В данном примере - вывод системного уведомления с датой и временем.

Секция `%prep`. Макрос `%setup` с флагом `-q` распаковывает архив, описанный в секции `Source0`.

В секции `%build` происходит *сборка исходного кода*. Так как в примере присутствует Makefile для автоматизации процесса сборки, то в секции будет указан макрос `%make_build`, использующий Makefile для сборки программы.

Секция `%install`

Здесь происходит эмуляция конечных путей при установке файлов в систему. Мы переносим файл в `buildroot` в те пути, куда файлы будут помещены после установки пакета в систему пользователя. Так как файла три, для каждого пропишем конечный путь:

1. `notify` - скомпилированный бинарный файл. В Unix-подобных системах бинарные файлы располагаются в каталоге `/bin`. `mkdir -p %buildroot/bin` - строка, в которой создаётся каталог `bin` в окружении `buildroot`. Следующая строка - `install -Dm0644 %name %buildroot/bin/` - установка бинарного файла `notify` в каталог `%buildroot/bin/` с разрешениями 644.
2. `%name.timer`, `%name.service` - юниты `systemd`. Данные юниты относятся к пользовательским

и находятся в `/etc/xdg/systemd/user/`. Как и для предыдущего файла, создадим в окружении buildroot каталог `mkdir -p %buildroot%_sysconfdir/xdg/systemd/user/`. В пути использован макрос `%_sysconfdir`, который заменяется путём `/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 какие файлы и директории будут скопированы в систему при установке пакета.

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



# Join

Если Вы считаете, что какого-то пакета в Сизифе не хватает, или какой-то пакет заслуживает большего внимания и готовы заняться им — значит, настало время присоединиться к команде [ALT Linux Team](#).

Процедура Join - это процесс вступления в ALT Linux Team, результатом которого является возможность непосредственно участвовать в разработке репозитория Сизиф. После прохождения Join Вы станете мейнтейнером.

Кто такой мейнтейнер? У каждого пакета в репозитории Сизиф есть один или несколько мейнтейнеров — это те участники ALT Linux Team, которые собирают новые версии пакета, следят за ошибками, которые заводят пользователи в багтрекере и общаются с upstream-разработчиками программы.

## Процедура принятия в Team

Для принятия человека в Team создаётся небольшая команда заинтересованных в этом тимовцев:

1. Секретарь команды. В его обязанность входит отслеживать стадии принятия и выполнять административные действия.
2. Ментор вступающего. Он помогает кандидату со вступлением, отвечает на его вопросы и принимает решение о готовности кандидата.
3. Рецензент работы вступающего. Он проводит независимую оценку готовности вступающего по результатам его работы и подтверждает его готовность.
4. Сам кандидат.

## Сбор информации

Для принятия в [ALT Linux Team](#) необходима следующая информация:

1. Имя ментора — участника команды, имеющего желание помогать в процессе приёма в Team. Менторов можно искать в [списках рассылки](#), канале [Telegram](#), или узнать у Ваших знакомых членов сообщества ALT Linux Team, если таковые есть, готовы ли они стать Вашим ментором.
2. Псевдоним участника (имя пользователя). Выбирается им самим. Имя должно начинаться с буквы, содержать только строчные латинские буквы и цифры, быть не короче трёх символов.

### ПРИМЕЧАНИЕ

Псевдоним — это фактически Ваше второе имя в команде. Так Вас будут называть в глаза и за глаза, по нему на Вас будут ссылаться. Поэтому псевдоним лучше выбирать короткий, запоминающийся и не отягощённый мусором. Например, [yoda](#) — удобный псевдоним, а [travellingwilburys1998](#) — неудобный. Список уже занятых имён можно посмотреть в пакете [alt-gpgkeys](#).

1. Адрес почты, на который будет производиться пересылка с адреса `псевдоним@altlinux.org`;
2. SSH-ключ (ED25519 или RSA >= 4096bit). Принимающему нужна публичная часть ключа. Этот ключ будет использоваться для SSH-доступа на ресурсы Sisyphus ([git.alt](https://git.altlinux.org) и другие);
3. GPG-ключ (RSA >= 4096bit). В ключе должны быть 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.

Если у Вас ещё нет SSH- или GPG-ключа, прочтите статью [Работа с ключами разработчика](#).

## Создание заявки

Когда Вы нашли своего куратора процедуры join - ментора, Вам необходимо зафиксировать начало вашей работы. На этом этапе вам понадобится [BugZilla](#). В данном случае bugzilla выступает площадкой для процедуры вступления в ALT Linux Team. Здесь ведётся официальный диалог с участниками, Вы можете посмотреть прогресс других кандидатов, узнать, какие задачи они взяли. Введите в поиске [join](#) и увидите список, относящийся к этой тематике.

Для начала необходимо пройти регистрацию на [BugZilla](#).

После того, как Вы прошли регистрацию и нашли ментора, можно начинать регистрировать заявку. Регистрация заявки происходит следующим образом: на главной странице сайта ALT Bugzilla выберите "[Зарегистрировать ошибку](#)" или "[Новая ошибка](#)". Будущие участники ALT Linux Team регистрируются в разделе [Team Accounts: ALT Linux Team: присоединение](#). При наведении мыши будут видны подсказки к заполнению полей формы. Клик по любому из пунктов формы приведёт Вас на страницу [Поля ошибок](#) с описанием терминов. Выберите компонент *Join*. Если Вы посмотрите другие компоненты, то убедитесь, что они не так популярны. При выборе уровня важности задачи учитывайте, что майнтейнеры просматривают ошибки в том числе по этому фильтру. Также будет видно, к какому разделу относится ваша задача. (Важность определяете Вы сами, исходя из своих возможностей и целеполагания. Скорее, это параметр для Вас.) Поставим здесь средний приоритет **P3**. Платформа определяется автоматически. Здесь же Вы можете изменить значения, в том числе поставить all, и научиться собирать пакеты для любых платформ.

Отметим новое состояние нашей заявки.

Поля *Исполнитель* и *Приёмка* заполнятся автоматически. В пункте *Подписчики* укажем всех, кто заинтересован в решении этой задачи. В случае join - это ментор.

В стандартной форме bugzilla есть поле для заполнения адреса URL, с которым может быть связана ошибка. Так как мы регистрируем по сути наше участие в прохождении Join, отметьте страницу на вики [Join](#)

Поле Аннотация предназначено для краткого описания сути ошибки. Так как мы заводим эту ошибку в рамках вступления в ALT Linux Team, напомним join с указанием предполагаемой почты: `join user@`.

Пример комментария к открытой заявке на вступление в ALT Linux Team.

```
Псевдоним: sova
Почта: Valentin Sokolov <sova@altlinux.org>
Адрес пересылки почты: sokolovvaly.158@gmail.com
Имя ментора: Grigory Ustinov
Почта ментора: grenka@altlinux.org
```

Хочу научиться собирать пакеты.

На данном этапе необходимо прикрепить SSH и GPG ключи. О том, как их создать и прикрепить к заявке - читайте ниже.

## Создание SSH и GPG ключей

### SSH Key

Для того, чтобы создать SSH ключ, можно воспользоваться командой:

```
$ ssh-keygen -t ed25519
```

На вопрос о файле сохранения ключа можно ничего не отвечать, а просто нажать Enter, чтобы принять путь файла ключа по умолчанию. Далее будет задан вопрос о пароле к ключу. Укажите придуманный пароль для защиты ключа.

В каталоге `~/.ssh` появятся файлы:

- `id_ed25519` - секретный ключ
- `id_ed25519.pub` - публичный ключ

**ПРИМЕЧАНИЕ** | НИКОГДА И НИКОМУ НЕ ПЕРЕСЫЛАЙТЕ СЕКРЕТНЫЙ КЛЮЧ

### GPG Key

Для того, чтобы создать GPG-ключ, можно воспользоваться командой:

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

В процессе ответа на вопросы выберете тип ключа `RSA` и `RSA`, длина ключа должна быть не менее 4096 bit, поэтому впишем 4096. Срок действия ключа - без ограничения срока действия. 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`.

Теперь экспортируем публичную часть ключа для отправки, используя команду:

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

Для копирования публичного ключа в файл Вы можете использовать следующую команду:

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

Экспортировать приватную часть ключа можно с помощью команды:

```
$ gpg --armor --export-secret-keys псевдоним@altlinux.org >>private.key
```

#### ПРИМЕЧАНИЕ

**ВНИМАНИЕ** Никогда и ни при каких условиях никому не сообщайте и не присылайте Ваши ПРИВАТНЫЕ ключи. Сохраните Ваши приватные и публичные ключи на отдельном носителе для лучшей сохранности.

Для более подробного изучения темы создания и редактирования ключей рекомендуется перейти по ссылке: [Работа с ключами разработчика](#).

После того, как Вы создали ключи, загрузите их публичные части в комментарий к Вашей заявке. На странице заявки есть отдельный пункт **Приложить файл**.

Поздравляем! Вы создали заявку на вступление в ALT Linux Team. Её закрытие будет означать либо успешное прохождение процедуры и получение доступов к работе с Сизифом и вступление в Team, либо отказ в нём.

Следующие шаги по прохождению процедуры Join будут описаны позже, на данный момент документация находится на стадии разработки.