Альт Платформа

ДОКУМЕНТАЦИЯ ALT LINUX TEAM

Альт Платформа

Практикум

Редакция октябрь, 2025

Георгий Курячий
george@basealt.ru

Степан Мальчевский
usamstudent21@gmail.com

Аннотация

Названия компаний и продуктов, встречающихся в руководстве, могут являться торговыми знаками соответствующих компаний.

Данное руководство предлагает набор практических задач по сборке пакетов. Описанные лабораторные работы являются документацией по разработке с использованием дистрибутива Альт Платформа.

Глава 1. Введение

Для выполнения работ требуются инструменты для сборки RPM-пакетов — **`hasher`** и **`gear`**.

Перед началом работы необходимо:

1. Установить любую версию дистрибутива Альт.
2. Установить пакеты *hasher* и *gear*:

```
# apt-get install hasher gear
```

Все примеры выполнены с использованием дистрибутива Альт Платформа на базе ветки p11.

Глава 2. Пакетная разработка на Альт с помощью Hasher

Итогом в рамках разработки под Linux является **пакет**. Пакетом называются ресурсы, необходимые для установки и интеграции в систему некоторого компонента (архив файлов, до- и послеустановочные сценарии, информация о пакете и его сопровождающем), объединённые в одном файле. Основной материал курса направлен на изучение парадигм и технологий локальной разработки. Лабораторные работы будут объединять полученные знания в рамках сборки пакетов программ. В линейке дистрибутивов Альт используется формат пакетов RPM. Различают *.rpm*-пакеты, непосредственно предназначенные для установки компонентов в систему, и *.src.rpm*-пакеты, содержащие все необходимые для сборки *.rpm*-пакета данные — исходные тексты и сценарии сборки.

2.1. Hasher by ALT Linux Team

В рамках разработки на Альт Платформе используется специальный инструмент Hasher. Это средство безопасной и воспроизводимой сборки пакетов в «чистой» и контролируемой среде. Hasher создает изолированное пространство (файловая система, процессы, отсутствие доступа к сети по умолчанию и т. п.), устанавливает туда базовую систему и все сборочные зависимости, после чего запускает сборку пакета. Главная особенность hasher — возможность запуска в нем непроверенного и потенциально небезопасного кода без какого-либо воздействия на внешнюю систему. В сборочном окружении hasher нет суперпользователя (его заменяет системный пользователь с обычными правами и библиотека **fakeroot**), все процессы запускаются либо с его правами, либо с правами второго системного пользователя, и ни один из этих процессов не имеет доступа к ресурсам внешней системы. Однако для сборки пакетов предоставляется некоторая

информация извне (например, объём памяти и количество ядер процессора) и некоторые свойства ОС (например, работа с псевдотерминалами, поддержка сети и т. п.). Скорость работы приложений в hasher не снижается, но каждый раз среда разворачивается заново, чтобы исключить влияние предыдущей сборки на последующую.

Некоторые следствия данной парадигмы работы hasher:

- все необходимые для сборки зависимости должны быть установлены заранее, либо взяты из *.src.rpm*-пакета;
- сборка не зависит от конфигурации компьютера пользователя, собирающего пакет (установленное, система и их версии), 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.

2.1.1. Настройка hasher

Ещё одной особенностью hasher является независимость от прав суперпользователя. Все действия внутри системы выполняются в изолированном блоке файловой системы, а действия, требующие прав суперпользователя, журналируются и выполняются фиктивно в изолированном блоке.

Для работы hasher необходимо зарегистрировать пользователя, который будет выполнять сборки, чтобы на его основе создать специальных внутренних пользователей для работы.

Регистрация выполняется суперпользователем с помощью команды **`hasher-useradd`**:

```
[root@VM ~]# id user
uid=1000(user) gid=1000(user) группы=1000(user),10(wheel),100(users),36(vmusers)

[root@VM ~]# hasher-useradd user
useradd: Warning: missing or non-executable shell '/dev/null'
useradd: Warning: missing or non-executable shell '/dev/null'
Добавление пользователя user в группу user_a
Добавление пользователя user в группу user_b
Добавление пользователя user в группу hashman
hasher-useradd: enabling hasher-privd
Внимание: Отправляется запрос 'systemctl enable hasher-privd.service'.
Synchronizing state of hasher-privd.service with SysV service script with /usr/
lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable hasher-privd hasher-
useradd: starting hasher-privd

[root@VM ~]# id user
uid=1000(user) gid=1000(user)
группы=1000(user),10(wheel),100(users),997(hashman),1001(user_a),1002(user_b),36(
vmusers)
[root@VM ~]#
```

При регистрации создаются два дополнительных пользователя. **B**-пользователь отвечает за непосредственно сборку пакета, **A**-пользователь выполняет роль суперпользователя в рамках системы сборки (при помощи **fakeroot**).

2.1.2. Первый запуск

Все действия, связанные с работой hasher, контролируются множеством утилит семейства **hsh**.

Перед каждой сборкой нового пакета необходимо пересоздавать окружение, сделать это можно с помощью ключа ***--init***. Также окружение автоматически пересоздаётся при открытии архива исходников пакета (*.src.rpm*-файлы). При первом создании окружения необходимо отдельно создать директорию для расположения изолированного блока файловой системы. По умолчанию инструмент ожидает директорию **~/hasher/**, однако она может быть любой из разрешённых в файле **/etc/hasher-priv/system** (ключ ***prefix=***), в таком случае необходимо одним из параметров передавать путь к расположению директории:

```
[user@VM ~]$ hsh --init
/usr/bin/hsh-sh-functions: строка 281: cd: /home/user/hasher: Нет такого файла
или каталога
[user@VM ~]$ mkdir hasher
[user@VM ~]$ hsh -v --init |& tee log
```

Рассмотрим вывод при создании окружения и обсудим некоторые составные части hasher:

```
hsh: Executing wrapper: systemd-run  --user --scope --same-dir --
property=Delegate=yes --send-sighup --collect
Running as unit: run-p8052-i8352.scope; invocation ID:
4eed2a0919cc4c27bb1bc0b063dc7735
hsh: changed working directory to `/home/papillon_jaune/hasher'
hsh: Locked working directory `/home/papillon_jaune/hasher'

<...>

Чтение списков пакетов...
Построение дерева зависимостей...
Selected version fakeroot#1.29-alt3:p11+348779.600.1.1@1716502783 for
fakeroot>=0:0.7.3
Следующие дополнительные пакеты будут установлены:
 bash       getopt            libelf        libpopt             sh
 bash5      glibc-core        libgcc1       libreadline8        sh5
 bashrc     glibc-preinstall  libgcrypt20   librpm7             terminfo
 bzlib      grep              libgmp10      libselinux          zlib
 coreutils  libacl            libgpg-error  libsha1detectcoll1
 fakeroot   libattr           liblua5.3     libtinfo6
 findutils  libcap            liblzma       libzstd
 gawk       libdb4.7          libpcre2      sed
Следующие НОВЫЕ пакеты будут установлены:
 bash        gawk              libdb4.7      libpcre2            rpm
 bash5       getopt            libelf        libpopt             sed
 bashrc      glibc-core        libgcc1       libreadline8        setup
 bzlib       glibc-preinstall  libgcrypt20   librpm7             sh
 coreutils   grep              libgmp10      libselinux          sh5
 fakeroot    libacl            libgpg-error  libsha1detectcoll1  terminfo
 filesystem  libattr           liblua5.3     libtinfo6           zlib
 findutils   libcap            liblzma       libzstd
<...>
Завершено.

hsh-initroot: Calculated package file list.
hsh-initroot: Generated initial package file list.
Чтение списков пакетов...
Построение дерева зависимостей...
```

```
<...>
Следующие дополнительные пакеты будут установлены:
 alt-os-release                        libmount
 autoconf                              libmpc3
 autoconf-common                       libmpfr6
 autoconf_2.71                         libncursesw6
 automake                              libpam0
 automake-common                       libpasswdqc
 automake_1.16                         libpcre2
 bash                                  libpopt
 bash5                                 libproc2_1
 bashrc                                libreadline8
 binutils                              librpm
 bison                                 librpm7
 bison-runtime                         librpmbuild
 branding-xalt-kworkstation-release    librpmbuild7
 bzip2                                 libseccomp
 bzlib                                 libselinux
 chkconfig                             libsframe1
 common-licenses                       libsha1detectcoll1
 control                               libshell
 coreutils                             libsmartcols
 cpio                                  libstdc++6
 cpp                                   libtcb
 cpp13                                 libtic6
 debugedit                             libtinfo6
 diffutils                             libtool
 elfutils                              libtool-common
 emacs-base                            libtool_2.4
 etcskel                               libtsan2
 file                                  libubsan1
 filesystem                            libudev1
 findutils                             libunistring2
 gawk                                  libuuid
 gcc                                   libvtv0
 gcc-common                            libxml2
 gcc13                                 libzio
 getopt                                libzstd
 gettext                               m4
 gettext-tools                         make
 glib2                                 nss_tcb
 glib2-locales                         pam
 glibc                                 pam-config
 glibc-core                            pam-config-control
 glibc-devel                           pam0_mktemp
 glibc-gconv-modules                   pam0_passwdqc
 glibc-kernheaders                     pam0_tcb
 glibc-kernheaders-generic             pam0_userpass
 glibc-kernheaders-x86                 passwdqc-control
 glibc-locales                         patch
 glibc-nss                             perl-CPAN-Meta-Requirements
 glibc-preinstall                      perl-base
 glibc-pthread                         perl-parent
 glibc-timezones                       perl-threads
 glibc-utils                           pkg-config
 gnu-config                            procps
 grep                                  psmisc
 gzip                                  rootfiles
 iconv                                 rpm
```

```
info-install                            rpm-build
kernel-headers-common                   rpm-build-file
libacl                                  rpm-build-perl
libasan8                                rpm-macros-python
libasm                                  rpm-macros-python3
libatomic1                              rpm-macros-systemd
libattr                                 rpmspec
libaudit1                               sed
libbeecrypt7                            service
libblkid                                setarch
libcap                                  setup
libcap-ng                               sh
libcap-utils                            sh5
libcrypt                                shadow-convert
libcrypt-devel                          shadow-utils
libctf-nobfd0                           sisyphus_check
libdb4.7                                sysvinit-utils
libdw                                   tar
libelf                                  tcb-utils
libffi8                                 terminfo
libgcc1                                 termutils
libgcrypt20                             tzdata
libgmp10                                util-linux
libgpg-error                            util-linux-control
libgpm                                  vim-minimal
libhwasan0                              vitmp
libitm1                                 which
liblsan0                                xml-common
liblua5.3                               xz
liblzma                                 zlib
libmagic                                zstd

Следующие НОВЫЕ пакеты будут установлены:
 alt-os-release                          libmount
 autoconf                                libmpc3
 autoconf-common                         libmpfr6
 autoconf_2.71                           libncursesw6
 automake                                libpam0
 automake-common                         libpasswdqc
 automake_1.16                           libpcre2
 basesystem                              libpopt
 bash                                    libproc2_1
 bash5                                   libreadline8
 bashrc                                  librpm
 binutils                                librpm7
 bison                                   librpmbuild
 bison-runtime                           librpmbuild7
 branding-xalt-kworkstation-release      libseccomp
 bzip2                                   libselinux
 bzlib                                   libsframe1
 chkconfig                               libsha1detectcoll1
 common-licenses                         libshell
 control                                 libsmartcols
 coreutils                               libstdc++6
 cpio                                    libtcb
 cpp                                     libtic6
 cpp13                                   libtinfo6
 debugedit                               libtool
 diffutils                               libtool-common
```

```
elfutils
emacs-base
etcskel
file
filesystem
findutils
gawk
gcc
gcc-common
gcc13
getopt
gettext
gettext-tools
glib2
glib2-locales
glibc
glibc-core
glibc-devel
glibc-gconv-modules
glibc-kernheaders
glibc-kernheaders-generic
glibc-kernheaders-x86
glibc-locales
glibc-nss
glibc-preinstall
glibc-pthread
glibc-timezones
glibc-utils
gnu-config
grep
gzip
iconv
info-install
kernel-headers-common
libacl
libasan8
libasm
libatomic1
libattr
libaudit1
libbeecrypt7
libblkid
libcap
libcap-ng
libcap-utils
libcrypt
libcrypt-devel
libctf-nobfd0
libdb4.7
libdw
libelf
libffi8
libgcc1
libgcrypt20
libgmp10
libgpg-error
libgpm
libhwasan0
libitm1
libtool_2.4
libtsan2
libubsan1
libudev1
libunistring2
libuuid
libvtv0
libxml2
libzio
libzstd
m4
make
nss_tcb
pam
pam-config
pam-config-control
pam0_mktemp
pam0_passwdqc
pam0_tcb
pam0_userpass
passwdqc-control
patch
perl-CPAN-Meta-Requirements
perl-base
perl-parent
perl-threads
pkg-config
procps
psmisc
rootfiles
rpm
rpm-build
rpm-build-file
rpm-build-perl
rpm-macros-python
rpm-macros-python3
rpm-macros-systemd
rpmspec
sed
service
setarch
setup
sh
sh5
shadow-convert
shadow-utils
sisyphus_check
sysvinit-utils
tar
tcb-utils
terminfo
termutils
time
tzdata
util-linux
util-linux-control
vim-minimal
vitmp
which
```

```
  liblsan0                          xml-common
  liblua5.3                         xz
  liblzma                           zlib
  libmagic                          zstd
0 будет обновлено, 178 новых установлено, 0 пакетов будет удалено и 0 не будет
обновлено.
Необходимо получить 73,3MB/86,2MB архивов.
После распаковки потребуется дополнительно 514MB дискового пространства.
<...>
Завершено.

Running /usr/lib/rpm/posttrans-filetriggers hsh-initroot: RPM database updated.
<86>Jul  1 09:29:43 groupadd[10102]: group added to /etc/group: name=caller,
GID=1000^M
<86>Jul  1 09:29:43 groupadd[10102]: group added to /etc/gshadow: name=caller^M
<86>Jul  1 09:29:43 groupadd[10102]: new group: name=caller, GID=1000^M
<86>Jul  1 09:29:43 useradd[10108]: new user: name=caller, UID=1000, GID=1000,
home=/, shell=/bin/bash, from=none^M
<86>Jul  1 09:29:43 groupadd[10117]: group added to /etc/group: name=rooter,
GID=1001^M
<86>Jul  1 09:29:43 groupadd[10117]: group added to /etc/gshadow: name=rooter^M
<86>Jul  1 09:29:43 groupadd[10117]: new group: name=rooter, GID=1001^M
<86>Jul  1 09:29:43 useradd[10123]: new user: name=rooter, UID=1001, GID=1001,
home=/root, shell=/bin/bash, from=none^M
<86>Jul  1 09:29:43 groupadd[10132]: group added to /etc/group: name=builder,
GID=1002^M
<86>Jul  1 09:29:43 groupadd[10132]: group added to /etc/gshadow: name=builder^M
<86>Jul  1 09:29:43 groupadd[10132]: new group: name=builder, GID=1002^M
<86>Jul  1 09:29:43 useradd[10138]: new user: name=builder, UID=1002, GID=1002,
home=/usr/src, shell=/bin/bash, from=none^M
mode of '/usr/src' changed from 0755 (rwxr-xr-x) to 1777 (rwxrwxrwt)
hsh-initroot: First time initialization complete.
hsh-initroot: RPM database archivation complete.
hsh-initroot: Chroot archivation complete.

mkdir: created directory '/usr/src/tmp'
mkdir: created directory '/usr/src/RPM'
mkdir: created directory '/usr/src/RPM/BUILD'
mkdir: created directory '/usr/src/RPM/SOURCES'
mkdir: created directory '/usr/src/RPM/SPECS'
mkdir: created directory '/usr/src/RPM/SRPMS'
mkdir: created directory '/usr/src/RPM/RPMS'
mkdir: created directory '/usr/src/RPM/RPMS/noarch'
hsh-initroot: Created RPM build directory tree.
```

В состав hasher по умолчанию идёт набор утилит, связанных с работой самого инструмента (в частности, **fakeroot**, за счёт которого реализован псевдосуперпользователь). Также по умолчанию hasher устанавливает классический набор утилит для разработки, в том числе использующихся в рамках данного курса (**autoconf**, **automake**, **gcc**, **diffutils** и т. д.).

Внутри hasher основной пользователь и два дополнительных обретают описанные выше характеристики: основной пользователь (внутри он называется **caller**) имеет право лишь просматривать данные и получать доступ к готовым пакетам, **A**-пользователь (@**rooter**) выполняет действия суперпользователя в рамках инструмента, **B**-пользователь (@**builder**) осуществляет сборку пакетов.

Hasher предоставляет **минимальное** необходимое для сборки окружение. Вследствие этого все инструменты по разработке или отладке, требуемые в рамках работы с hasher, необходимо устанавливать после **каждого пересоздания** окружения. Всюду далее при рассмотрении отдельных сборок будут опускаться команды пересоздания окружения и установки постоянно необходимых для разработки утилит:

- **`hsh --init`** для пересоздания окружения;
- **`hsh-install`** для установки пакетов из репозиториев Альт.

Например:

```
[user@VM ~]$ hsh --init
<...>
[user@VM ~]$ hsh-install vim-console tree
<...>
[user@VM ~]$
```

Также будут опускаться команды входа в изолированное окружение от имени @**builder** и @**rooter**, все кодовые вставки будут сопровождаться отметками о том, от имени какого пользователя проведён вход:

- **`hsh-shell`** для входа от имени @**builder**;
- **`hsh-shell --rooter`** для входа от имени @**rooter**.

2.1.3. Создание нулевого пакета

Разберём структуру hasher согласно правилам разработки RPM-пакетов. RPM-пакет состоит из архива файлов, а также заголовка, содержащего метаданные о пакете. Различают **пакеты с исходным кодом** (*.src.rpm*), состоящих из исходников и spec-файла, представляющего из себя инструкцию по сборке пакета, и собственно **пакеты** (*.rpm*), непосредственно устанавливающиеся в систему.

Hasher для работы содержит специальное дерево директорий, по которому автоматически или вручную распределяются файлы для сборки:

@**builder**

```
[builder@localhost ~]$ tree RPM
RPM
├── BUILD
├── RPMS
│   └── noarch
├── SOURCES
├── SPECS
└── SRPMS

7 directories, 0 files
```

Директории **`RPMS/`** и **`SRPMS/`** содержат готовые пакеты, **`SOURCES/`** содержит исходные файлы. Исходники чаще всего хранятся в виде архива (обычно **`.tar.gz`**). Директория **`SPECS/`** хранит соответствующий **`.spec`**-файл. В директории **`BUILD/`** проводится сборка пакета, перед началом сборки содержимое директории очищается, что позволяет повторно проводить независимую сборку для одного и того же пакета.

Напишем самый простой пакет, не содержащий ничего кроме spec-файла:

@**builder**: **`RPM/SPECS/null-pkg.spec`**

```
Name: null-pkg
Version: 1.0
Release: alt1

Summary: Null package

License: GPL-3.0-or-later
Group: Development/Other


%description
This is the smallest ever alt package without any functionality

%files

%changelog
* Tue Jul 01 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial build
```

@**builder**

```
[builder@localhost ~]$ tree RPM
RPM
├── BUILD
├── RPMS
│   └── noarch
├── SOURCES
├── SPECS
│   └── null-pkg.spec
└── SRPMS

7 directories, 1 file
[builder@localhost ~]$
```

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 необходима для сборки и идентификации пакета.

Spec состоит из **преамбулы**, содержащей метаданные пакета, и **основной части**, содержащей инструкции по сборке и установке пакета.

Минимально необходимые параметры spec-файла для сборки пакета:

- **преамбула:**
  - имя пакета;
  - версия согласно правилам версионирования;
  - релиз пакета;
  - краткое описание пакета;
  - лицензия на собираемое ПО;

- категория пакета (это поле морально устарело, но в ALT продолжает использоваться);

- **основная часть:**

  - директива **`%files`** для описания устанавливаемых файлов у конечного пользователя (даже если этих файлов нет);

  - директива **`%changelog`** для записи изменений, произошедших в пакете между сборками разных версий или релизов.

Подробнее познакомиться с другими директивами можно по ссылке. В рамках лабораторных работ в дальнейшем будут рассмотрены и другие директивы (например, необязательная директива **`%description`**, содержащая более подробное описание функциональности пакета).

Сборка пакетов осуществляется с помощью команды **`rpmbuild`**. Ключ **`-ba`** (build all) собирает как двоичный пакет, так и новый пакет с исходным кодом:

@builder

```
[builder@localhost ~]$ rpmbuild -ba RPM/SPECS/null-pkg.spec
Processing files: null-pkg-1.0-alt1
Wrote: /usr/src/RPM/SRPMS/null-pkg-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/null-pkg-1.0-alt1.x86_64.rpm (w2.lzdio)

[builder@localhost ~]$ tree RPM
RPM
├── BUILD
├── RPMS
│   ├── noarch
│   └── x86_64
│       └── null-pkg-1.0-alt1.x86_64.rpm
├── SOURCES
├── SPECS
│   └── null-pkg.spec
└── SRPMS
    └── null-pkg-1.0-alt1.src.rpm

8 directories, 3 files
[builder@localhost ~]$
```

Попробуем установить полученный пакет в hasher, для этого необходимо от имени суперпользователя воспользоваться установщиком *rpm* с ключом **`-i`**:

@user

```
[user@VM ~]$ hsh-shell --rooter
```

@rooter

```
[root@localhost .in]# rpm -i /usr/src/RPM/RPMS/x86_64/null-pkg-1.0-
alt1.x86_64.rpm
<13>Jul  1 16:46:01 rpm: null-pkg-1.0-alt1 1751388308 installed

[root@localhost .in]#
```

```
[root@localhost .in]# which null-pkg
which: no null-pkg in (/root/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/sbin:/
usr/local/bin:/usr/games)
[root@localhost .in]#
```

Пакет успешно установился, и, поскольку он пустой, он никак не отображается среди других программ (потому что никаких исполняемых программ в нем нет).

2.1.4. Создание примитивного пакета

Теперь соберём пакет, состоящий из одного **`shell`**-сценария, чтобы проверить реальную функциональность пакета. Напомним, что необходимо пересоздать окружение, а также добавить те инструменты разработки, которыми вы пользуетесь:

@builder

```
[builder@localhost ~]$ tree RPM
RPM
├── BUILD
├── RPMS
│   └── noarch
├── SOURCES
├── SPECS
└── SRPMS
```

@builder: **`RPM/SPECS/not-null-pkg.spec`**

```
Name: not-null-pkg
Version: 1.0
Release: alt1

Summary: Not Null package

License: GPL-3.0-or-later
Group: Development/Other

Source: %name-%version.sh

%description
This is not the smallest ever alt package cause of functionality

%install
install -D -pm 755 %SOURCE0 %buildroot%_bindir/%name

%files
%_bindir/*

%changelog
* Tue Jul 08 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial build
```

@builder: **`RPM/SOURCES/not-null-pkg-1.0.sh`**

```
echo "This is not null pkg"
```

Разберём новые директивы и команды.

Обычно при сборке RPM каталог с исходными текстами программы упакован в **.tar.gz**-архив, в названии которого встречается имя программы и её версия — их удобно сразу заменять на соответствующие макросы. Мы же для упрощения укажем исключительно один скрипт, сохранив правила именования.

В основной части spec-файла добавилась директива ***%install***, которая описывает команды установки/копирования файлов из сборочного каталога в псевдокорневой каталог. Утилита **install** занимается размещением всех файлов, которые должны входить в пакет (исполняемых файлов, документации, библиотек; в нашем случае — исполняемого скрипта), по их конечным директориям. При этом используются предопределённые макросы, описывающие место установки данных. Все исходные файлы размещаются в каталоге **RPM/SOURCES/**. Явно к объекту, указанному в директиве ***Source*** (или ***Source0***) можно обращаться через макрос ***%SOURCE0***, ***Source1***: — ***%SOURCE1*** и т. д.

Для сборки пакета не нужны (в ALT — **запрещены**) права суперпользователя. Во время сборки файлы устанавливаются в псевдокорневой каталог (как правило, **/usr/src/tmp/_имя-пакета_-buildroot/;** он обозначается макросом ***%buildroot***). Сценарий попадает в поддиректорию **/usr/bin**.

В директиве ***%files*** указывается расположение итоговых данных. В нашем случае в итоговый пакет должен попасть исполняемый файл.

Запустим сборку пакета и соберём информацию о сборке:

@builder

```
builder@localhost ~]$ rpmbuild -ba RPM/SPECS/not-null-pkg.spec >& log
```

- создание каталога сборки (в примере он остаётся пустым) и удаление старого псевдокорневого каталога:

```
Executing(%install): /bin/sh -e /usr/src/tmp/rpm-tmp.81856
+ umask 022
+ /bin/mkdir -p /usr/src/RPM/BUILD
+ cd /usr/src/RPM/BUILD
+ /bin/chmod -Rf u+rwX -- /usr/src/tmp/not-null-pkg-buildroot
+ /bin/rm -rf -- /usr/src/tmp/not-null-pkg-buildroot
```

- установка файлов пакета (в примере это единственный сценарий):

```
+ PATH=/usr/libexec/rpm-build:/usr/src/bin:/usr/bin:/bin:/usr/local/bin:/usr/
games
+ install -D -pm 755 /usr/src/RPM/SOURCES/not-null-pkg-1.0.sh /usr/src/tmp/
not-null-pkg-buildroot/usr/bin/not-null-pkg
```

- проверка соответствия файлов пакета Build ALT Policy (дисциплине сборки пакетов ALT):

```
+ /usr/lib/rpm/brp-alt
Cleaning files in /usr/src/tmp/not-null-pkg-buildroot (auto)
Verifying and fixing files in /usr/src/tmp/not-null-pkg-buildroot
(binconfig,pkgconfig,libtool,desktop,gnuconfig)
Checking contents of files in /usr/src/tmp/not-null-pkg-buildroot/ (default)
Compressing files in /usr/src/tmp/not-null-pkg-buildroot (auto)
Verifying ELF objects in /usr/src/tmp/not-null-pkg-buildroot
```

```
(arch=normal,fhs=normal,lfs=relaxed,lint=relaxed,rpath=normal,stack=normal,tex
trel=normal,unresolved=normal)
Splitting links to aliased files under /{,s}bin in /usr/src/tmp/not-null-pkg-
buildroot
```

- определение эксплуатационных зависимостей пакета, а также предоставляемых им зависимостей и возможных отладочных компонентов:

```
Processing files: not-null-pkg-1.0-alt1
Finding Provides (using /usr/lib/rpm/find-provides)
Executing: /bin/sh -e /usr/src/tmp/rpm-tmp.QrK1pE
find-provides: running scripts
(alternatives,debuginfo,lib,pam,perl,pkgconfig,python,python3,shell)
Finding Requires (using /usr/lib/rpm/find-requires)
Executing: /bin/sh -e /usr/src/tmp/rpm-tmp.aiR5Fd
find-requires: running scripts
(cpp,debuginfo,files,lib,pam,perl,pkgconfig,pkgconfiglib,python,python3,
rpmlib,shebang,shell,static,symlinks,systemd-services)
Finding debuginfo files (using /usr/lib/rpm/find-debuginfo-files)
```

- сборка *.prm* и *.src.rpm*:

```
Executing: /bin/sh -e /usr/src/tmp/rpm-tmp.CsoTgW

Wrote: /usr/src/RPM/SRPMS/not-null-pkg-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/not-null-pkg-1.0-alt1.x86_64.rpm (w2.lzdio)
```

Результат:

```
[builder@localhost ~]$ tree RPM/
RPM/
├── BUILD
├── RPMS
│   ├── noarch
│   └── x86_64
│       └── not-null-pkg-1.0-alt1.x86_64.rpm
├── SOURCES
│   └── not-null-pkg-1.0.sh
├── SPECS
│   └── not-null-pkg.spec
└── SRPMS
    └── not-null-pkg-1.0-alt1.src.rpm

8 directories, 5 files
[builder@localhost ~]$
```

После сборки пакета в **/tmp** некоторое время хранятся данные с последней сборки:

```
[builder@localhost ~]$ tree -A tmp
tmp
└── not-null-pkg-buildroot
    └── usr
        └── bin
            └── not-null-pkg

4 directories, 1 file
```

В данных двоичного пакета размещаются описание составляющих его файлов, а также зависимости пакета:

```
[builder@localhost ~]$ rpmquery --list --package RPM/RPMS/x86_64/not-null-pkg-1.0-alt1.x86_64.rpm
/usr/bin/not-null-pkg
[builder@localhost ~]$ rpmquery --requires --package RPM/RPMS/x86_64/not-null-pkg-1.0-alt1.x86_64.rpm
rpmlib(PayloadIsLzma)
[builder@localhost ~]$
```

При установке пакета в систему файлы раскладываются по соответствующим поддиректориям корневого каталога. Поскольку скрипт исполняемый и лежит в **`/usr/bin`**, а эта директория входит в стандартный **`$PATH`**, скрипт можно запустить просто по имени:

@user

```
[user@VM ~]$ hsh-shell --rooter
```

@rooter

```
[root@localhost .in]# rpm -i /usr/src/RPM/RPMS/x86_64/not-null-pkg-1.0-alt1.x86_64.rpm
<13>Jul  1 16:33:50 rpm: not-null-pkg-1.0-alt1 1751387597 installed
[root@localhost .in]# which not-null-pkg
/usr/bin/not-null-pkg
[root@localhost .in]# /usr/bin/not-null-pkg
This is not null pkg
[root@localhost .in]# not-null-pkg
This is not null pkg
[root@localhost .in]#
```

Глава 3. Пакеты с внешними зависимостями

На примере библиотеки NCurses разберём сборку пакета из исходным файлов. В отличие от примера, состоящего из уже исполняемого файла, в этом случае необходимо будет провести сборку с использованием сторонней библиотеки.

3.1. Явная сборка внутри hasher

Соберём пакет из одного исходного файла. Для начала загрузим необходимую для работы библиотеку. Для разработки используются специальные devel-версии пакетов с библиотеками. Как правило, devel-версии включают в себя всё, что не нужно в процессе эксплуатации: примеры, документацию в разных форматах, **`.h`**-файлы, профили для различных инструментов сборки, а также специально оформленную символьную ссылку на файл с библиотекой, установленный из основного пакета. По этой причине вместе с devel-пакетом ставится и основной:

@**user**:

```
[user@VM ~]$ hsh-install libncurses-devel
<13>Jul 11 02:07:34 rpmi: libncurses6-6.3.20220618-alt4 sisyphus+327286.4600.14.1
1711486705 installed
<13>Jul 11 02:07:34 rpmi: libtinfo-devel-6.3.20220618-alt4
sisyphus+327286.4600.14.1 1711486705 installed
<13>ul 11 02:07:34 rpmi: libncurses-devel-6.3.20220618-alt4
sisyphus+327286.4600.14.1 1711486705 installed
[user@VM ~]$
```

Рассмотрим ближе структуру devel-библиотеки:

@**builder**

```
[builder@localhost ~]$ rpm -ql libncurses-devel
```

- конфигурационные файлы библиотеки:

```
/usr/bin/ncurses6-config
/usr/bin/ncurses6-config
```

- заголовочные файлы:

```
/usr/include/curses.h
/usr/include/eti.h
/usr/include/form.h
/usr/include/menu.h
/usr/include/ncurses
/usr/include/ncurses.h
/usr/include/ncurses/curses.h
/usr/include/ncurses/eti.h
/usr/include/ncurses/form.h
/usr/include/ncurses/menu.h
/usr/include/ncurses/ncurses.h
/usr/include/ncurses/panel.h
/usr/include/ncurses/tic.h
/usr/include/ncurses/unctrl.h
/usr/include/panel.h
/usr/include/unctrl.h
```

- специальные символические ссылки на файлы динамической библиотеки и файлы компоновки **`pkg-config`**:

```
/usr/lib64/libcurses.so
/usr/lib64/libform.so
/usr/lib64/libmenu.so
/usr/lib64/libncurses.so
/usr/lib64/libpanel.so
/usr/lib64/pkgconfig/form.pc
/usr/lib64/pkgconfig/menu.pc
/usr/lib64/pkgconfig/ncurses.pc
/usr/lib64/pkgconfig/panel.pc
```

- набор документации и **man**:

```
/usr/share/doc/ncurses-6.3.20220618
/usr/share/doc/ncurses-6.3.20220618/announce.html
/usr/share/doc/ncurses-6.3.20220618/demo.cc
/usr/share/doc/ncurses-6.3.20220618/hackguide.doc
/usr/share/doc/ncurses-6.3.20220618/hackguide.html
/usr/share/doc/ncurses-6.3.20220618/index.html
/usr/share/doc/ncurses-6.3.20220618/ncurses-intro.doc
/usr/share/doc/ncurses-6.3.20220618/ncurses-intro.html
/usr/share/man/man1/ncurses5-config.1.xz
/usr/share/man/man1/ncurses6-config.1.xz
/usr/share/man/man1/ncurses5-config.1.xz
/usr/share/man/man1/ncurses6-config.1.xz
<...>
/usr/share/man/man3/wvline_set.3x.xz
[builder@localhost ~]$
```

Перейдём к сборке пакета. В качестве тестовой программы рассмотрим программу, которая считывает приходящие символы и выводит их вместе с ASCII-кодом:

@**builder**: **RPM/SOURCES/pkg-ncurses-1.0.c**

```
#include <curses.h>

int main(void) {
       WINDOW* win;
       WINDOW* frame;
       char c = 0;

       initscr();
       noecho();
       cbreak();

       move(4, 10);
       printw("window:");
       refresh();

       frame = newwin(LINES - 8, COLS - 18, 4, 9);
       box(frame, 0, 0);
       mvwaddstr(frame, 0, (int)((COLS - 25) / 2), "Рамка");
       wrefresh(frame);

       win = newwin(LINES - 10, COLS - 20, 5, 10);
       keypad(win, TRUE);
       scrollok(win, TRUE);

       while((c = wgetch(win)) != 27) {
              wprintw(win, "  %d: %s\n", c, keyname(c));
              wrefresh(win);
       }

       delwin(win);
       endwin();
       return 0;
}
```

Поскольку для сборки пакета появилась новая подзадача — компиляция исходников в исполняемые файлы, меняется и spec-файл. Для описания команд по сборке итоговых файлов пакета используется директива ***%build***:

@**builder**: **RPM/SPECS/pkg-ncurses.spec**

```
Name: pkg-ncurses
Version: 1.0
Release: alt1

Summary: Test pkg with ncurses library

License: GPL-3.0-or-later
Group: Development/Other

Source: %name-%version.c
BuildRequires: libncurses-devel

%description
This is a small testing package with ncurses functionality

%build
gcc %SOURCE0 -lncurses -o %name

%install
install -D -pm 755 %name %buildroot%_bindir/%name

%files
%_bindir/*

%changelog
* Thu Jul 03 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- InitialBuild
```

Сборка пакета проходит успешно, и наряду с обычным двоичным пакетом собирается пакет с *debuginfo*, хранящий в себе информацию, необходимую для отладки:

@**builder**

```
[builder@localhost ~]$ tree RPM
RPM
├── BUILD
├── RPMS
│ ├── noarch
│ └── x86_64
├── SOURCES
│ └── pkg-ncurses-1.0.c
├── SPECS
│ └── pkg-ncurses.spec
└── SRPMS

7 directories, 2 files
[builder@localhost ~]$

[builder@localhost ~]$ rpmbuild -ba RPM/SPECS/pkg-ncurses.spec
<...>
Wrote: /usr/src/RPM/SRPMS/pkg-ncurses-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/pkg-ncurses-1.0-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/pkg-ncurses-debuginfo-1.0-alt1.x86_64.rpm
(w2.lzdio)
[builder@localhost ~]$ tree -A RPM/
RPM/
```

```
├── BUILD
│   └── pkg-ncurses
├── RPMS
│   ├── noarch
│   └── x86_64
│       ├── pkg-ncurses-1.0-alt1.x86_64.rpm
│       └── pkg-ncurses-debuginfo-1.0-alt1.x86_64.rpm
├── SOURCES
│   └── pkg-ncurses-1.0.c
├── SPECS
│   └── pkg-ncurses.spec
└── SRPMS
    └── pkg-ncurses-1.0-alt1.src.rpm

8 directories, 6 files
[builder@localhost ~]$
```

Попробуем установить пакет в **это же** окружение:

@user

```
[user@VM ~]$ hsh-shell --rooter
```

@rooter

```
[root@localhost .in]# rpm -i /usr/src/RPM/RPMS/x86_64/pkg-ncurses-1.0-
alt1.x86_64.rpm
<13>Jul 3 05:26:57 rpm: pkg-ncurses-1.0-alt1 1751520368 installed
[root@localhost .in]#
[root@localhost .in]# which pkg-ncurses
/usr/bin/pkg-ncurses
[root@localhost .in]# pkg-ncurses
```

```
        ┌──────────────Рамка──────────────┐
        │  72: H                         │
        │  101: e                        │
        │  108: l                        │
        │  108: l                        │
        │  111: o                        │
        │  44: ,                         │
        │  32:                           │
        │  105: i                        │
        │  115: s                        │
        │  32:                           │
        │  105: i                        │
        │  116: t                        │
        │  32:                           │
        │  109: m                        │
        │  101: e                        │
        │  32:                           │
        │  121: y                        │
        │  111: o                        │
        │  117: u                        │
        │  32:                           │
        │  108: l                        │
        │  111: o                        │
        │  111: o                        │
        │  107: k                        │
```

```
        │  105: i                        │
        │  110: n                        │
        │  103: g                        │
        │  32:                           │
        │  102: f                        │
        │  111: o                        │
        │  114: r                        │
        │  63: ?                         │
        └────────────────────────────────┘
```

Во время разработки окружение hasher должно содержать сборочные зависимости. Для эксплуатации пакета достаточно установить только эксплуатационные. Эксплуатационные зависимости можно указывать явно, но в большинстве случаев они добавляются в *.rpm*-пакет автоматически в процессе сборки:

@builder

```
[builder@localhost ~]$ rpmbuild -ba RPM/SPECS/pkg-ncurses.spec
<...>
Finding Requires (using /usr/lib/rpm/find-requires)
Executing: /bin/sh -e /usr/src/tmp/rpm-tmp.BxgcRM
find-requires: running scripts
(cpp,debuginfo,files,lib,pam,perl,pkgconfig,pkgconfiglib,python,python3,rpmlib,sh
ebang,shell,static,symlinks,systemd-services)
Requires: /lib64/ld-linux-x86-64.so.2, libc.so.6(GLIBC_2.2.5)(64bit),
libc.so.6(GLIBC_2.34)(64bit), libncurses.so.6()(64bit) >=
set:njzUlMJGoZaFrOm1ipLX6ub3NlTS2FoKH5pxjl802, libtinfo.so.6()(64bit) >= set:liZK
bfJOyUPV1IqfEUb0, rtld(GNU_HASH)
Requires(rpmlib): rpmlib(SetVersions)
<...>
Wrote: /usr/src/RPM/SRPMS/pkg-ncurses-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/pkg-ncurses-1.0-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/pkg-ncurses-debuginfo-1.0-alt1.x86_64.rpm
(w2.lzdio)
[builder@localhost ~]$
```

Скопируем пакет из hasher и попробуем установить его в чистое окружение без предварительной установки Ncurses. Чтобы у **@rooter** была возможность обратиться к полученному пакету, необходимо будет перенести его в директорию общего доступа для всех трёх пользователей — специальную директорию **`.in`**:

@user

```
[user@VM ~]$ cp hasher/chroot/usr/src/RPM/RPMS/x86_64/pkg-ncurses-1.0-
alt1.x86_64.rpm
[user@VM ~]$ hsh --init
<...>
[user@VM ~]$ cp pkg-ncurses-1.0-alt1.x86_64.rpm hasher/chroot/.in/
[user@VM ~]$ hsh-shell --rooter
```

Поскольку библиотека является не только сборочной, но и эсплуатационной зависимостью пакета, её отсутствие не позволяет установить пакет:

@rooter

```
[root@localhost .in]# rpm -i pkg-ncurses-1.0-alt1.x86_64.rpm
error: Failed dependencies:
       libncurses.so.6()(64bit) >= set:mi6NHG6gUJ4yHTS2FoKH5hxjh80KggmL6K9X2A2 is
needed by pkg-ncurses-1.0-alt1.x86_64
[root@localhost .in]#
```

Для решения проблемы необходимо установить все недостающие зависимости (которые, вообще говоря, могут и сами потребовать каких-либо других зависимостей и т. д.). Также файлы, передаваемые в директорию **.in** в hasher, автоматически удаляются из неё при нарушении условий целостности сессии (одним из таких условий является установка пакетов в hasher), вследствие чего их необходимо вновь добавить.

При установке библиотеки для эксплуатации следует пользоваться не devel-версией, а обычной. В репозитории *Sysiphus* часто встречаются несколько версий библиотек: старые, для корректной работы некоторых устаревших пакетов, и обновлённые. Обновлённые библиотеки имеют формат именования **lib<имя_библиотеки>#**, где номер в конце обозначает версию библиотеки. Старые библиотеки не имеют нумерации. В devel-версиях формат именования всегда **lib<имя_библиотеки>-devel**, но ссылка в них указывает на обновлённую библиотеку. Поскольку мы работаем с обновлённой библиотекой, необходимо установить её:

@user

```
[user@VM ~]$ hsh-install libncurses6
<13>Jul 11 02:14:27 rpmi: libncurses6-6.3.20220618-alt4 sisyphus+327286.4600.14.1
1711486705 installed
[user@VM ~]$ cp pkg-ncurses-1.0-alt1.x86_64.rpm hasher/chroot/.in
[user@VM ~]$ hsh-shell --rooter
```

@rooter

```
[root@localhost .in]# rpm -i pkg-ncurses-1.0-alt1.x86_64.rpm
<13>Jul 11 02:14:41 rpm: pkg-ncurses-1.0-alt1 1752199720 installed

[root@localhost .in]# rpmquery --requires pkg-ncurses
/lib64/ld-linux-x86-64.so.2
libc.so.6(GLIBC_2.2.5)(64bit)
libc.so.6(GLIBC_2.34)(64bit)
libncurses.so.6()(64bit) >= set:mi6NHG6gUJ4yHTS2FoKH5hxjh80KggmL6K9X2A2
rpmlib(SetVersions)
libtinfo.so.6()(64bit) >= set:liZKbfJOyUPV1IqfEUb0
rtld(GNU_HASH)
rpmlib(PayloadIsLzma)
[root@localhost .in]#
```

3.2. Сборка с помощью make, tarball

Добавим к пакету автоматическую сборку с применением утилиты **make**. При этом возникает две задачи:

- создание единого файла-хранилища исходных данных;
- списание нового формата сборки исполняемого файла.

Из двух файлов (**Makefile** и исходный код) соберём архив по правилам разработки RPM-пакетов (почти по правилам, поскольку правильную иерархию поддиректорий внутри архива соблюдать не будем).

Важно

Выполнения команд по сборке архива необходимо делать непосредственно из директории с материалами архива. Выполнение сборки архива из внешних директорий притянет весь путь в архив и нарушит именование файлов.

Для начала создадим директорию будущего архива с именем в формате **%name-%version** (как этого требуют правила оформления пакета в ALT):

@builder

```
[builder@localhost ~]$ cd RPM/SOURCES/
[builder@localhost SOURCES]$ mkdir pkg-ncurses-1.1
```

Перенесём в эту директорию исходный текст программы и добавим **Makefile**.

Важно

При работе с Makefile в обязательном порядке необходимо использовать табуляцию:

- между целью рецепта и его зависимостями;
- при описании самого рецепта.

@builder: **RPM/SOURCES/pkg-ncurses-1.1/Makefile**

```
CC=cc
LDLIBS=-lncurses
CFLAGS=-Wall

%:      %.c
        $(CC) $(CFLAGS) $< $(LDLIBS) -o $@

all:    pkg-ncurses

clean:
        rm -f o oo $(EXE) *.o
```

Теперь соберём сам архив с помощью утилиты **tar** и сожмём его с помощью **gzip**:

@builder

```
[builder@localhost SOURCES]$ tar -cf pkg-ncurses-1.1.tar pkg-ncurses-1.1/*
[builder@localhost SOURCES]$ gzip pkg-ncurses-1.1.tar
[builder@localhost SOURCES]$ cd
[builder@localhost ~]$ rm -rf RPM/SOURCES/pkg-ncurses-1.1
```

Отдельно обсудим появившиеся в spec-файле изменения:

@**builder**: **RPM/SPECS/pkg-ncurses.spec**

```
Name: pkg-ncurses
Version: 1.1
Release: alt1

Summary: Test pkg with ncurses library

License: GPL-3.0-or-later
Group: Development/Other

Source: %name-%version.tar.gz
BuildRequires: libncurses-devel

%description
This is a small testing package with ncurses functionality

%prep
%setup

%build
%make_build

%install
install -D -pm 755 %_builddir/%name-%version/%name %buildroot%_bindir/%name

%files
%_bindir/*

%changelog
```

- в преамбуле директива ***Source*** указывает на архив исходных файлов;
- в основную часть добавлена директива ***%prep***, в которой описываются действия по подготовке исходных материалов к сборке. Макрос ***%setup*** включает в себя развёртывание архива в **RPM/BUILD/** для проведения сборки;
- в директиве ***%build*** макрос ***%make_build*** запускает автосборку проекта с помощью **make**;
- для утилиты **install** указывается путь к исполняемого файлу, используется макрос ***%_builddir*** для указания пути через **RPM/BUILD/**.

Сборка и установка пакета проводится успешно:

@**builder**

```
[builder@localhost ~]$ tree RPM
RPM
├── BUILD
├── RPMS
│ └── noarch
├── SOURCES
│ └── pkg-ncurses-1.1.tar.gz
├── SPECS
│ └── pkg-ncurses.spec
└── SRPMS

7 directories, 2 files
```

```
[builder@localhost ~]$ rpmbuild -ba RPM/SPECS/pkg-ncurses.spec
<...>
Wrote: /usr/src/RPM/SRPMS/pkg-ncurses-1.1-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/pkg-ncurses-1.1-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/pkg-ncurses-debuginfo-1.1-alt1.x86_64.rpm
(w2.lzdio)

[builder@localhost ~]$ tree RPM
RPM
├── BUILD
│      └── pkg-ncurses-1.1
│            ├── Makefile
│            ├── pkg-ncurses
│            └── pkg-ncurses.c
├── RPMS
│      ├── noarch
│      └── x86_64
│            ├── pkg-ncurses-1.1-alt1.x86_64.rpm
│            └── pkg-ncurses-debuginfo-1.1-alt1.x86_64.rpm
├── SOURCES
│      └── pkg-ncurses-1.1.tar.gz
├── SPECS
│      └── pkg-ncurses.spec
└── SRPMS
       └── pkg-ncurses-1.1-alt1.src.rpm

9 directories, 9 files
```

@user

```
[user@VM ~]$ hsh-shell --rooter
```

@rooter

```
[root@localhost .in]# rpm -i /usr/src/RPM/RPMS/x86_64/pkg-ncurses-1.1-
alt1.x86_64.rpm
<13>Jul 3 07:15:56 rpm: pkg-ncurses-1.1-alt1 1751526866 installed
[root@localhost .in]#
[root@localhost .in]# pkg-ncurses
[root@localhost .in]#
```

3.3. Hasher: работа с существующим пакетом

Кроме создания пакетов с нуля в hasher существует возможность установки окружения вокруг некоторого уже существующего пакета. Это необходимо для проверки работоспособности сборки некоторого пакета или для внесения обновлений или изменений в его структуру. Для начала обсудим особенности сопровождения пакетов.

Одним из параметров RPM-пакета является указание информации о разработчике и / или сопровождающем этот пакет. Макросы, описывающие основные параметры утилит RPM-сборки, включая информацию о разработчике, описаны в специальном файле **`.rpmmacros`**:

@builder

```
[builder@localhost ~]$ cat .rpmmacros
%_tmppath /usr/src/tmp
%_topdir /usr/src/RPM
%packager Automated package hasher <hasher@localhost>
%buildhost user.hasher.altlinux.org
%__BTE hasher
%__nprocs 2
%_pkg_contents_index_bin /.host/contents_index_bin
%_rpmbuild_clean 0
%_rpmbuild_packagesource 1
[builder@localhost ~]$
```

При создании окружения hasher автоматически добавляет информацию о мнимом пользователе **`Automated package hasher <hasher@localhost>`**. При сборке пакета в его метаданные записывается информация из spec-файла, а также добавляется информация о разработчике из **`.rpmmacros`**. Вообще говоря, для spec-файлов существует специальная директива ***`Packager`*** для указания разработчика, но допустимо (и рекомендуется) её не указывать:

@builder

```
[user@VM ~]$ rpm -qip hasher/chroot/usr/src/RPM/SRPMS/null-pkg-1.0-alt1.src.rpm
Name        : null-pkg
Version     : 1.0
Release     : alt1
Architecture: x86_64
Install Date: (not installed)
Group       : Development/Other
Size        : 281
License     : GPL-3.0-or-later
Signature   : (none)
Source RPM  : (none)
Build Date  : Вт 08 июл 2025 17:41:43
Build Host  : user.hasher.altlinux.org
Relocations : (not relocatable)
Packager    : Automated package hasher <hasher@localhost>
Vendor      : ALT Linux Team
Summary     : Null package
Description :
This is the smallest ever alt package without any functionality
[user@VM ~]$
```

При создании окружения вокруг пакета запускается автоматическая сборка, а также проводится проверка соответствия сборки пакетов правилам *Sysiphus* с помощью встроенной в hasher утилиты **sysiphus_check**. По правилам ALT Linux Team пакет должен принадлежать разработчику <User@altlinux.org>, все комментарии в директиве ***`%changelog`*** должны также принадлежать только таким пользователям. При создании окружения также и текущий разработчик должен соответствовать формату.

Если попробовать без предварительной настройки собрать окружение вокруг пакета, *sysiphus_check* выдаст ошибку и прервёт сборку:

@user

```
[user@VM ~]$ hsh null-pkg-1.0-alt1.src.rpm # src.rpm предварительно сохранен в
домашней директории
<...>
/usr/src/in/srpm/null-pkg-1.0-alt1.src.rpm: wrong PACKAGER: Automated package
```

```
hasher <hasher@localhost>
sisyphus_check: check-packager ERROR: packager name violation
/usr/src/in/srpm/null-pkg-1.0-alt1.src.rpm: wrong packager in CHANGELOGNAME:
UsamG1t <usamstudent21@gmail.com> 1.0-alt1
sisyphus_check: check-changelog ERROR: changelog format violation
hsh-rebuild: null-pkg-1.0-alt1.src.rpm: sisyphus_check failed.
[user@VM ~]$
```

3.4. Решение конфликта именования

3.4.1. Ручная настройка

Для ручной настройки необходимо:

1. При создании окружения для **инициализации нового пакета** указать флаг ***--packager*** с описанием правильного именования разработчика:

   ```
   [user@VM ~]$ hsh --init --packager="Name Surname <user@domen>"
   <...>
   [user@VM ~]$ hsh-shell
   [builder@localhost .in]$ cd
   [builder@localhost ~]$ cat .rpmmacros
   %_tmppath /usr/src/tmp
   %_topdir /usr/src/RPM
   %packager Name Surname <user@domen>
   %buildhost user.hasher.altlinux.org
   %__BTE hasher
   %__nprocs 2
   %_pkg_contents_index_bin /.host/contents_index_bin
   %_rpmbuild_clean 0
   %_rpmbuild_packagesource 1
   [builder@localhost ~]$
   ```

   Допустимо также вручную отредактировать файл **.rpmmacros**;

2. В ***%changelog*** spec-файла указать правильное именование разработчика;

3. **Не** использовать директиву spec-файла ***Packager***;

4. При создании окружения **на основе пакета** строго указывать флаг ***--packager*** с описанием правильного именования разработчика. Ручное редактирование **.rpmmacros** невозможно, поскольку проверка **sysiphus_check** срабатывает **до** передачи управления.

Примечание

Данные действия небходимо будет повторять для каждого пакета.

3.4.2. Автоматическая настройка

Hasher может автоматически добавлять правильное именование разработчика в систему. Для этого необходимо добавить соответствующее описание в специальный конфигурационный файл **~/.hasher/config**:

@**user**: **`.hasher/config`**

```
packager="UsamG1t <usamg1t@altlinux.org>"
```

@**user**

```
[user@VM ~]$ hsh --init
<...>
[user@VM ~]$ hsh-shell
[builder@localhost .in]$ cd
[builder@localhost ~]$ cat .rpmmacros
%_tmppath /usr/src/tmp
%_topdir /usr/src/RPM
%packager UsamG1t <usamg1t@altlinux.org>
%buildhost user.hasher.altlinux.org
%__BTE hasher
%__nprocs 2
%_pkg_contents_index_bin /.host/contents_index_bin
%_rpmbuild_clean 0
%_rpmbuild_packagesource 1
[builder@localhost ~]$
```

Для заполнения ***`%changelog`*** теперь можно будет воспользоваться специальными плагинами автодобавления сообщений (например, помощью vim-плагина).

При создании окружения вокруг пакета *sysiphus_check* теперь не выдаст ошибки. Сборка без ключей создаст окружение, соберёт пакет, и, в случае успеха, автоматически очистит окружение. Для последующей разработки или устранения неполадок сборки необходимо использовать ключ ***`--lazy`***.

Настройка окружения вокруг некоторого пакета открывает дополнительную возможность hasher — развёртывание собственного локального репозитория пакетов на устройстве. Доступ к репозиторию, в котором хранятся как пакеты с исходным кодом, так и двоичные пакеты, осуществляется с @**user** через директорию **`hasher/repo/`**:

@**user**

```
[user@VM ~]$ tree hasher/repo/
hasher/repo/
├── SRPMS.hasher
│   ├── not-null-pkg-1.0-alt1.src.rpm
│   └── null-pkg-1.0-alt1.src.rpm
└── x86_64
    └── RPMS.hasher
        ├── not-null-pkg-1.0-alt1.x86_64.rpm
        └── null-pkg-1.0-alt1.x86_64.rpm

4 directories, 4 files
[user@VM ~]$
[user@VM ~]$ cp hasher/chroot/usr/src/RPM/SRPMS/pkg-ncurses-1.0-alt1.src.rpm
[user@VM ~]$ hsh pkg-ncurses-1.0-alt1.src.rpm
<...>
[user@VM ~]$ tree hasher/repo/
hasher/repo/
├── SRPMS.hasher
│   ├── not-null-pkg-1.0-alt1.src.rpm
│   ├── null-pkg-1.0-alt1.src.rpm
```

```
│   └── pkg-ncurses-1.0-alt1.src.rpm
└── x86_64
    └── RPMS.hasher
        ├── not-null-pkg-1.0-alt1.x86_64.rpm
        ├── null-pkg-1.0-alt1.x86_64.rpm
        ├── pkg-ncurses-1.0-alt1.x86_64.rpm
        └── pkg-ncurses-debuginfo-1.0-alt1.x86_64.rpm
4 directories, 7 files
[user@VM ~]$
```

Глава 4. Использование многофайловой сборки в пакете

При написании крупных программ многофайловость позволяет лучше ориентироваться в программе, разделять блоки кода и пространства имён. Для автоматической сборки программ, состоящих из множества файлов, используется утилита GNU make.

При работе с крупными проектами часто используются сборочные библиотеки, при этом не только сторонние, но и собирающиеся из исходных файлов самого проекта. Например, если проект состоит из нескольких программ, общую их часть естественно вынести в отдельную библиотеку и собирать все программы с ней.

Построим пакет, в котором соберём библиотеку и два исполняемых файла: один будет компоноваться со статическим, а второй — с динамическим вариантом библиотеки.

Программа состоит из четырёх файлов: основной код, код библиотеки в двух файлах и заголовочный файл:

@builder: **RPM/SOURCES/Multilab-1.0/fun.c**

```
#include <stdio.h>
#include "outlib.h"
void output(char *str) {
        printf("%d: %s\012", Count++, str);
}

void usage(char *prog) {
        fprintf(stderr, "%s v%.2f: Print all arguments\012\t"\
                        "Usage: %s arg1 [arg2 […]]\012", prog, VERSION, prog);
}
```

@builder: **RPM/SOURCES/Multilab-1.0/Multilab.c**

```
#include <stdio.h>
#include "outlib.h"

int main(int argc, char *argv[]) {
        int i;
        if((Count = argc)>1) {
                output("<INIT>");
                for(i=1; i<argc; i++)
                        output(argv[i]);
                output("<DONE>");
        }
```

```
        else
                usage(argv[0]);
return 0;
}
```

@builder: **RPM/SOURCES/Multilab-1.0/Multilab.c**

```
int Count=0;
```

@builder: **RPM/SOURCES/Multilab-1.0/outlib.h**

```
void output(char *);
void usage(char *);
extern int Count;
#define VERSION 1.0
```

Опишем **Makefile** для проекта и рассмотрим подробнее его рецепты:

@builder: **RPM/SOURCES/Multilab-1.0/Makefile**

```
GENS = Multilab-* README-* lib*
TRASH = *.o *~ o.*
CFLAGS = -Wall -fPIC
CC = cc

.PHONY: clean distclean
.SECONDARY:     fun.o const.o
.INTERMEDIATE:  libstatlib.a(fun.o const.o)


all:    binfiles documentation

binfiles:       Multilab-a Multilab-so

Multilab-a:     Multilab.o libstatlib.a
        $(CC) $(CFLAGS) $< -L. -lstatlib -o $@

Multilab-so:    Multilab.o libdynlib.so
        $(CC) $(CFLAGS) $< -o $@

libstatlib.a:   libstatlib.a(fun.o const.o)

libdynlib.so:   fun.o const.o
        $(CC) $(CFLAGS) $^ -o $@ -shared

fun.o Multilab.o:       outlib.h

documentation:  README-a README-so

README-a:       Multilab-a
        ./$< > $@ 2>&1

README-so:      Multilab-so
        LD_LIBRARY_PATH=`pwd` ./$< > $@ 2>&1

clean:
```

```
        rm -f $(TRASH)

distclean:      clean
        rm -f $(GENS)
```

- **make** поддерживает множество встроенных рецептов, для работы которых используются переменные окружения. Это позволяет задавать рецепты, лишь описывая правильный формат цели и исходников;
- при повторных сборках **make** ориентируется на метки времени создания и модификации файлов для проведения выборочной перекомпиляции. В случае самостоятельной генерации статических библиотек необходимы дополнительные цели для настройки правил пересборки;
- для каждого из целевых исполняемых файлов собирается своя библиотека на основе исходных файлов функции и глобальной переменной. Заметим, что для статических библиотек **make** поддерживает встроенный сценарий сборки (несколько неожиданный синтаксис, задающий такое правило, см. в примере), в то время как для динамической необходимо явно указывать сценарий;
- оба варианта сборки программы с библиотекой — вида **cc файлы.o libбиблиотека -o программа** и вида **cc файлы.o -:путь_к_библиотеке-l библиотека -o программа** — годятся и для статической, и для динамической сборок. Второй вариант предпочтительнее;
- ключ ***-fPIC*** (сгенерировать позиционно-независимый код) нужен только для динамической сборки, в статической он сравнительно бесполезен.

Для сборки пакета оформим **`tarball`** и перейдём к описанию spec-файла:

@builder

```
[builder@localhost Multilab-1.0]$ cd
[builder@localhost ~]$ tree RPM
RPM
├── BUILD
├── RPMS
│   └── noarch
├── SOURCES
│   └── Multilab-1.0
│       ├── Makefile
│       ├── Multilab.c
│       ├── const.c
│       ├── fun.c
│       └── outlib.h
├── SPECS
└── SRPMS

8 directories, 5 files
[builder@localhost ~]$ cd RPM/SOURCES/
[builder@localhost SOURCES]$ tar -cf Multilab-1.0.tar Multilab-1.0/*
[builder@localhost SOURCES]$ gzip Multilab-1.0.tar
[builder@localhost SOURCES]$ cd
[builder@localhost ~]$ tree -A RPM
RPM
├── BUILD
├── RPMS
│   └── noarch
├── SOURCES
│   ├── Multilab-1.0
```

```
│   │   ├── Makefile
│   │   ├── Multilab.c
│   │   ├── const.c
│   │   ├── fun.c
│   │   └── outlib.h
│   └── Multilab-1.0.tar.gz
├── SPECS
└── SRPMS

8 directories, 6 files
[builder@localhost ~]$
```

@builder: **RPM/SPECS/Multilab.spec**

```
Name: Multilab
Version: 1.0
Release: alt1

Summary: Package with both types of libraries

License: GPL-3.0-or-later
Group: Development/Other

Source: %name-%version.tar.gz

%description
This is an example of make usage. Make is compiling project with static and
dynamic libraries

%prep
%setup

%build
%make_build

%install
install -D %name-a %buildroot%_bindir/%name-a
install -D %name-so %buildroot%_bindir/%name-so
install -D -m644 libdynlib.so %buildroot%_libdir/libdynlib.so


%files
%_bindir/*
%_libdir/*

%changelog
* Thu Jul 04 2025 UsamG1t <usamg1t@altlinux.org> 1.1-alt1
- Makefile big package
```

Распределение файлов по каталогам системы при установке должно соответствовать общепринятой иерархии (она описана, например, в man 7 hier).

Динамически собранная программа заработает, только когда библиотека окажется в системном каталоге (обычно — **/usr/lib64**), или в окружении будет явно задан каталог (**LD_LIBRARY_PATH**), содержащий эту библиотеку, или сама библиотека будет заранее подгружена с помощью **LD_LIBRARY_PATH**. Для проверки соберём оба исполняемых файла и сравним:

@builder

```
[builder@localhost ~]$ cd RPM/SOURCES/Multilab-1.0
[builder@localhost Multilab-1.0]$ make
cc -Wall -fPIC   -c -o Multilab.o Multilab.c
cc -Wall -fPIC   -c -o fun.o fun.c
ar -rv libstatlib.a fun.o
ar: creating libstatlib.a
a - fun.o
cc -Wall -fPIC   -c -o const.o const.c
ar -rv libstatlib.a const.o
a - const.o
cc -Wall -fPIC Multilab.o -L. -lstatlib -o Multilab-a
cc -Wall -fPIC fun.o const.o -o libdynlib.so -shared
cc -Wall -fPIC Multilab.o -L. -ldynlib -o Multilab-so
./Multilab-a > README-a 2>&1
LD_LIBRARY_PATH=`pwd` ./Multilab-so > README-so 2>&1
```

Запустим утилиту **ldd**, возвращающую список динамических библиотек, требуемых файлу:

@builder

```
[builder@localhost Multilab-1.0]$ ldd Multilab-a
       linux-vdso.so.1 (0x00007f6ca740e000)
       libc.so.6 => /lib64/libc.so.6 (0x00007f6ca7219000)
       /lib64/ld-linux-x86-64.so.2 (0x00007f6ca7410000)
[builder@localhost Multilab-1.0]$ ldd Multilab-so
       linux-vdso.so.1 (0x00007fcb4fe1a000)
       libdynlib.so => not found
       libc.so.6 => /lib64/libc.so.6 (0x00007fcb4fc25000)
       /lib64/ld-linux-x86-64.so.2 (0x00007fcb4fe1c000)
[builder@localhost Multilab-1.0]$ LD_LIBRARY_PATH=`pwd` ldd Multilab-so
       linux-vdso.so.1 (0x00007f266dbe7000)
       libdynlib.so > /usr/src/RPM/SOURCES/Multilab-1.0/libdynlib.so
(0x00007f266dbd7000)
       libc.so.6 => /lib64/libc.so.6 (0x00007f266d9ed000)
       /lib64/ld-linux-x86-64.so.2 (0x00007f266dbe9000)
```

- в **Multilab-a** добавлена статическая библиотека **statlb.a**, из динамических нужна только стандартная LibC;
- **Multilab-so** собран с динамической библиотекой **libdynlib.so**, которой нет в стандартных местах;
- после явного указания расположения библиотека нашлась.

@builder

```
[builder@localhost Multilab-1.0]$ ./Multilab-a
./Multilab-a v1.00: Print all arguments
       Usage: ./Multilab-a arg1 [arg2 […]]

[builder@localhost Multilab-1.0]$ ./Multilab-so
./Multilab-so: error while loading shared libraries: libdynlib.so: cannot open
shared object file: No s
uch file or directory

[builder@localhost Multilab-1.0]$ LD_LIBRARY_PATH=`pwd` ./Multilab-so
```

```
./Multilab-so v1.00: Print all arguments
       Usage: ./Multilab-so arg1 [arg2 […]]

[builder@localhost Multilab-1.0]$ make distclean
rm -f *.o *~ o.*
rm -f Multilab-* README-* lib*
[builder@localhost Multilab-1.0]$ cd
[builder@localhost ~]$ mv RPM/SOURCES/Multilab-1.0
```

При сборке пакета явно выделяется директива ***%build***:

@builder

```
[builder@localhost ~]$ rpmbuild -ba RPM/SPECS/Multilab.spec
<...>

Executing(%build): /bin/sh -e /usr/src/tmp/rpm-tmp.37137
+ umask 022
+ /bin/mkdir -p /usr/src/RPM/BUILD
+ cd /usr/src/RPM/BUILD
+ cd Multilab-1.0
+ make
make: Entering directory '/usr/src/RPM/BUILD/Multilab-1.0'
cc -Wall -fPIC   -c -o Multilab.o Multilab.c
cc -Wall -fPIC   -c -o fun.o fun.c
ar -rv libout.a fun.o
ar: creating libout.a
a - fun.o
cc -Wall -fPIC   -c -o const.o const.c
ar -rv libout.a const.o
a - const.o
cc -Wall -fPIC Multilab.o -L. -lout -o Multilab-a
cc -Wall -fPIC fun.o const.o -o libout.so -shared
cc -Wall -fPIC Multilab.o libout.so -o Multilab-so
./Multilab-a > README-a 2>&1
LD_LIBRARY_PATH=`pwd` ./Multilab-so > README-so 2>&1

<...>

Wrote: /usr/src/RPM/SRPMS/Multilab-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/Multilab-1.0-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/Multilab-debuginfo-1.0-alt1.x86_64.rpm (w2.lzdio)
```

Вот так выглядит дерево каталогов после сборки: распакованный архив и генераты в **~/RPM/BUILD/**, *.rpm*-пакеты в **~/RPM/RPMS/** (включая автоматически собранный debuginfo-пакет, в который попадает отладочная информация) и *.src.rpm*-пакет в **~/RPM/SRPMS**

```
[builder@localhost ~]$ tree RPM
RPM
├── BUILD
│   └── Multilab-1.0
│       ├── Makefile
│       ├── Multilab-a
│       ├── Multilab-so
│       ├── Multilab.c
│       ├── Multilab.o
│       ├── README-a
│       ├── README-so
```

```
│       ├── const.c
│       ├── const.o
│       ├── fun.c
│       ├── fun.o
│       ├── libout.a
│       ├── libout.so
│       └── outlib.h
├── RPMS
│   ├── noarch
│   └── x86_64
│       ├── Multilab-1.0-alt1.x86_64.rpm
│       └── Multilab-debuginfo-1.0-alt1.x86_64.rpm
├── SOURCES
│   └── Multilab-1.0.tar.gz
├── SPECS
│   ├── Multilab.spec
└── SRPMS
    └── Multilab-1.0-alt1.src.rpm

9 directories, 20 files
[builder@localhost ~]$
```

После установки пакета динамическая библиотека попадает в стандартный каталог **/usr/lib64**, так что **Multilab-so** можно запускать просто по имени:

@rooter

```
[root@localhost .in]# rpm -i /usr/src/RPM/RPMS/x86_64/Multilab-1.0-
alt1.x86_64.rpm
<13>Jul  4 18:05:27 rpm: Multilab-1.0-alt1 1751652277 installed
[root@localhost .in]#
[root@localhost .in]# rpm -ql Multilab
/usr/bin/Multilab-a
/usr/bin/Multilab-so
/usr/lib64/libdynlib.so
[root@localhost .in]# Multilab-a qwerty
2: <INIT>
3: qwerty
4: <DONE>
[root@localhost .in]# Multilab-so qwerty
2: <INIT>
3: qwerty
4: <DONE>
[root@localhost .in]#
```

Глава 5. Работа со сценариями

5.1. Unix Shell Package

При разработке пакета для универсальной работы с внешними программами используются языки склейки. Это универсальный интерфейс управления системы, позволяющий использовать встроенные в систему программы.

В качестве базового набора используются команды Unix Shell. Соберём пакет с приложением, написанном на языке склейки:

@builder: **RPM/SOURCE/todo-pkg-1.0.sh**

```
#!/bin/bash

WORKDIR=$HOME/.config/shell-pkg
TODOLIST=$WORKDIR/todo-list
ANSWER_FILE=`mktemp --suffix=-shell-pkg`

exit_handler() { trap - EXIT; rm -f "$ANSWER_FILE"; }
trap exit_handler EXIT HUP INT QUIT PIPE TERM

mkdir -p $WORKDIR
test -r $TODOLIST || touch $TODOLIST
TODOCOUNT=`wc -l < $TODOLIST`

Auto_screensize() {
        eval `dialog --print-maxsize --stdout | sed -E 's/.* (.*), (.*)/W=\1;
H=\2; WW=$((W-10)); HH=$((H-10))/'`
}

Menu() {
        Auto_screensize
        if dialog --title ShellPkg --ok-label "Choose" --cancel-label "Exit" \
                --menu "" $WW $HH 3 \
                Show_todo "Todo list" \
                Add_todo "Add TODO" \
                Solve_todo "Solve TODO" \
                2> "$ANSWER_FILE"
        then
                read answer < "$ANSWER_FILE"
                $answer
        else
                return -1
        fi
}

Add_todo() {
        Auto_screensize
        if dialog --inputbox "Please write your TODO" $WW $HH 2> "$ANSWER_FILE"
        then
                read answer < "$ANSWER_FILE"
                ((TODOCOUNT++))
                echo "$TODOCOUNT NEW $answer" >> $TODOLIST
        fi
}

Show_todo() {
        solved_todo="Solved TODO:\n"
        unsolved_todo="Unsolved TODO:\n"

        while read number status todo; do
                if [ $status = "NEW" ]; then
                        unsolved_todo="$unsolved_todo - $todo\n"
                else
                        solved_todo="$solved_todo - $todo\n"
                fi
```

```
        done < "$TODOLIST"

        Auto_screensize
        dialog --title "List of all your TODO" --msgbox
"$solved_todo$unsolved_todo" $WW $HH
}


Solve_todo() {
        unsolved_todo=""
        count=0
        while read number status todo; do
                if [ $status = "NEW" ]; then
                        unsolved_todo="$unsolved_todo $number ${todo// / } off"
                        ((count++))
                fi
        done < "$TODOLIST"

        Auto_screensize
        if dialog --title "Mark solved TOSOs" \
                  --checklist "" $WW $HH $count $unsolved_todo \
                  2> "$ANSWER_FILE"
        then
                read answer < "$ANSWER_FILE"
                for num in $answer; do
                sed -i -E "s/^($num) NEW/\\1 DONE/" "$TODOLIST"
                done
        fi
}

while Menu; do :; done
```

Подробнее рассмотрим некоторые конструкции программы:

- для удобства работы определены функции:

```
Menu() {
        Auto_screensize
        if dialog --title ShellPkg --ok-label "Choose" --cancel-label "Exit" \
                --menu "" $WW $HH 3 \
                Show_todo "Todo list" \
                Add_todo "Add TODO" \
                Solve_todo "Solve TODO" \
                2< "$ANSWER_FILE"
        then
                read answer < "$ANSWER_FILE"
                $answer
        else
                return -1
        fi
}
```

- для проверки атрибутов используются условные конструкции ***if-then-else-fi***, для описания условий проверки используется сокращённый макрос оператора ***test*** — ***[***;

- нам пришлось пойти на маленькую хитрость. Подстановка получившегося текста в ***$unsolved_todo*** разбивает этот текст на отдельные слова, и если в ***$todo*** были пробелы, **dialog** работает не так, как ожидалось. Простое добавление кавычек не помогает. Поэтому необходимо заменить пробел на неразрывный пробел (символ с кодом 0xa0c2), который отображается так же, но **shell** не считает его разделителем;

```
while read number status todo; do
        if [ $status = "NEW" ]; then
                unsolved_todo="$unsolved_todo $number ${todo// / } off"
                ((count++))
        fi
done < "$TODOLIST"
```

- основное управление осуществляется через меню с помощью цикла **`while`**:

```
while Menu; do :; done
```

- псевдографический интерфейс обеспечивается с помощью утилиты dialog;

```
Auto_screensize() {
        eval `dialog --print-maxsize --stdout | sed -E 's/.* (.*), (.*)/W=\1;
H=\2; WW=$((W-10)); HH=$((H-10))/'`
}

<...>

dialog --title "List of all your TODO" --msgbox "$solved_todo$unsolved_todo"
$WW $HH
```

- для UI dialog использует Ncurses, так что весь управляющий вывод (например, метка выбранного пункта меню) выводится в другой дескриптор (2, то есть stderr) и перенаправляется в файл;

- файл временный, заводится при старте с помощью утилиты **mktemp**, она же обеспечивает ему уникальность имени;

- удаляется файл по окончании работы сценария.

Теперь разберём spec-файл:

@builder: **`RPM/SPECS/todo-pkg.spec`**

```
Name: todo-pkg
Version: 1.0
Release: alt1

Summary: Terminal TODO-list

License: GPL-3.0-or-later
Group: Development/Other

Source: %name-%version.sh

%description
Application Add and solve your TODO-s in this app

%install
```

```
install -D %SOURCE0 %buildroot%_bindir/%name

%files
%_bindir/*

%changelog
* Wed Jul 09 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial Build
```

Заметим, что сборочных зависимостей у пакета нет, но будут эксплуатационные — пакет *dialog*, необходимый для отрисовки интерфейса. Автоматический поиск зависимостей, срабатывающий при сборке пакета, позволяет явно не указывать в spec-файле данный пакет:

@builder

```
[builder@localhost ~]$ tree -A RPM
RPM
├── BUILD
├── RPMS
│   └── noarch
├── SOURCES
│   └── todo-pkg-1.0.sh
├── SPECS
│   └── todo-pkg.spec
└── SRPMS

7 directories, 2 files
[builder@localhost ~]$ rpmbuild -ba RPM/SPECS/todo-pkg.spec
Executing(%install): /bin/sh -e /usr/src/tmp/rpm-tmp.96219
<...>
Finding Requires (using /usr/lib/rpm/find-requires)
...
find-requires: FINDPACKAGE-COMMANDS: dialog mkdir rm sed touch
Requires: /bin/bash, /etc/bashrc, coreutils, dialog, sed
<...>
Wrote: /usr/src/RPM/SRPMS/todo-pkg-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/todo-pkg-1.0-alt1.x86_64.rpm (w2.lzdio)
[builder@localhost ~]$
```

Поскольку зависимость эксплуатационная, сборка пакета не требует наличия *dialog*, однако при установке появится предупреждение:

@rooter

```
[root@localhost .in]# rpm -i todo-pkg-1.0-alt1.x86_64.rpm
error: Failed dependencies:
       dialog is needed by todo-pkg-1.0-alt1.x86_64
[root@localhost .in]# rpmquery --requires --package todo-pkg-1.0-alt1.x86_64.rpm
/bin/bash
/etc/bashrc
coreutils
dialog
sed
rpmlib(PayloadIsLzma)
[root@localhost .in]#
```

@user

```
[user@VM ~]$ hsh-install dialog
<13>Jul 17 08:04:33 rpmi: libdialog-1.3.20171209-alt2 sisyphus+328094.100.1.1
1693228848 installed
<13>Jul 17 08:04:33 rpmi: dialog-1.3.20171209-alt2 sisyphus+328094.100.1.1
1693228848 installed
[user@VM ~]$ cp todo-pkg-1.0-alt1.x86_64.rpm hasher/chroot/.in
[user@VM ~]$ hsh-shell --rooter
```

@rooter

```
[root@localhost .in]# rpm -i todo-pkg-1.0-alt1.x86_64.rpm
<13>Jul 17 08:04:56 rpm: todo-pkg-1.0-alt1 1752738978 installed
[root@localhost .in]#
```

5.2. Gear by ALT Linux Team

Большим минусом разработки с использованием только hasher является постоянная пересборка не только файлов, связанных с предыдущими сборками, но и всего рабочего окружения разработчика, из-за чего постоянно необходимо настраивать «рабочее место». Решением этого выступает Gear, использующий для хранения данных git-репозиторий. Gear позволяет единожды настроить локальное рабочее пространство для разработки, вести работу в git, а сборку осуществлять с использованием передачи пакета для сборки в hasher.

Создадим gear-окружение вокруг пакета. Для этого необходимо создать репозиторий и распаковать в него пакет. Для начала (аналогично настройке именования разработчика hasher) необходимо указать правильное именование git-разработчика:

@user

```
[user@VM ~]$ git config --global user.name 'UsamG1t'
[user@VM ~]$ git config --global user.email 'usamg1t@altlinux.org'
[user@VM ~]$
```

@user

```
[user@VM ~]$ mkdir todo-pkg
[user@VM ~]$ cd todo-pkg
[user@VM todo-pkg]$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/user/todo-pkg/.git/
[user@VM todo-pkg]$
```

Распаковка *.src.rpm*-пакета и формирование gear-окружения осуществляется с помощью специальной команды **`gear-srpmimport`**:

```
[user@VM todo-pkg]$ gear-srpmimport ../todo-pkg-1.0-alt1.src.rpm
[srpms (root-commit) 4c1a0a3] 1.0-alt1
3 files changed, 109 insertions(+)
create mode 100644 .gear/rules
create mode 100644 todo-pkg-1.0.sh
create mode 100644 todo-pkg.spec
gear-srpmimport: Imported /home/user/todo-pkg-1.0-alt1.src.rpm
gear-srpmimport: Created master branch
[user@VM todo-pkg]$
```

В Gear все исходники и spec-файл хранятся в едином пространстве. Для правильной компоновки файлов (в tarball или просто набор исходников) при сборке пакета или генерации *.src.rpm*-пакета используется специальный файл правил экспорта **`.gear/rules`**:

@user

```
[user@VM todo-pkg]$ tree -A
.
├── todo-pkg-1.0.sh
└── todo-pkg.spec

1 directory, 2 files
[user@VM todo-pkg]$ tree -Aa
.
├── .gear
│   └── rules
├── .git
│   ├── branches
...
│   └── refs
│       ├── heads
│       │   ├── master
│       │   └── srpms
│       └── tags
│           └── 1.0-alt1
├── todo-pkg-1.0.sh
└── todo-pkg.spec

22 directories, 37 files
[user@VM todo-pkg]$
```

Для сборки пакета с помощью hasher используется команда **`gear-hsh`**. При этом допустимы ключи, используемые в hasher:

@user

```
[user@VM todo-pkg]$ gear-hsh --lazy
<...>
Wrote: /usr/src/in/srpm/todo-pkg-1.0-alt1.src.rpm (w1.gzdio)
Installing todo-pkg-1.0-alt1.src.rpm
<...>
Wrote: /usr/src/RPM/SRPMS/todo-pkg-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/todo-pkg-1.0-alt1.x86_64.rpm (w2.lzdio)
0.89user 1.12system 0:03.79elapsed 53%CPU (0avgtext+0avgdata 7612maxresident)k
0inputs+264outputs (0major+90729minor)pagefaults 0swaps
[user@VM todo-pkg]$ hsh-shell
```

@builder

```
[builder@localhost .in]$ cd
[builder@localhost ~]$ ls RPM/SOURCES/ RPM/SPECS/
RPM/SOURCES/:
todo-pkg-1.0.sh

RPM/SPECS/:
todo-pkg.spec
[builder@localhost ~]$
```

Глава 6. Отладка разработки

При разработке не всегда есть возможность обратиться к исходникам программы для поиска неисправностей при работе. Часто при отладке получается довольствоваться лишь ассемблерным кодом. Для полноценной работы необходимы знания всех внутренностей исполняемых файлов и исходников программы, с которой нужно работать.

Структура бинарных файлов в Linux имеет особый формат ELF (Executable and Linkable Format). Он состоит из большого количества секций, описывающих связи компонентов программы, хранящихся в ней данных и обо всех именах объектов, присутствующих в коде. Информация о разделах, именах и связанных объектах доступна с помощью специальных утилит **nm** (описывает связи по статически собранным объектам) и **readelf** или **objdump**, позволяющих изучить в том числе и динамически собранные связи:

@builder: **`ex.c`**

```
#include <stdio.h>
int N = 42;

int fun2(int n) {
        return n*2+1;
}

int fun1(int c) {
        c += 1;
        return fun2(c) + fun2(c*2);
}

int main(int argc, char *argv[]) {
        int i;
        for(i=0; i <10; i++)
                printf("%d\n", fun1(N+i));
        return 0;
}
```

@builder

```
[builder@localhost ~]$ cc -O0 -g ex.c -o ex
[builder@localhost ~]$ nm ex
0000000000004004 D N
0000000000003dc8 d _DYNAMIC
0000000000003fb8 d _GLOBAL_OFFSET_TABLE_
0000000000002000 R _IO_stdin_used
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
0000000000002130 r __FRAME_END__
0000000000002008 r __GNU_EH_FRAME_HDR
0000000000004008 D __TMC_END__
000000000000039c r __abi_tag
0000000000004008 B __bss_start
                 w __cxa_finalize@GLIBC_2.2.5
0000000000004000 D __data_start
00000000000010f0 t __do_global_dtors_aux
0000000000003db8 d __do_global_dtors_aux_fini_array_entry
0000000000003dc0 D __dso_handle
0000000000003db0 d __frame_dummy_init_array_entry
                 w __gmon_start__
                 U __libc_start_main@GLIBC_2.34
0000000000004008 D _edata
0000000000004010 B _end
00000000000011cc T _fini
0000000000001000 T _init
0000000000001050 T _start
0000000000004008 b completed.0
0000000000004000 W data_start
0000000000001080 t deregister_tm_clones
0000000000001130 t frame_dummy
000000000000114a T fun1
0000000000001139 T fun2
000000000000117a T main
                 U printf@GLIBC_2.2.5
00000000000010b0 t register_tm_clones
[builder@localhost ~]$
[builder@localhost ~]$ readelf -h ex
ELF Header:
 Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
 Class:                             ELF64
 Data:                              2's complement, little endian
 Version:                           1 (current)
 OS/ABI:                            UNIX - System V
 ABI Version:                       0
 Type:                              DYN (Position-Independent Executable file)
 Machine:                           Advanced Micro Devices X86-64
 Version:                           0x1
 Entry point address:               0x1050
 Start of program headers:          64 (bytes into file)
 Start of section headers:          17048 (bytes into file)
 Flags:                             0x0
 Size of this header:               64 (bytes)
 Size of program headers:           56 (bytes)
 Number of program headers:         13
 Size of section headers:           64 (bytes)
 Number of section headers:         39
 Section header string table index: 38
[builder@localhost ~]$
```

Для отладки программы необходимо сделать специальную подготовку исходных данных:

1. Компиляция программы с нулевой оптимизацией (флаг ***-O0***) для избежания несостыковок исходного текста программы с бинарным файлом (например, в исходнике есть переменная, которая нигде не используется. При оптимизации переменная будет удалена, а в исходниках её упоминание останется);

2. Генерация полного **`debuginfo`** (флаг ***-g***) для сбора данных, сгенерированных компилятором для описания исходного текста во время отладки;

3. Доступ к исходникам, на которые будет осуществляться ссылка. Файлы с исходными текстами должны лежать там, где было указано во время сборки, по умолчанию — в текущем каталоге, однако в большинстве отладчиков есть возможность указать альтернативный каталог для поиска. Отлаживать возможно и без исходников, но только в формате ассемблерного кода.

6.1. GDB

Вообще, отладка — это огромная тема, подробнее этот аспект можно изучить в этой книжке. Рассмотрим более подробно отладчик GDB — GNU Debugger.

GDB предоставляет достаточно удобный терминальный интерфейс для отладки программ. Основные команды отладчика:

- run — запуск программы для отладки;
- set args — установка параметров командной строки для отлаживаемой программы;
- установка точек останова:
  - breakpoint — срабатывает в момент выполнения отмеченной строки кода;
  - watchpoint — срабатывает по изменению регистра или ячейки памяти;
  - catchpoint — срабатывает по c++-исключению или системному вызову;
- start — создание временной точки останова в начале функции ***`main()`*** и запуск программы;
- next — следующий шаг программы (без перехода по функциям);
- step — следующий шаг программы (со всеми переходами в функции);
- continue — продолжение исполнения программы до следующей точки останова;
- finish — продолжение исполнения подпрограммы до выхода из неё;
- list — просмотр части программы, где сейчас остановился отладчик;
- print — просмотр значений переменных и выражений;
- backtrace — просмотр стека связанных фреймов текущего места обработки отладчика:
  - up / down — перемещение «фокуса» отладчика по стеку фреймов;
- display — всё время показывает значение переменной;
- dump — запись в файл.

Для запуска GDB в hasher необходимо выдать права на монтирование системной директории **/proc** с правами на чтение и запись (причина этого описана тут). Корректное монтирование файловых систем происходит при одновременном выполнении следующих четырех условий:

1. Файловая система описана в файле **/etc/hasher-priv/fstab** (причём для **/proc** в случае работы с GDB должны быть явно описаны права на чтение и запись):

```
[root@VM ~]# cat /etc/hasher-priv/fstab
# Information about mount points for the hasher-priv(8) helper program.
# See fstab(5) for details.

proc /proc proc rw,nosuid,nodev,noexec,gid=proc,hidepid=2 0 0
[root@VM ~]#
```

2. В конфигурации **hasher-priv** (**/etc/hasher-priv/system**) файловая система указана в опции ***allowed_mountpoints***:

```
[root@VM ~]# cat /etc/hasher-priv/system
# Systemwide configuration for the hasher-priv(8) helper program.
# See hasher-priv.conf(5) for details.

allowed_mountpoints=/proc,/dev/pts,/dev/shm,/sys
prefix=~:/tmp/.private
[root@VM ~]#
```

   - при изменении вышеуказанных параметров необходимо перезапускать сервис **hasher-privd**:

```
[root@VM ~]# systemctl restart hasher-privd.service
[root@VM ~]#
```

3. При запуске hasher файловая система должна быть указана в опции ***--mountpoints*** или, что то же самое, в ключе ***known_mountpoints*** конфигурационного файла hasher (**~/.hasher/config**):

```
[user@VM ~]$ hsh-shell --mountpoints=/proc
```

4. При сборке пакета файловая система должна быть указана сборочной зависимостью (например, ***BuildReq: /proc***) собираемого пакета, прямой или косвенной (через зависимости сборочных зависимостей пакета).

Важно

Все примеры командной строки, содержащие подсказку **«[root@...]#»**, требуют прав суперпользователя. Получив такие права, обращайтесь с системой осторожнее!

Запустим GDB с программой из примера выше и проследим изменение некоторых параметров:

@builder

```
[builder@localhost ~]$ gdb ex
GNU gdb (GDB) 14.1.0.56.d739d4fd457-alt1 (ALT Sisyphus)
<...>
Reading symbols from ex...
(gdb)
```

Запустим выполнение кода и будем отслеживать значения переменных при выполнении:

@builder

```
(gdb) run
Starting program: /usr/src/ex
<...>
Breakpoint 2, fun1 (c=42) at ex.c:9
9               c += 1;
(gdb) print c
$1 = 42
(gdb) next
10            return fun2(c) + fun2(c*2);
(gdb) continue
Continuing.

Breakpoint 1, fun2 (n=43) at ex.c:5
5               return n*2+1;
(gdb) display n
1: n = 43
(gdb) continue
Continuing.

Breakpoint 1, fun2 (n=86) at ex.c:5
5               return n*2+1;
1: n = 86
(gdb)
```

При работе с исходными текстами программ большого размера удобно просматривать (не покидая GDB) блоки кода вокруг выполняемой строки:

@builder

```
(gdb) list
1       #include <stdio.h>
2       int N = 42;
3
4       int fun2(int n) {
5               return n*2+1;
6       }
7
8       int fun1(int c) {
9               c += 1;
10            return fun2(c) + fun2(c*2);
(gdb)
```

@builder

```
(gdb) delete breakpoints
Delete all breakpoints? (y or n) y
(gdb) cont
Continuing.
260
266
272
278
284
290
296
302
308
314
[Inferior 1 (process 155112) exited normally]
(gdb) quit
[builder@localhost ~]$
```

6.2. Удалённая отладка программ

Иногда нельзя запустить GDB в том же текстовом окне, что и программу для отладки — например, если эта программа использует Ncurses. Однако можно запустить программу под управлением «удаленного» варианта отладчика — GDB server, интерфейс которого доступен по сети или с другого терминала. Подключаться к GDB server можно обычным GDB, отдельно указав расположение удалённого сеанса.

Мы рассмотрим слегка усложнённую ситуацию: попробуем повести сеанс удалённой отладки, пользуясь только пакетами, установленными в окружении hasher. Запустить **gdbserver** в hasher, а потом зайти туда же, чтобы пользоваться GDB, нам не удастся: настройки доступа hasher дают возможность единовременного доступа к окружению только одному процессу зарегистрированного пользователя. Для единовременной работы двух (и более) hasher-пользователей необходимо создать дополнительные окружения с другой парой **builder-rooter**-пользователей.

Для создания другой пары необходимо воспользоваться ключами ***--names*** для именования этих пользователей и ***--names*** для описания дополнительного идентификатора пары (без этого ключа дополнительные пары на одного и того же изначального пользователя не создадутся):

@root

```
[root@VM ~]# id user
uid=1000(user) gid=1000(user)
группы=1000(user),10(wheel),100(users),997(hashman),1001(user_a),1002(user_b),36(
vmusers)

[root@VM ~]# hasher-useradd --names=user_c:user_d --number=2 user
useradd: Warning: missing or non-executable shell '/dev/null'
useradd: Warning: missing or non-executable shell '/dev/null'
Добавление пользователя user в группу user_c
Добавление пользователя user в группу user_d
Добавление пользователя user в группу hashman
hasher-useradd: enabling hasher-privd
Внимание: Отправляется запрос 'systemctl enable hasher-privd.service'.
Synchronizing state of hasher-privd.service with SysV service script with /usr/
lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable hasher-privd
```

```
hasher-useradd: starting hasher-privd

[root@VM ~]# id user
uid=1000(user) gid=1000(user)
группы=1000(user),10(wheel),100(users),997(hashman),1001(user_a),1002(user_b),100
3(user_c),1004(user_d),36(vmusers)
[root@VM ~]#
```

Создание дополнительного окружения аналогично созданию первого: необходимо создать директорию и в ней собрать окружение. Все команды по работе с дополнительными окружениями должны сопровождаться явным указанием директории расположения окружения с помощью ключа ***--workdir*** (команды по умолчанию будут выполнять действия над директорией hasher и окружением в ней):

@**user**

```
[user@VM ~]$ mkdir hasher2
[user@VM ~]$ hsh --init --workdir=hasher2
<...>
[user@VM ~]$
```

Теперь при помощи двух процессов-пользователей (с помощью, например, двух ssh-подключений к машине) будем работать с двумя окружениями. Установим в них необходимые пакеты: *gdbserver* для серверной стороны, *gdb* для клиентской стороны:

@**user**

```
[user@VM ~]$ hsh-install gdbserver
<...>
<13>Jul 16 11:57:37 rpmi: gdbserver-14.1.0.56.d739d4fd457-alt1
sisyphus+338901.300.3.1 1706109034 installed
[user@VM ~]$ hsh-install --workdir=hasher2 gdb
<...>
<13>ul 16 11:58:08 rpmi: gdb-14.1.0.56.d739d4fd457-alt1 sisyphus+338901.300.3.1
1706109034 installed
[user@VM ~]$
```

Для совместной работы клиенту нужно будет подключаться к серверу по сети. Для этого необходимо добавить в hasher доступ в интернет. Для этого необходимо:

- указать разрешение доступа ***export share_network=1***;
- в hasher указать адрес DNS-сервера (например, ***8.8.8.8***, но сгодится и содержимое **/etc/resolv.conf** на хост-системе):

  @**user**

  ```
  [user@VM ~]$ hsh-copy --rooter /etc/resolv.conf /etc/resolv.conf
  [user@VM ~]$ hsh-copy --workdir=hasher2 --rooter /etc/resolv.conf /etc/
  resolv.conf
  ```

Запустим удалённую отладку (не забываем, что для работы GDB необходимо на входе в **hsh-shell** указывать ***--mountpoints=proc***, а для работы сети — ***share_network=1***).

Будем заниматься отладкой той же самой программы:

@**user**: **ex.c**

```
#include <stdio.h>
int N = 42;

int fun2(int n) {
        return n*2+1;
}

int fun1(int c) {
        c += 1;
        return fun2(c) + fun2(c*2);
}

int main(int argc, char *argv[]) {
        int i;
        for(i=0; i <10; i++)
                printf("%d\n", fun1(N+i));
        return 0;
}
```

На сервере для работы должен располагаться исполняемый файл, собранный с отладочными флагами. Текст программы не обязателен:

@user

```
[user@VM ~]$ share_network=1 hsh-shell --mountpoints=/proc
```

@builder

```
[builder@localhost ~]$ cat > ex.c
<текст программы>
[builder@localhost ~]$ cc -g -O0 ex.c -o ex
[builder@localhost ~]$ rm -f ex.c
```

На клиенте достаточно запустить сам GDB, но так как исходные тексты показывает именно клиент, рекомендуется иметь их под рукой:

@user

```
[user@VM ~]$ share_network=1 hsh-shell --mountpoints=/proc --workdir=hasher2
```

@builder2

```
[builder@localhost ~]$ cat > ex.c
<текст программы>
[builder@localhost ~]$
```

Запустим **gdbserver** и подключимся к нему с клиента с помощью **`target remote`**:

@builder

```
[builder@localhost ~]$ gdbserver localhost:5000 ex
Process /usr/src/ex created; pid = 188151
Listening on port 5000
```

@builder2

```
[builder@localhost ~]$ gdb
GNU gdb (GDB) 14.1.0.56.d739d4fd457-alt1 (ALT Sisyphus)
<...>
(gdb) target remote localhost:5000
Remote debugging using localhost:5000
Reading /usr/src/ex from remote target...
<...>
0x00007ffff7fe4e40 in _start () from target:/lib64/ld-linux-x86-64.so.2
(gdb)
```

@builder

```
<...>
Remote debugging from host 127.0.0.1, port 48528
```

@builder2

```
(gdb) b 9
Breakpoint 1 at 0x555555555156: file ex.c, line 9.
(gdb) cont
Continuing.
<...>
Breakpoint 1, fun1 (c=42) at ex.c:9
warning: Source file is more recent than executable.
9               c += 1;
(gdb) list
4       int fun2(int n) {
5               return n*2+1;
6       }
7
8       int fun1(int c) {
9               c += 1;
10            return fun2(c) + fun2(c*2);
11      }
12
13      int main(int argc, char *argv[]) {

(gdb) delete breakpoints
Delete all breakpoints? (y or n) y
(gdb) cont
Continuing.
[Inferior 1 (process 188103) exited normally]
(gdb) quit
[builder@localhost ~]$
```

@builder

```
[builder@localhost ~]$ gdbserver localhost:5000 ex
Process /usr/src/ex created; pid = 188151
Listening on port 5000
Remote debugging from host 127.0.0.1, port 48528
[builder@localhost ~]$
```

Для удобства отладки программ на основе GDB создано множество GUI-приложений (например, плагины к различным IDE). Сам GDB поддерживает представление кода при отладке с помощью флага ***-tui***.

Отдельно упомянем класс ***UI***, интерфейсом для которых выступает браузер. В качестве примера можно рассмотреть Gdbgui и GDBFrontend — python-пакеты, которые достаточно установить в отладочное окружение с помощью **`pip install`**, прав суперпользователя при этом не нужно. После запуска нужно зайти браузером на заданный порт, где будет ожидать крошечный веб-сервер, управляющий GDB.

6.3. Автонастройка hasher

Постоянная настройка hasher не очень удобна в процессе интенсивной работы. Поэтому мы предлагаем использовать bash-сценарий **`hypersh.sh`** для быстрой настройки рабочего окружения.

`Hypersh` — hyper hsh — позволяет одной командой настраивать окружение, ставить дополнительные пакеты, работать с пакетами с исходным кодом.

При отсутствии собранного окружения hypersh первым делом создаст новое чистое окружение (**`hsh --init`**). После этого, если ему был передан в качестве параметра *.src.rpm*-пакет, разворачивает его в окружении, а также ставит все его сборочные зависимости.

Важно

Разворачивание пакета с исходным кодом в hasher **не** пересобирает окружение. То есть если изначально оно было собрано (**`hsh --init`** не отработал) и в нём уже был развёрнут какой-то пакет, второй развернётся в дополнение к первому.

После начинается автоматическая настройка окружения:

- добавляется доступ в интернет из hasher;
- устанавливаются пакеты (пакеты, устанавливающиеся всегда, указаны в переменной ***ADDPACKAGES***, дополнительные пакеты для установки можно указать с ключом ***-p***);
- монтируются специальные файловые системы **`/proc`** и **`/dev/pts`**.

После этого осуществляется вход в **hsh-shell** с указанными правами:

hypersh.sh

```
#!/bin/sh -E
PROG=`basename "$0"`
TEMP=$(getopt -o 'p:rvw:h' --long 'rooter,packages:,verbose,workdir:,help' -n
"$PROG" -- "$@") || exit $?
eval set -- "$TEMP"
unset TEMP

WORKDIR="$HOME/hasher"
SHFLAGS=""
ALLFLAGS=""
MOUNTPOINTS=/proc,/dev/pts
ADDPACKAGES="vim-plugin-spec_alt-ftplugin vim-console tree rpm-utils"

USAGE="$PROG – run hsh-shell with network on and some packages installed

Usage: $PROG [OPTIONS] [SRC-RPM]
```

```
    -p|--packages Additional packages to install
    -r|--rooter   Run shell commands as «rooter» user
    -w|--workdir  Specify hasher hierarchy directory ($WORKDIR)
    -v|--verbose  Print more information
    -h|--help     Print this help

SRC-RPM is source RPM file to build inside hasher.
When installed, it's build dependencies will be installed as well.
"
while true; do
  case "$1" in
    -p|--packages) ADDPACKAGES="$ADDPACKAGES $2"; shift;;
    -r|--rooter) SHFLAGS="$SHFLAGS --rooter";;
    -w|--workdir) WORKDIR="$2"; shift;;
    -v|--verbose) ALLFLAGS="$ALLFLAGS --verbose";;
    -h|--help) echo "$USAGE" >&2; exit 0;;
    --) shift; break;;
  esac
  shift
done

test -d "$WORKDIR/chroot" || hsh --workdir="$WORKDIR" --mountpoints=$MOUNTPOINTS
$ALLFLAGS --init
if [ "$#" = 1 ]; then
  hsh-rebuild --workdir="$WORKDIR" $ALLFLAGS --install-only "$1"
  hsh-run --workdir="$WORKDIR" $ALLFLAGS -- rpm -i /usr/src/in/srpm/`basename
"$1"`
fi

hsh-copy $ALLFLAGS --workdir="$WORKDIR" --rooter /etc/resolv.conf /etc/
resolv.conf
hsh-install $ALLFLAGS --workdir="$WORKDIR" --mountpoints=$MOUNTPOINTS
$ADDPACKAGES
share_network=1 hsh-shell $ALLFLAGS $SHFLAGS --workdir="$WORKDIR" --
mountpoints=$MOUNTPOINTS
```

Глава 7. Сопоставление шаблону

При работе с текстовыми данными одной из важных задач является сопоставление шаблону. Поиск шаблонов встречается постоянно для уточнения и выделения интересующих нас данных. На основании полученных данных можно принимать решение по фильтрации или редактированию материала.

Одним из примитивных языков шаблонов выступает представленный в Shell вариант. Также этот язык шаблонов оформлен в качестве отдельной утилиты glob и библиотечного вызова glob(). Он используется преимущественно для проверок соответствия пути файлов.

Основным языком шаблонов в большинстве задач поиска выступает язык регулярных выражений. Он позволяет описывать существенно более сложные в сравнении с предыдущим языком конструкции — регулярные (автоматные) языки по классификации Хомского — , однако в этой классификации представляет наиболее узкое подмножество. В отличие от шаблонов **glob**, синтаксис регулярных выражений допускает довольно сложные, а временами — трудно понимаемые конструкции.

Классическими утилитами, использующими регулярные выражения, являются утилита поиска по шаблону grep и потоковый редактор sed. Они поддерживают работу как с классическими регулярными выражениями, так и с расширенными.

При разработке используются библиотеки, предназначенные для работы с регулярными выражениями. В стандартной библиотеке языка Си есть встроенная обработка регулярных выражений. Существуют также диалекты регулярных выражений, выходящие за рамки автоматных грамматик (например, за счёт задания контекста появления шаблона) — таковы, например, регулярные выражения в языках Perl и Python.

7.1. Пример использования регулярных выражений в СИ

Соберём пакет, программы в котором будут выполнять роль утилиты **grep**. Для сборки воспользуемся Gear — утилитой, которая позволяет хранить авторские исходные тексты, спецификацию сборки и дополнительные файлы (например, исправления исходников или документацию, относящуюся к особенностям сборки для ALT) в едином каталоге под управлением системы контроля версий Git:

@user

```
[user@VM ~]$ mkdir regex-pkg
[user@VM ~]$ cd regex-pkg/
[user@VM regex-pkg]$ git init
<...>
Initialized empty Git repository in /home/user/regex-pkg/.git/
[user@VM regex-pkg]$
```

В дальнейшем, поскольку вся сборка пакетов будет осуществляться с помощью Gear, данный блок с настройкой пустого git-репозитория будет опускаться:

@user: **regex-pkg/regex-no-bags.c**

```
#include <stdio.h>
#include <stdlib.h>
#include <regex.h>

int main(int argc, char** argv) {
	char *text;
	size_t size = 0;
	int len;
	regex_t regex;

	regcomp(&regex, argv[1], REG_EXTENDED);

	for (text = NULL; (len = getline(&text, &size, stdin)) != -1; free(text),
text = NULL) {
		text[len - 1] = 0;
		if (regexec(&regex, text, 0, NULL, 0) == 0)
			puts(text);
	}

	regfree(&regex);
	return 0;
}
```

Подробнее обсудим ключевые функции для работы с регулярными выражениями:

- **regcomp** на основе полученного шаблона строит структуру регулярного выражения для поиска подходящих шаблонов. Флаг ***REG_EXTENDED*** указывает на использование расширенных регулярных выражений;
- **regexec** находит в указанной строке шаблон регулярного выражения по принципу «Самый левый, самый длинный»;
- **regfree** освобождает структуру, выделенную в динамической памяти;

@**user**: **`regex-pkg/regex-bags.c`**

```
#include <stdio.h>
#include <stdlib.h>
#include <regex.h>

#define MAXGR 10

int main(int argc, char** argv) {
       char *text;
       size_t size = 0;
       int len;
       regex_t regex;
       regmatch_t bags[MAXGR];


       regcomp(&regex, argv[1], REG_EXTENDED);

       for (text = NULL; (len = getline(&text, &size, stdin)) != -1;
free(text), text = NULL) {
               text[len - 1] = '\0';
               if (regexec(&regex, text, MAXGR, bags, 0) == 0) {
                       puts(text);
                       for(int i = 1; i <screen MAXGR && bags[i].rm_so >= 0;
i++) {
                               int begin = bags[i].rm_so;
                               int end = bags[i].rm_eo;
                               printf("Bag %d: position %d - %d\t%.*s\n",
                                               i, begin, end, end - begin,
text + begin);
                       }
               }
       }
       regfree(amp;regex);
       return 0;
}
```

- дополнительные параметры в **regexec** позволяют не только найти подходящую под шаблон подстроку, но и разбить её на «карманы» соответственно структуре регулярного выражения;
- каждый карман описывается двумя параметрами: индексом начала и индексом конца кармана в исходной строке.

`Makefile` и spec-файл не отличаются никакими специальными командами или директивами:

@**user**: **`regex-pkg/Makefile`**

```
GENS = regex-no-bags regex-bags
TRASH = *.o *~ o.*
CFLAGS = -Wall
CC = cc

all:    regex-no-bags regex-bags

clean:
	rm -f $(TRASH)

distclean:      clean
	rm -f $(GENS)
```

@user: **`regex-pkg/regex-pkg.spec`**

```
Name: regex-pkg
Version: 1.0
Release: alt1

Summary: Test pkg with regex

License: GPL-3.0-or-later
Group: Development/Other

Source0: %name-%version.tar.gz

%description
This is a small testing package with Regular Expressions

%prep
%setup

%build
%make_build

%install
install -D regex-no-bags %buildroot%_bindir/regex-no-bags
install -D regex-bags %buildroot%_bindir/regex-bags

%files
%_bindir/*

%changelog
* Tue Jul 15 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial Build
```

Рассмотрим структуру gear-репозитория:

@user

```
[user@VM regex-pkg]$ tree
.
├── Makefile
├── regex-bags.c
├── regex-no-bags.c
```

```
└── regex-pkg.spec

1 directory, 4 files
[user@VM regex-pkg]$
```

Поскольку в gear-репозитории исходные тексты программ хранятся в нескомпонованном виде, необходимо указать правила компоновки и сжатия для будущей сборки пакета. Данные о компоновке описываются в файле **`.gear/rules`**. Правила оформления **`.gear/rules`** можно найти в справочнике по gear:

@user: **`.gear/rules`**

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

Проведём сборку с помощью hasher. Для этого необходимо занести все изменения в git и запустить hasher:

@user

```
[user@VM regex-pkg]$ git add
[user@VM regex-pkg]$ gear-commit
[master (root-commit) 0080d40] 1.0-alt1
5 files changed, 99 insertions(+)
create mode 100644 .gear/rules
create mode 100644 Makefile
create mode 100644 regex-bags.c
create mode 100644 regex-no-bags.c
create mode 100644 regex-pkg.spec
[user@VM regex-pkg]$ gear-hsh
<...>
[user@VM regex-pkg]$ cd/
[user@VM ~]$ ls hasher/repo/x86_64/RPMS.hasher/ | grep regex
regex-pkg-1.0-alt1.x86_64.rpm
regex-pkg-debuginfo-1.0-alt1.x86_64.rpm
[user@VM ~]$
```

Соберём пустое окружение hasher и установим туда полученный пакет:

@user

```
[user@VM ~]$ hsh --init
<...>
[user@VM ~]$ cp hasher/repo/x86_64/RPMS.hasher/regex-pkg-1.0-alt1.x86_64.rpm
hasher/ch
root/.in/
[user@VM ~]$ hsh-shell --rooter
```

@rooter

```
[root@localhost .in]# rpm -i regex-pkg-1.0-alt1.x86_64.rpm
<13>Jul 15 15:03:44 rpm: regex-pkg-1.0-alt1 1752591330 installed
[root@localhost .in]#
[root@localhost .in]# which regex-bags
/usr/bin/regex-bags
[root@localhost .in]# which regex-no-bags
/usr/bin/regex-no-bags
```

```
[root@localhost .in]# cal | regex-no-bags "(2)(.*)(3)"
      1  2  3  4  5
20 21 22 23 24 25 26
27 28 29 30 31
[root@localhost .in]# cal | regex-bags "(2)(.*)(3)"
      1  2  3  4  5
Bag 1: position 10 - 11 2
Bag 2: position 11 - 13
Bag 3: position 13 - 14 3
20 21 22 23 24 25 26
Bag 1: position 0 - 1   2
Bag 2: position 1 - 10  0 21 22 2
Bag 3: position 10 - 11 3
27 28 29 30 31
Bag 1: position 0 - 1   2
Bag 2: position 1 - 12  7 28 29 30
Bag 3: position 12 - 13 3
[root@localhost .in]#
```

Глава 8. Трассировка вызовов

8.1. Управление вызовами

При разработке программы необходимо учитывать возможное поведение системы при ошибках. Отслеживание исполнения инструкций программы, выявление проблем и непосредственный контроль исполнения кода — всё это называется *трассировка выполнения*. Более подробно об этом можно прочитать в источниках, здесь же будет подробнее рассмотрена *трассировка системных вызовов*, направленная на отслеживание выполнения обращений программы к ОС.

Ключевым системным вызовом, позволяющим проводить трассировку, является ptrace. На его основе работает большинство (если не все) отладчиков (включая GDB). Также на его основе работают команды по предзагрузке библиотек.

Один из классических инструментов трассировки системных вызовов на основе ptrace — проект Strace. Рассмотрим некоторые возможности утилиты:

- непосредственно трассировка — показ всех произведённых системных вызовов (как видим, даже простейшая утилита date делает их немало):

  @user

```
[user@VM ~]$ date
Пт 18 июл 2025 11:52:32 MSK
[user@VM ~]$ strace date
execve("/usr/bin/date", ["date"], 0x7ffdb8d7cbe0 /* 79 vars */) = 0
brk(NULL)                               = 0x55b7781f6000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или
каталога)
<...>
close(1)                        = 0
close(2)                        = 0
exit_group(0)                           = ?
+++ exited with 0 +++
[user@VM ~]$ strace date |& wc
   116     715    8509
[user@VM ~]$
```

- фильтрация трассировки по конкретным системным вызовам:

@user

```
[user@VM ~]$ strace -e brk date
brk(NULL)                                = 0x55fdaf32f000
brk(NULL)                                = 0x55fdaf32f000
brk(0x55fdaf350000)                      = 0x55fdaf350000
Пт 18 июл 2025 11:54:10 MSK
+++ exited with 0 +++
[user@VM ~]$
```

- трассировка системных вызовов, связанных с конкретными объектами файловой системы:

@user

```
[user@VM ~]$ strace -P /usr/lib/locale/ru_RU.utf8/LC_MESSAGES/SYS_LC_MESSAGES
date
openat(AT_FDCWD, "/usr/lib/locale/ru_RU.utf8/LC_MESSAGES/SYS_LC_MESSAGES",
O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=70, ...}) = 0
mmap(NULL, 70, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc1adbc6000
close(3)                                 = 0
Пт 18 июл 2025 11:56:14 MSK
+++ exited with 0 +++
[user@VM ~]$
```

- получение более полной диагностики:

@user

```
[user@VM ~]$ strace -e fstat date
fstat(3, {st_mode=S_IFREG|0644, st_size=43543, ...}) = 0
fstat(3, {st_mode=S_IFREG|0755, st_size=2012192, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=341, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=27012, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=23, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=52, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=165, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=62, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=34, ...}) = 0
fstat(3, {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=70, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=294, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=2586930, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=3416, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=54, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=360460, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=908, ...}) = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=908, ...}) = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
Пт 18 июл 2025 11:57:54 MSK
+++ exited with 0 +++
[user@VM ~]$ strace -e fstat date |& wc
    21     125    1109
[user@VM ~]$ strace -e fstat -v date |& wc
    21     543    7702
[user@VM ~]$
```

- вывод таблицы выполнения системных вызовов:

  @user

```
[user@VM ~]$ strace -w -c date
Пт 18 июл 2025 11:59:58 MSK
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
24,36    0,001554          50        31        14 openat
17,81    0,001136          54        21           mmap
13,39    0,000854         854         1           execve
13,25    0,000845          44        19           close
12,34    0,000787          41        19           fstat
 3,59    0,000229          76         3           mprotect
 2,36    0,000150         150         1         1 access
 2,00    0,000128          42         3           brk
 1,94    0,000124          41         3           read
 1,58    0,000101          50         2           pread64
 1,33    0,000085          85         1           munmap
 0,85    0,000054          54         1           write
 0,85    0,000054          54         1           futex
 0,68    0,000043          43         1           arch_prctl
 0,65    0,000041          41         1           getrandom
 0,62    0,000040          39         1           prlimit64
 0,61    0,000039          38         1           rseq
 0,60    0,000038          38         1           set_robust_list
 0,59    0,000038          37         1           set_tid_address
 0,59    0,000038          37         1           lseek
------ ----------- ----------- --------- --------- ----------------
100,00    0,006379          56       11315 total
[user@VM ~]$
```

8.1. Управление вызовами

Поскольку ptrace отлавливает системные вызовы в момент их вызова и в момент выхода из них, возможно подменять результат работы вызовов вручную. Для подмены предпочтительнее всего использовать ключ ***--inject***, параметры ключа доступны в man.

Соберём пакет с использованием в директиве ***%check*** проверки на основе **strace**. Для начала напишем программу с *достаточным количеством* системных вызовов:

1. Программа последовательно создаёт 10 файлов, пишет в них с помощью системного вызова ***write()***, читает из них с помощью системного вызова ***read()***, после чего выводит сообщение об успешной итерации в файл вывода:

   - все файлы являются временными и создаются в директории **/tmp**;

   - некорректная работа системных вызовов сопровождается записью информационных сообщений в файл вывода (что, вообще говоря, также сопровождается системными вызовами ☺);

2. Добавлена обработка двух пользовательских сигналов: ***SIGUSR1*** выполняет исключительно информационную функцию, ***SIGUSR2*** по достижении некоторого количества получений сигнала включает механизм завершения программы:

   @user: **strace-pkg/strace-pkg-1.0.c**

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <signal.h>

volatile int count = 0;

volatile int output_fd;
volatile int fd;


void hdlr(int sig) {
       switch(sig) {
                case SIGUSR1:
                        dprintf(output_fd, "SIGUSR1 arrived: 42\n");
                        break;
                case SIGUSR2:
                        dprintf(output_fd, "SIGUSR2 arrived %d time(s)\n", +
+count);
                        if (count == 3) {close(fd); _exit(0);}
                        break;
       }
}

int main(void) {
       output_fd = open("/tmp/strace-pkg/output", O_TRUNC | O_CREAT |
O_WRONLY);
       sigaction(SIGUSR1, &(struct sigaction) { .sa_handler =
hdlr, .sa_flags = SA_RESTART}, NULL);
       sigaction(SIGUSR2, &(struct sigaction) { .sa_handler =
hdlr, .sa_flags = SA_RESTART}, NULL);

       for (int i = 0; i < 10; i++) {
                char filename[100] = {}, string[100] = {};
                snprintf(filename, 100, "/tmp/strace-pkg/strace-file-%d", i);
                fd = open(filename, O_TRUNC | O_CREAT | O_RDWR, 0666);
                if (write(fd, "Hello\n\0", strlen("Hello\n\0")) !=
strlen("Hello\n\0")) {
                        dprintf(output_fd, "Bad writing\n");
                        close(fd);
                        _exit(0);
                }
                if (read(fd, string, strlen(string)) < 0) {
                        dprintf(output_fd, "Bad reading\n");
                        close(fd);
                        _exit(0);
                }
                dprintf(output_fd, "strace-file-%d success\n", i);
                close(fd);
       }

       return 0;
}
```

При работе **strace** будет отлавливать исполнение системных вызовов, с помощью ***--inject*** будет проводиться ручная замена возвращаемых значений, а также будут отправляться сигналы. Программа будет реагировать на приходящие ответы и вести себя согласно описанной структуре.

В **Makefile** добавим описание команд **strace --inject**:

@**user**: **strace-pkg/Makefile**

```
NAME=strace-pkg
VERSION=1.0

PROG=./$(NAME)-$(VERSION)
LOGFILE=/tmp/$(NAME)/log
OUTPUTFILE=/tmp/$(NAME)/output

GENS = /tmp/$(NAME)/
TRASH = *.o *~ o.*
CFLAGS = -Wall
CC = cc

all:    $(PROG)

check:  prep $(PROG)
        strace -o $(LOGFILE) $(PROG)
        diff $(OUTPUTFILE) ./check/test-0
        strace --inject=read:error=EBADF:when=5 -o $(LOGFILE) $(PROG)
        diff $(OUTPUTFILE) ./check/test-1
        strace --inject=close:retval=0:signal=SIGUSR2:when=4+ \
               --inject=read:retval=6:signal=SIGUSR1:when=3 \
               -o $(LOGFILE) $(PROG)
        diff $(OUTPUTFILE) ./check/test-2
        strace --inject=rt_sigaction:retval=1:when=1 \
               --inject=read:retval=0:signal=SIGUSR1:when=3 \
               -o $(LOGFILE) $(PROG) || true
        grep +++ $(LOGFILE) >> $(OUTPUTFILE)
        diff $(OUTPUTFILE) ./check/test-3

prep:
        mkdir /tmp/$(NAME)
        touch $(LOGFILE) $(OUTPUTFILE)

clean:
        rm -rf $(TRASH)

distclean:      clean
        rm -rf $(GENS)
```

Секция ***check*** состоит из четырех тестов, описанных с помощью **strace --inject**:

1. **Нулевой тест** без дополнительных действий над системными вызовами. Программа должна завершиться корректно, а в выводе должна быть информация о корректном завершении всех десяти итераций;

2. **Проверка обработки системного вызова *read***: на его пятый вызов будет возвращена ошибка **EBADF** (некорректный файловый дескриптор), программа должна будет вывести сообщение об ошибке чтения и корректно завершиться;

3. **Обработка сигналов**: после третьего вызова ***read*** в систему должен будет придти сигнал ***SIGUSR1***, что должно будет отобразиться на выводе, а начиная с четвёртого и далее вызовов ***close*** в систему будут поступать сигналы ***SIGUSR2***, которые должны будут привести к досрочному завершению программы;

4. **Проверка на некорректную обработку сигнала**: системный вызов ***sigaction*** для ***SIGUSR1*** не сработает, и по приходу сигнала после третьего ***read*** программа аварийно завершится, что будет отражено в записях ***strace***.

Для проверки тестов добавим файлы с правильным выводом, который ожидается по завершении тестов.

@**user**: **strace-pkg/check/test-0** — корректная работа программы:

```
strace-file-0 success
strace-file-1 success
strace-file-2 success
strace-file-3 success
strace-file-4 success
strace-file-5 success
strace-file-6 success
strace-file-7 success
strace-file-8 success
strace-file-9 success
```

@**user**: **sstrace-pkg/check/test-1** — ***read***-тест. Заметим, что поскольку до исполнения кода программы производится чтение динамической библиотеки **libc** (с помощью того же системного вызова ***read***), ошибка должна будет появиться не на пятом, а на четвёртом вызове ***read*** программы:

```
strace-file-0 success
strace-file-1 success
strace-file-2 success
Bad reading
```

@**user**: **strace-pkg/check/test-2** — тест сигналов. Аналогичная особенность со «сдвигом» вызовов происходит и с ***close***: до выполнения кода программы открываются (и, соответственно, закрываются) файлы кеша и библиотеки **libc**:

```
strace-file-0 success
SIGUSR1 arrived: 42
strace-file-1 success
SIGUSR2 arrived 1 time(s)
strace-file-2 success
SIGUSR2 arrived 2 time(s)
strace-file-3 success
SIGUSR2 arrived 3 time(s)
```

@**user**: **strace-pkg/check/test-3** — тест «необработки» сигнала. Для отслеживания ошибок необходимо будет объединить данные из файла вывода и информацию о завершении процесса:

```
strace-file-0 success
+++ killed by SIGUSR1 +++
```

Файл спецификации получается довольно лаконичным:

@**user**: **strace-pkg/strace-pkg.spec**

```
Name: strace-pkg
Version: 1.0
Release: alt1

Summary: Test pkg with strace

License: GPL-3.0-or-later
Group: Development/Other

Source0: %name-%version.tar.gz

%description
This is a small testing package with Strace check section

%prep
%setup

%build
%make_build

%install
install -D %name-%version %buildroot%_bindir/%name-%version

%check
make check

%files
%_bindir/*

%changelog
* Fri Jul 18 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial build
```

@**user**: **.gear/rules**

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

@**user**:

```
[user@VM strace-pkg]$ tree
.
├── check
│   ├── test-0
│   ├── test-1
│   ├── test-2
│   └── test-3
├── Makefile
├── strace-pkg-1.0.c
└── strace-pkg.spec

2 directories, 7 files
[user@VM strace-pkg]$ git add
[user@VM strace-pkg]$ gear-commit
[master (root-commit) 54d15cc] 1.0-alt1
8 files changed, 144 insertions(+)
create mode 100644 .gear/rules
create mode 100644 Makefile
create mode 100644 check/test-0
```

```
create mode 100644 check/test-1
create mode 100644 check/test-2
create mode 100644 check/test-3
create mode 100644 strace-pkg-1.0.c
create mode 100644 strace-pkg.spec
[user@VM strace-pkg]$
```

Пример сборки пакета с секцией ***%check***:

```
[user@VM strace-pkg]$ gear-hsh --lazy
<...>
Wrote: /usr/src/in/srpm/strace-pkg-1.0-alt1.src.rpm (w1.gzdio)
Installing strace-pkg-1.0-alt1.src.rpm
<...>
Executing(%check): /bin/sh -e /usr/src/tmp/rpm-tmp.64947
+ umask 022
+ /bin/mkdir -p /usr/src/RPM/BUILD
+ cd /usr/src/RPM/BUILD
+ cd strace-pkg-1.0
+ make check
make: Entering directory '/usr/src/RPM/BUILD/strace-pkg-1.0'
mkdir /tmp/strace-pkg
touch /tmp/strace-pkg/log /tmp/strace-pkg/output
strace -o /tmp/strace-pkg/log ./strace-pkg-1.0
diff /tmp/strace-pkg/output ./check/test-0
strace --inject=read:error=EBADF:when=5 -o /tmp/strace-pkg/log ./strace-pkg-1.0
diff /tmp/strace-pkg/output ./check/test-1
strace --inject=close:retval=0:signal=SIGUSR2:when=4+ \
       --inject=read:retval=6:signal=SIGUSR1:when=3 \
       -o /tmp/strace-pkg/log ./strace-pkg-1.0
diff /tmp/strace-pkg/output ./check/test-2
strace --inject=rt_sigaction:retval=1:when=1 \
       --inject=read:retval=0:signal=SIGUSR1:when=3 \
       -o /tmp/strace-pkg/log ./strace-pkg-1.0 || true
cat /tmp/strace-pkg/log | grep +++ >> /tmp/strace-pkg/output
diff /tmp/strace-pkg/output ./check/test-3
make: Leaving directory '/usr/src/RPM/BUILD/strace-pkg-1.0'
+ exit 0
<...>
Wrote: /usr/src/RPM/SRPMS/strace-pkg-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/strace-pkg-1.0-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/strace-pkg-debuginfo-1.0-alt1.x86_64.rpm
(w2.lzdio)
0.99user 1.29system 0:06.08elapsed 37%CPU (0avgtext+0avgdata 24428maxresident)k
672inputs+1168outputs (0major+142115minor)pagefaults 0swaps
[user@VM strace-pkg]$
```

Глава 9. Сборочные зависимости

9.1. Autotools

Использовавшийся до этого инструмент автосборки GNU Make показывает удобство сборки, однако для более тонкой настройки, проверки зависимостей, опциональности сборки необходим некоторый высокоуровневый инструмент с простым интерфейсом, использующий **make** в качестве сборочного компонента. Данную задачу решает семейство инструментов autotools, позволяющее автоматически искать, настраивать и параметризировать сборочные зависимости, а также обрабатывать тесты и, в целом, поддерживать рабочее окружение.

На примере сборки пакета разберёмся с тонкостями работы этого набора инструментов. Рассмотрим Ncurses-программу, считывающую символы и выводящую их на экран. Добавим в неё условные макросы: ***DX*** для размера отступа и ***KEYPAD*** для режима определения функциональных клавиш. Также заметим наличие некоего конфигурационного файла **`config.h`**, речь о котором пойдёт чуть позже:

@**user**: **`autoenv-pkg/src/prog.cs`**

```
#include <ncurses.h>
#include <unistd.h>
#include <error.h>
#include "config.h"

#ifndef DX
#define DX 3
#endif

int main(void) {
        WINDOW *win;
        int c = 0;

        if (!isatty(0))
                error(1, 0, "Not a terminal");
        initscr();
        noecho();
        cbreak();
        printw("%s-%s (%s)", PACKAGE_NAME, PACKAGE_VERSION, PACKAGE_BUGREPORT);
        refresh();

        win = newwin(LINES-2*DX, COLS-2*DX, DX, DX);
#ifdef KEYPAD
        keypad(win, TRUE);
#else
        keypad(win, FALSE);
#endif
        scrollok (win, TRUE);
        do {
                wprintw(win, "  Key: %d, Name: %s\n", c, keyname(c));
                box(win, 0, 0);
                wrefresh(win);
        } while((c = wgetch(win)) != 27);

        endwin();
        return 0;
}
```

Для начала работы необходимо воспользоваться утилитой autoscan. Она генерирует профиль на основе имеющихся файлов в директории с исходными текстами, затем добавляет некоторые зависимости в конфигурационный файл **`configure.scan`**. После использования утилиты получается заготовка профиля для итоговой сборки:

@**user**: **`autoenv-pkg/configure.scan`**

```
#                                               -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.

AC_PREREQ([2.72])
AC_INIT([FULL-PACKAGE-NAME], [VERSION], [BUG-REPORT-ADDRESS])
AC_CONFIG_SRCDIR([src/prog.c])
AC_CONFIG_HEADERS([config.h])

# Checks for programs.
AC_PROG_CC

# Checks for libraries.

# Checks for header files.
AC_CHECK_HEADERS([unistd.h])

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.
AC_FUNC_ERROR_AT_LINE

AC_OUTPUT
```

Основные изменения, внесённые профиль:

- убрана зависимость по версии **autoconf**;
- поскольку основным сборочным инструментом будет являться **make**, добавлены зависимости для него (***`AM_INIT_AUTOMAKE`*** и указание ***`Makefile`*** в ***`AC_CONFIG_FILES`***);
- добавлена зависимость на библиотеку Ncurses. Для этого используется макрос ***`AC_CHECK_LIB`***, который генерирует временный файл с текстовым кодом, использующим указанную нами функцию исследуемой библиотеки (в нашем случае **`initscr`**). Для проверки наличия заголовочного файла библиотеки используется ***`AC_CHECK_HEADERS`***;
- добавлены два дополнительных ключа сборки: без параметров для ***`KEYPAD`*** и с параметром для ***`DX`***.

Следующий использующийся инструмент — autoheader. На основе **`confugire.ac`** он генерирует прототип общего заголовочного файла **`config.h.in`**. Здесь описаны все макросы, которые будут проверяться и обрабатываться при сборке программы:

@**user**: **`autoenv-pkg/config.h.in`**

```
/* config.h.in.  Generated from configure.ac by autoheader.  */

/* Frame width */
#undef DX

/* Define to 1 if you have the <inttypes.h> header file. */
#undef HAVE_INTTYPES_H
```

```
/* Define to 1 if you have the 'ncurses' library (-lncurses). */
#undef HAVE_LIBNCURSES

/* Define to 1 if you have the <ncurses.h> header file. */
#undef HAVE_NCURSES_H

/* Define to 1 if you have the <stdint.h> header file. */
#undef HAVE_STDINT_H

/* Define to 1 if you have the <stdio.h> header file. */
#undef HAVE_STDIO_H

/* Define to 1 if you have the <stdlib.h> header file. */
#undef HAVE_STDLIB_H

/* Define to 1 if you have the <strings.h> header file. */
#undef HAVE_STRINGS_H

/* Define to 1 if you have the <string.h> header file. */
#undef HAVE_STRING_H

/* Define to 1 if you have the <sys/stat.h> header file. */
#undef HAVE_SYS_STAT_H

/* Define to 1 if you have the <sys/types.h> header file. */
#undef HAVE_SYS_TYPES_H

/* Define to 1 if you have the <unistd.h> header file. */
#undef HAVE_UNISTD_H

/* ESC-sequences recognition */
#undef KEYPAD

/* Define to the address where bug reports for this package should be sent. */
#undef PACKAGE_BUGREPORT

/* Define to the full name of this package. */
#undef PACKAGE_NAME

/* Define to the full name and version of this package. */
#undef PACKAGE_STRING

/* Define to the one symbol short name of this package. */
#undef PACKAGE_TARNAME

/* Define to the home page for this package. */
#undef PACKAGE_URL

/* Define to the version of this package. */
#undef PACKAGE_VERSION

/* Define to 1 if all of the C89 standard headers exist (not just the ones
   required in a freestanding environment). This macro is provided for
   backward compatibility; new code need not use it. */
#undef STDC_HEADERS
```

Следующие два инструмента — aclocal и automake. Утилита **aclocal** занимается переопределением процедур поиска и инструментов **automake** для конкретной системы. Утилита **automake** генерирует на основе прототипа **Makefile.am** специальный **Makefile.in**-файл для обработки **autoconf**-ом (такие **.in**-файлы обычно добавляются в макрос ***AC_CONFIG_FILES()***). При работе утилиты также используется ключ ***--add-missing***, с которым она автоматически устанавливает необходимые для работы исполняемые файлы, библиотеки макросов и прочую технологическую оснастку в каталог сборки.

Прототип пишется самостоятельно исходя из задач сборки. Это фактически **Makefile**, в который большая часть рецептов подставляется автоматически в процессе работы **automake**:

@user: **autoenv-pkg/Makefile.am**

```
            CFLAGS = -Wall -O0 -g

bin_PROGRAMS=autoenv

autoenv_SOURCES=src/prog.c

TESTS=isterm.sh

isterm.sh:
	echo 'test "`./autoenv < /dev/null 2>&1`" = "./autoenv: Not a terminal"' >
$@
	chmod +x $@

gitclean:
	git clean -df
```

В нашем генераторе присутствует:

- один итоговый исполняемый файл, исходники для которого лежат в отдельной директории **src**;

Важно

Для описания исходников используется конструкция вида ***<имя исполняемого файла только из alnum-символов>_SOURCES***; имена исполняемых файлов могут быть перечислены через пробел в строке ***bin_PROGRAMS=***

- описан список тестов (из одного теста). Тест — это произвольный исполняемый файл; если его выполнение завершится с ненулевым кодом ошибки, тест считается непройденным.

Для объединения всех полученных профилей в единый сборочный сценарий используется утилита autoconf. Она генерирует основной исполняемый сборочный сценарий **configure**, который поддерживает разные параметры сборки, в том числе и добавленные разработчиком (например, ***--enable-keypad***). При выполнении **configure** собираются все итоговые файлы сборки. К ним относятся, например, итоговый **Makefile** и заполненный **config.h**, описанный в заголовках исходного кода программы. После выполнения **configure** он заполнен всеми данными, необходимыми для сборки:

@user: **autoenv-pkg/config.h**

```
/* config.h.  Generated from config.h.in by configure.  */
/* config.h.in.  Generated from configure.ac by autoheader.  */

/* Frame width */
#define DX 20

/* Define to 1 if you have the <inttypes.h> header file. */
#define HAVE_INTTYPES_H 1

/* Define to 1 if you have the 'ncurses' library (-lncurses). */
#define HAVE_LIBNCURSES 1

/* Define to 1 if you have the <ncurses.h> header file. */
#define HAVE_NCURSES_H 1

/* Define to 1 if you have the <stdint.h> header file. */
#define HAVE_STDINT_H 1

/* Define to 1 if you have the <stdio.h> header file. */
#define HAVE_STDIO_H 1

/* Define to 1 if you have the <stdlib.h> header file. */
#define HAVE_STDLIB_H 1

/* Define to 1 if you have the <strings.h> header file. */
#define HAVE_STRINGS_H 1

/* Define to 1 if you have the <string.h> header file. */
#define HAVE_STRING_H 1

/* Define to 1 if you have the <sys/stat.h> header file. */
#define HAVE_SYS_STAT_H 1

/* Define to 1 if you have the <sys/types.h> header file. */
#define HAVE_SYS_TYPES_H 1

/* Define to 1 if you have the <unistd.h> header file. */
#define HAVE_UNISTD_H 1

/* ESC-sequences recognition */
/* #undef KEYPAD */

/* Define to the address where bug reports for this package should be sent. */
#define PACKAGE_BUGREPORT "UsamG1t"

/* Define to the full name of this package. */
#define PACKAGE_NAME "autoenv-pkg"

/* Define to the full name and version of this package. */
#define PACKAGE_STRING "autoenv-pkg 1.0"

/* Define to the one symbol short name of this package. */
#define PACKAGE_TARNAME "autoenv-pkg"

/* Define to the home page for this package. */
#define PACKAGE_URL ""

/* Define to the version of this package. */
#define PACKAGE_VERSION "1.0"
```

```
/* Define to 1 if all of the C89 standard headers exist (not just the ones
   required in a freestanding environment). This macro is provided for
   backward compatibility; new code need not use it. */
#define STDC_HEADERS 1
```

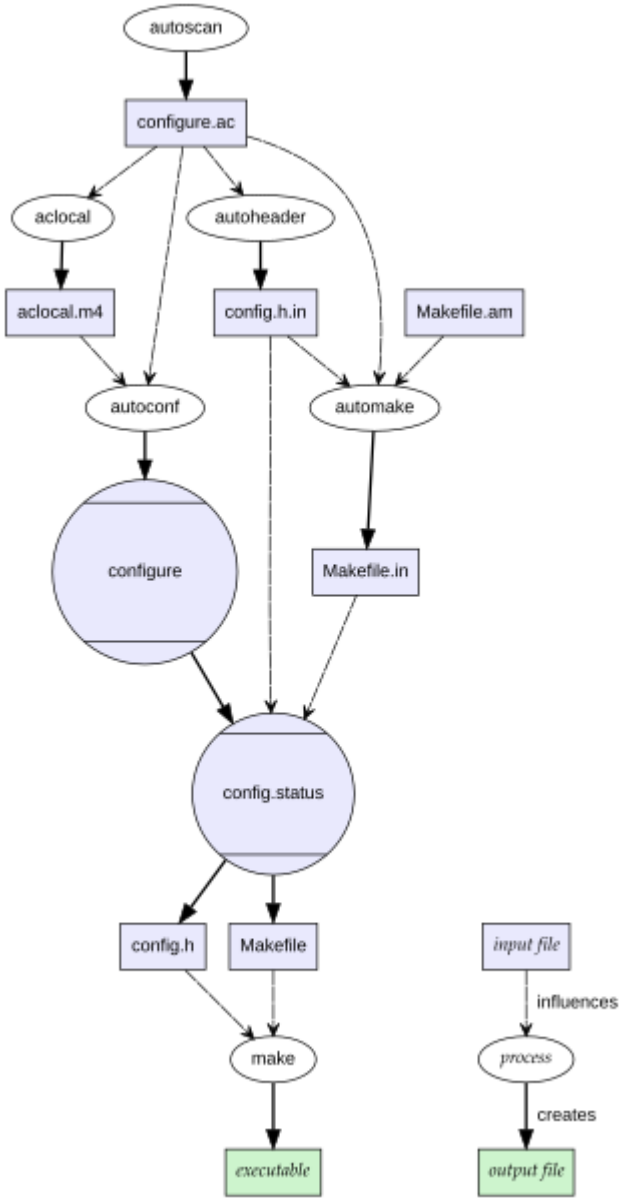
После исполнения **`configure`** можно запускать **make**, который произведёт итоговую сборку самой программы с учётом всех настроек и зависимостей.

9.2. Авто-автосборка

Общий поток автосборочного окружения одинаков:

- единожды используется **autoscan** для создания базового профиля;
- описывается **`configure.ac`**;
- вызываются **autoheader** для генерации **`config.h.in`** и **aclocal** для подготовки к работе **automake**;
- описывается make-генератор **`Makefile.am`**;
- вызывается **automake** ***`--add-missing`*** для подготовки **`Makefile.in`**;
- вызывается **autoconf** для генерации основного конфигурационного файла **`configure`**.

При уже описанных **`configure.ac`** и **`Makefile.am`** и лишь небольшом редактировании этих файлов подготовка к сборке представляет из себя последовательный вызов **aclocal** → **autoconf** → **autoheader** → **automake --add-missing**. Для того чтобы делать эти действия за один раз используется утилита autoreconf с ключами ***`-fisv`***. Весь путь по созданию конфигурационного окружения показан здесь:



В spec-файле для директивы ***%build*** есть даже специальный макрос ***%autoreconf***, буквально разворачивающийся в команду **autoreconf -fisv**. После его исполнения необходимо лишь вызвать **configure** и завершить сборку с помощью **make**:

@user: **autoenv-pkg/autoenv-pkg.spec**

```
Name: autoenv-pkg
Version: 1.0
Release: alt1

Summary: Test pkg with autotool

License: GPL-3.0-or-later
Group: Development/Other

Source0: %name-%version.tar.gz
BuildRequires: libncurses-devel


%description
This is a small testing package, builded by autotools
```

```
%prep
%setup

%build
%autoreconf
%configure
%make_build

%install
%makeinstall_std

%check
make check

%files
%_bindir/*

%changelog
* Mon Jul 21 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial build
```

Макрос ***%makeinstall_std*** разворачивается в **`make install`** с параметрами; ни цели ***`install`***, ни цели ***`check`*** в прототипе **`Makefile.am`** нет: их (и десятки других рецептов) добавляет в конечный **Makefile** сценарий **`configure`**.

Поскольку для корректной сборки конфигурационного окружения достаточно лишь двух файлов — **`configure.ac`** и **`Makefile.am`** — gear-репозиторий может выглядеть достаточно компактным. На практике часть промежуточных файлов генерации не считаются такими в рамках разработки. Обычно в итоговый набор файлов входит всё, что необходимо для работы основного конфигурационного сценария **`configure`**. Однако мы остановимся на минимальном рабочем варианте:

@user

```
[user@VM autoenv-pkg]$ tree . .gear
.
├── autoenv-pkg.spec
├── configure.ac
├── Makefile.am
└── src
    └── prog.c
.gear
└── rules

3 directories, 5 files
[user@VM autoenv-pkg]$
```

Рассмотрим информационный вывод при сборке пакета:

@user

```
[user@VM autoenv-pkg]$ gear-hsh
<...>
Wrote: /usr/src/in/srpm/autoenv-pkg-1.0-alt1.src.rpm (w1.gzdio)
Installing autoenv-pkg-1.0-alt1.src.rpm
<...>
```

При выполнении секции **%*build*** макрос **%*autoreconf***, действительно, разворачивается в команду пересборки **configure**:

```
Executing(%build): /bin/sh -e /usr/src/tmp/rpm-tmp.51756
+ umask 022
+ /bin/mkdir -p /usr/src/RPM/BUILD
+ cd /usr/src/RPM/BUILD
+ cd autoenv-pkg-1.0
+ autoreconf -fisv
<...>
```

Подробнее остановимся на самом исполнении **configure** при сборке. В макросе к нему добавляется множество флагов сборки, указывающих на местоположение исходников, библиотек и т.д:

```
+ ./configure --build=x86_64-alt-linux --host=x86_64-alt-linux --prefix=/usr --
exec-prefix=/usr --bindir=/usr/bin --sbindir=/usr/sbin --sysconfdir=/etc --
datadir=/usr/share --includedir=/usr/include --libdir=/usr/lib64 --libexecdir=/
usr/lib --localstatedir=/var/lib --sharedstatedir=/var/lib --mandir=/usr/share/
man --infodir=/usr/share/info --disable-dependency-tracking --disable-silent-
rules --runstatedir=/var/run --without-included-gettext
configure: WARNING: unrecognized options: --without-included-gettext
checking for a BSD-compatible install... /usr/bin/ginstall -c
checking whether build environment is sane... yes
checking for a race-free mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
```

В рамках выполнения **configure** проверяются наличие всех сборочных зависимостей системы: наличие **make**, компилятора **gcc** и т. д.:

```
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking for x86_64-alt-linux-gcc... x86_64-alt-linux-gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether the compiler supports GNU C... yes
checking whether x86_64-alt-linux-gcc accepts -g... yes
checking for x86_64-alt-linux-gcc option to enable C11 features... none needed
checking whether x86_64-alt-linux-gcc understands -c and -o together... yes
checking whether make supports the include directive... yes (GNU style)
checking dependency style of x86_64-alt-linux-gcc... none
```

Проверяется достижимость библиотек: как по указанным в ***AC_CHECK_LIB*** проверкам, так и по заголовочным файлам:

```
checking for initscr in -lncurses... yes
checking for stdio.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for strings.h... yes
checking for sys/stat.h... yes
checking for sys/types.h... yes
```

```
checking for unistd.h... yes
checking for unistd.h... (cached) yes
checking for ncurses.h... yes
checking for error_at_line... yes
checking that generated files are newer than configure... done
```

Создаётся специальный сценарий **`config.status`**, хранящий информацию и параметры последней удачной сборки окружения:

```
configure: creating ./config.status
config.status: creating Makefile
config.status: creating config.h
config.status: executing depfiles commands
configure: WARNING: unrecognized options: --without-included-gettext
<...>
```

Отдельное внимание заслуживает директива ***`%check`***, в которой описываются результаты тестирования сценариев из **`Makefile.am`**:

```
Executing(%check): /bin/sh -e /usr/src/tmp/rpm-tmp.61556
+ umask 022
+ /bin/mkdir -p /usr/src/RPM/BUILD
+ cd /usr/src/RPM/BUILD
+ cd autoenv-pkg-1.0
+ make check
make: Entering directory '/usr/src/RPM/BUILD/autoenv-pkg-1.0'
make  check-TESTS
make[1]: Entering directory '/usr/src/RPM/BUILD/autoenv-pkg-1.0'
make[2]: Entering directory '/usr/src/RPM/BUILD/autoenv-pkg-1.0'
echo 'test "`./autoenv < /dev/null 2>&1`" = "./autoenv: Not a terminal"' >
isterm.sh
chmod +x isterm.sh
PASS: isterm.sh
============================================================================
Testsuite summary for autoenv-pkg 1.0
============================================================================
# TOTAL: 1
# PASS:  1
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
make[2]: Leaving directory '/usr/src/RPM/BUILD/autoenv-pkg-1.0'
make[1]: Leaving directory '/usr/src/RPM/BUILD/autoenv-pkg-1.0'
make: Leaving directory '/usr/src/RPM/BUILD/autoenv-pkg-1.0'
+ exit 0
<...>
```

9.3. Сборка окружения с параметрами

При сборке пакетов иногда необходимо собирать пакеты с разными параметрами конфигурационного окружения (которые описываются флагами **`configure`**). Например, пакет сопровождается обширной документацией, для пересборки которой необходим *TeXlive*, или секция ***`%check`*** выполняется несколько десятков минут. Во время отладки сборки хочется уметь отключать эти ресурсоёмкие действия (или, допустим, наоборот — включать профилирование), не меняя spec-файла. Для этого в сообществе ALT придумали макросы-переключатели ***`%def_enable`*/`%def_with`**.

Парные к ним макросы ***`%subst_enable`*/`%subst_with`*** превращаются в соответствующий параметр **`confgure`**:

```
[user@VM ~]$ rpm --eval "%def_enable trololo
%{subst_enable trololo}"

--enable-trololo
```

Значение макроса RPM выставляется в spec-файле, но может быть переопределено из командной строки при запуске, например, **`rpmbuild … --disable=keypad …`**.

Добавим возможность включения и выключения режима распознавания функциональных клавиш при сборке. Для этого в spec-файле добавим макрос-переключатель на соответствующий параметр:

@**user**: **`autoenv-pkg/autoenv-pkg.spec`**

```
%def_enable keypad

Name: autoenv-pkg
Version: 1.0
Release: alt1

Summary: Test pkg with autotool

License: GPL-3.0-or-later
Group: Development/Other

Source0: %name-%version.tar.gz
BuildRequires: libncurses-devel

%description
This is a small testing package, builded by autotools

%prep
%setup

%build
%autoreconf
%configure %{subst_enable keypad}
%make_build

%install
%makeinstall_std

%check
make check
```

```
%files
%_bindir/*

%changelog
* Mon Jul 21 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial build
```

Соберём пакет с помощью **gear-hsh**. Заметим, что поскольку по умолчанию макрос определяется, как ***enabled***, сборка без явного указания параметра привела к автоматической подстановке ключа ***--enable-keypad***:

@user

```
[user@VM autoenv-pkg]$ gear-hsh -v
<...>
+ ./configure --build=x86_64-alt-linux --host=x86_64-alt-linux --prefix=/usr --
exec-prefix=/usr --bindir=/usr/bin --sbindir=/usr/sbin --sysconfdir=/etc --
datadir=/usr/share --includedir=/usr/include --libdir=/usr/lib64 --libexecdir=/
usr/lib --localstatedir=/var/lib --sharedstatedir=/var/lib --mandir=/usr/share/
man --infodir=/usr/share/info --disable-dependency-tracking --disable-silent-
rules --runstatedir=/var/run --without-included-gettext --enable-keypad
<...>
Wrote: /usr/src/RPM/SRPMS/autoenv-pkg-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/autoenv-pkg-1.0-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/autoenv-pkg-debuginfo-1.0-alt1.x86_64.rpm
(w2.lzdio)
4.09user 3.23system 0:11.55elapsed 63%CPU (0avgtext+0avgdata 23496maxresident)k
576inputs+5800outputs (0major+381106minor)pagefaults 0swaps
[user@VM autoenv-pkg]$
```

Установим пакет в hasher и проверим распознавание функциональных клавиш. Для этого воспользуемся сценарием hypersh, с помощью которого развернём пакет с исходным кодом в окружении и установим зависимости для пакета:

@user

```
[user@VM ~]$ hypersh hasher/repo/SRPMS.hasher/autoenv-pkg-1.0-alt1.src.rpm
<...>
<13>Jul 24 20:52:04 rpmi: libncurses6-6.3.20220618-alt4 sisyphus+327286.4600.14.1
1711486705 installed
<13>Jul 24 20:52:04 rpmi: libtinfo-devel-6.3.20220618-alt4
sisyphus+327286.4600.14.1 1711486705 installed
<13>Jul 24 20:52:04 rpmi: libncurses-devel-6.3.20220618-alt4
sisyphus+327286.4600.14.1 1711486705 installed
<...>
[builder@localhost .in]$
logout
[user@VM ~]$ cp hasher/repo/x86_64/RPMS.hasher/autoenv-pkg-1.0-alt1.x86_64.rpm
hasher/chroot/.in
[user@VM ~]$ hypersh --rooter
mesg: /dev/pts/0: Read-only file system
[root@localhost .in]# rpm -i autoenv-pkg-1.0-alt1.x86_64.rpm
<13>Jul 24 20:52:58 rpm: autoenv-pkg-1.0-alt1 1753389819 installed

[root@localhost .in]# which autoenv
/usr/bin/autoenv
[root@localhost .in]#
```

autoenv

```
autoenv-pkg-1.0 (UsamG1t)


  ┌───────────────────────────────────────┐
  │ Key: 68, Name: D                      │
  │ Key: 100, Name: d                     │
  │ Key: 265, Name: KEY_F(1)              │
  │ Key: 266, Name: KEY_F(2)              │
  │ Key: 267, Name: KEY_F(3)              │
  │ Key: 268, Name: KEY_F(4)              │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  │                                       │
  └───────────────────────────────────────┘
```

Теперь соберём пакет прямо в hasher с явным указанием выключенного ***keypad***:

@builder

```
[builder@localhost ~]$ tree -A
.
├── RPM
│   ├── BUILD
│   ├── RPMS
```

```
│   │   └── noarch
│   ├── SOURCES
│   │   └── autoenv-pkg-1.0.tar.gz
│   ├── SPECS
│   │   └── autoenv-pkg.spec
│   └── SRPMS
├── debug
├── in
│   ├── SOURCE_DATE_EPOCH
│   ├── nosrpm
│   │   └── autoenv-pkg-1.0-alt1.nosrc.rpm
│   └── srpm
│       └── autoenv-pkg-1.0-alt1.src.rpm
└── tmp

13 directories, 5 files
[builder@localhost ~]$ rpmbuild -ba --disable=keypad RPM/SPECS/autoenv-pkg.spec
<...>
+ ./configure --build=x86_64-alt-linux --host=x86_64-alt-linux --prefix=/usr --
exec-prefix=/usr --bindir=/usr/bin --sbindir=/usr/sbin --sysconfdir=/etc --
datadir=/usr/share --includedir=/usr/include --libdir=/usr/lib64 --libexecdir=/
usr/lib --localstatedir=/var/lib --sharedstatedir=/var/lib --mandir=/usr/share/
man --infodir=/usr/share/info --disable-dependency-tracking --disable-silent-
rules --runstatedir=/var/run --without-included-gettext --disable-keypad
<...>
Wrote: /usr/src/RPM/SRPMS/autoenv-pkg-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/autoenv-pkg-1.0-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/autoenv-pkg-debuginfo-1.0-alt1.x86_64.rpm
(w2.lzdio)
[builder@localhost ~]$
```

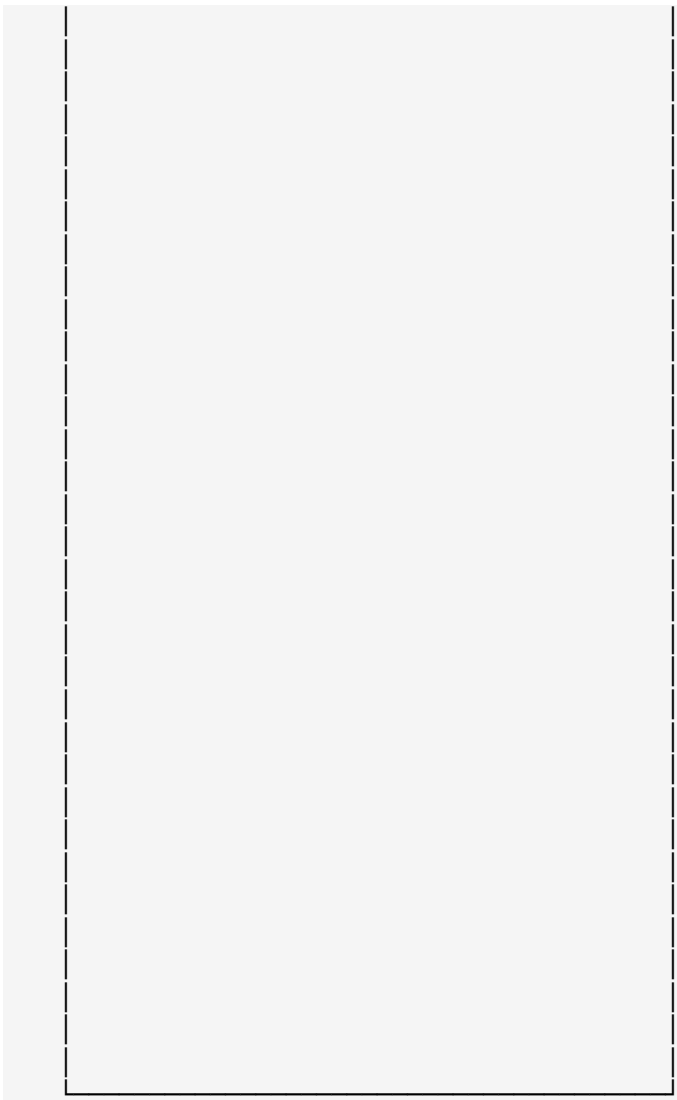
Установим эту версию пакета и убедимся, что функциональные клавиши не опознаются:

@rooter

```
[root@localhost .in]# rpm -i /usr/src/RPM/RPMS/x86_64/autoenv-pkg-1.0-
alt1.x86_64.rpm
<13>Jul 24 21:07:51 rpm: autoenv-pkg-1.0-alt1 1753391141 installed

[root@localhost .in]# which autoenv
/usr/bin/autoenv
[root@localhost .in]# ls -la /usr/bin/autoenv
-rwxr-xr-x 1 root root 14608 Jul 24 21:05 /usr/bin/autoenv
[root@localhost .in]# date
Thu Jul 24 21:08:23 UTC 2025
[root@localhost .in]#
```

autoenv

```
autoenv-pkg-1.0 (UsamG1t)


  ┌──────────────────────────────────────┐
  │ Key: 68, Name: D                     │
  │ Key: 100, Name: d                    │
  │                                      │
  │                                      │
  │                                      │
```



Поскольку функциональные клавиши больше не считываются как символы для вывода, программа автоматически завершается при получении соответствующего сигнала.

Второй параметр сборки — ***frame*** — требует задавать некоторое значение. Явно укажем его в качестве параметра **configure**, а для определения будем использовать специальную **ifdef**-конструкцию: ***configure … --enable-frame=%{?framewidth:%framewidth}%{!?framewidth:3}***. Тогда при вызове **rpmbuild** можно будет переопределить значение макроса с помощью ***rpmbuild --define 'framewidth 20' …***, а при его отсутствии будет использоваться значение по умолчанию.

Добавим обработку ***frame*** в наш пакет:

@**user**: **autoenv-pkg/autoenv-pkg.spec**

```
%def_enable keypad

Name: autoenv-pkg
Version: 1.0
Release: alt1

Summary: Test pkg with autotool

License: GPL-3.0-or-later
```

```
Group: Development/Other

Source0: %name-%version.tar.gz
BuildRequires: libncurses-devel

%description
This is a small testing package, builded by autotools

%prep
%setup

%build
%autoreconf
%configure %{subst_enable keypad} --enable-frame=%{?framewidth:%framewidth}%{!?
framewidth:3}

%make_build

%install
%makeinstall_std

%check
make check

%files
%_bindir/*

%changelog
* Mon Jul 21 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial build
```

Соберём пакет через **gear-hsh** и установим его в hasher:

@user

```
[user@VM autoenv-pkg]$ gear-hsh -v
<...>
+ ./configure --build=x86_64-alt-linux --host=x86_64-alt-linux --prefix=/usr --
exec-prefix=/usr --bindir=/usr/bin --sbindir=/usr/sbin --sysconfdir=/etc --
datadir=/usr/share --includedir=/usr/include --libdir=/usr/lib64 --libexecdir=/
usr/lib --localstatedir=/var/lib --sharedstatedir=/var/lib --mandir=/usr/share/
man --infodir=/usr/share/info --disable-dependency-tracking --disable-silent-
rules --runstatedir=/var/run --without-included-gettext --enable-keypad --enable-
frame=3
<...>
[user@VM autoenv-pkg]$
```

@user

```
[user@VM ~]$ hypersh hasher/repo/SRPMS.hasher/autoenv-pkg-1.0-alt1.src.rpm
<...>
<13>Jul 24 21:25:13 rpmi: libncurses6-6.3.20220618-alt4 sisyphus+327286.4600.14.1
1711486705 installed
<13>Jul 24 21:25:13 rpmi: libtinfo-devel-6.3.20220618-alt4
sisyphus+327286.4600.14.1 1711486705 installed
<13>Jul 24 21:25:13 rpmi: libncurses-devel-6.3.20220618-alt4
sisyphus+327286.4600.14.1 1711486705 installed
<...>
```

```
[builder@localhost .in]$
logout
[user@VM ~]$ cp hasher/repo/x86_64/RPMS.hasher/autoenv-pkg-1.0-alt1.x86_64.rpm
hasher/chroot/.in
[user@VM ~]$ hypersh --rooter
mesg: /dev/pts/0: Read-only file system
[root@localhost .in]# rpm -i autoenv-pkg-1.0-alt1.x86_64.rpm
<13>Jul 24 21:26:25 rpm: autoenv-pkg-1.0-alt1 1753392208 installed

[root@localhost .in]#
```

autoenv

```
autoenv-pkg-1.0 (UsamG1t)
```

Аналогично соберём пакет с другим размером рамки:

@builder

```
[builder@localhost ~]$ rpmbuild -ba --define 'framewidth 20' RPM/SPECS/autoenv-pkg.spec
<...>
+ ./configure --build=x86_64-alt-linux --host=x86_64-alt-linux --prefix=/usr --exec-prefix=/usr --bindir=/usr/bin --sbindir=/usr/sbin --sysconfdir=/etc --datadir=/usr/share --includedir=/usr/include --libdir=/usr/lib64 --libexecdir=/usr/lib --localstatedir=/var/lib --sharedstatedir=/var/lib --mandir=/usr/share/man --infodir=/usr/share/info --disable-dependency-tracking --disable-silent-rules --runstatedir=/var/run --without-included-gettext --enable-keypad --enable-frame=20
<...>
Wrote: /usr/src/RPM/SRPMS/autoenv-pkg-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/autoenv-pkg-1.0-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/autoenv-pkg-debuginfo-1.0-alt1.x86_64.rpm (w2.lzdio)
[builder@localhost ~]$
```

@rooter

```
[root@localhost .in]# rpm -i --replacefiles /usr/src/RPM/RPMS/x86_64/autoenv-pkg-1.0-alt1.x86_64.rpm
<13>Jul 28 06:28:55 rpm: autoenv-pkg-1.0-alt1 1753683871 installed
[root@localhost .in]#
```

autoenv

```
autoenv-pkg-1.0 (UsamG1t)
```

Глава 10. Интернационализация и локализация

При разработке ПО для широкой аудитории необходимо проводить интернационализацию и локализацию продукта. Одной из основных задач локализации является перевод текстов с поддержкой множественных форм. Локализация базовой библиотеки GLibC в ALT содержится в отдельном пакете — *glibc-locales*, все остальные программные продукты используют для хранения локализованных объектов файлы в подкаталогах **`/usr/share/locale`** с указанием кода локали, класса объектов и т. н. *домена*.

10.1. Работа с Gettext

Рассмотрим программу, последовательно считающую овец:

@**user**: **`in-place/src/sheepcounter.c`**

```
#include <stdio.h>
#include <stdlib.h>
#include <libgen.h>
#include <libintl.h>
#include <locale.h>

#define _(STRING) gettext(STRING)

int main(int argc, char *argv[])
{
       int count;

       setlocale (LC_ALL, "");
       bindtextdomain (PACKAGE, LOCALE_PATH);
       textdomain (PACKAGE);

       /* Simple text */
       puts(_("Let's count sheeps together!\n"));
       puts(_("How many sheeps do you want to count? "));

       scanf("%d", &count);

#ifndef NUMBERS
       while (count >= 10) {
```

```
                puts(_("Oh, I know only digits, not numbers, try again: "));
                scanf("%d", &count);
        }
#endif

        for(int i = 1; i <= count; i++)
                /* Plural example */
                printf(ngettext("We counted %d sheep\n", "We counted %d sheeps\n",
i), i);

        return 0;
}
```

Для работы с переводами используется специальный набор инструментов GNU gettext. С помощью специальных функций-обёрток **gettext** и **ngettext** из библиотеки **`locale.h`** обозначаются элементы, которые необходимо будет локализовать. Для элементов, в которых возможна множественная форма, указывается также параметр, от которого она будет зависеть.

В GNU gettext идентификатором сообщения является само сообщение — строка на исходном языке. Переводы объединены в т. н. *домены*, внутри которых идентификаторы уникальны, а сами сообщения принадлежат одному культурному контексту. Несколько приложений могут пользоваться одним доменом переводов; главное, чтобы исходные строки-идентификаторы были равны. Одно приложение может пользоваться несколькими доменами — например, в ситуации, когда одно и то же словосочетание необходимо в одной и той же локали перевести по-разному в зависимости от контекста. В большинстве случаев название домена совпадает с названием запускаемой программы или пакета, в состав которого она входит.

Соответствие домена и директории для поиска переводов устанавливаются функцией **bindtextdomain()**, при этом файл с переводом определяется поиском в этой директории по шаблону **`<директория>/<код_локали>/<класс_объекта>/<домен>.mo`**. Сам домен устанавливается функцией **textdomain()**.

Для генерации шаблона перевода (**pot — po template**) используется утилита **xgettext**, которой указывается название функции-обёртки, входной и выходной файлы: **`xgettext -k_ -c src/sheepcounter.c -o po/sheepcounter.pot`**:

@**user**: **`sheepcounter-pkg/po/sheepcounter.pot`**

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2025-07-24 17:22+0300\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"Language: \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=INTEGER; plural=EXPRESSION;\n"
```

```
#. Simple text
#: src/sheepcounter.c:19
msgid "Let's count sheeps together!\n"
msgstr ""

#: src/sheepcounter.c:20
msgid "How many sheeps do you want to count? "
msgstr ""

#: src/sheepcounter.c:26
msgid "Oh, I know only digits, not numbers, try again: "
msgstr ""

#. Plural example
#: src/sheepcounter.c:33
#, c-format
msgid "We counted %d sheep\n"
msgid_plural "We counted %d sheeps\n"
msgstr[0] ""
msgstr[1] ""
```

Шаблон содержит все элементы, требующие локализации. Для создания первичного (пустого) текстового файла с переводами используется утилита **msginit** с указанием локали: **`msginit -i po/sheepcounter.pot -o po/ru.po -l ru_RU.UTF-8`**. При наличии уже написанного перевода (например, при внесении каких-то изменений в исходный текст программы, требующих дополнительной локализации) шаблон сначала обновляется, а затем на его основе с помощью утилиты **msgmerge** обновляется и старый перевод: **`msgmerge -U po/ru.po po/sheepcounter.pot`**. Новые сообщения оказываются с пустым переводом, перевод исчезнувших сообщений комментируется, но не удаляется, а переводы старых, но слегка изменившихся сообщений, помечаются флагом ***`fuzzy`*** — «нечёткий».

Текстовый файл перевода помимо сообщений содержит мета-данные о конкретном переводе на заданном языке, которые включают формулу обработки множественных форм (в разных языках их разное количество и разные правила соответствия). Пустые и fuzzy-переводы самостоятельно исправляет в **`.po`**-файле автор:

@**user**: **`sheepcounter-pkg/po/ru.po`**

```
# Russian translations for PACKAGE package
# Английские переводы для пакета PACKAGE.
# Copyright (C) 2025 THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE package.
#  <user@usamg1tvm>, 2025.
#
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2025-07-24 17:22+0300\n"
"PO-Revision-Date: 2025-07-24 17:34+0300\n"
"Last-Translator:  <user@usamg1tvm>\n"
"Language-Team: Russian <gnu@d07.ru>\n"
"Language: ru\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=3; plural=(n%10==1 && n%100!=11 ? 0 : n%10>=2 && n"
```

```
"%10<=4 && (n%100<10 || n%100>=20) ? 1 : 2);\n"

#. Simple text
#: src/sheepcounter.c:19
msgid "Let's count sheeps together!\n"
msgstr "Давай считать овец вместе!\n"

#: src/sheepcounter.c:20
msgid "How many sheeps do you want to count? "
msgstr "Сколько овец ты хочешь посчитать? "

#: src/sheepcounter.c:26
msgid "Oh, I know only digits, not numbers, try again: "
msgstr "Оу, я знаю только цифры, не числа, напиши что-то другое: "

#. Plural example
#: src/sheepcounter.c:33
#, c-format
msgid "We counted %d sheep\n"
msgid_plural "We counted %d sheeps\n"
msgstr[0] "Мы посчитали %d овцу\n"
msgstr[1] "Мы посчитали %d овцы\n"
msgstr[2] "Мы посчитали %d овец\n"
```

Для компиляции перевода в файл с доменом используется утилита **msgfmt**. Предварительно необходимо создать и указать поддиректории для расположения файла перевода:

```
mkdir -p ru/LC_MESSAGES/
msgfmt po/ru.po -o ru/LC_MESSAGES/sheepcounter.mo
```

Скомпилируем программу и проверим её работу. Заметим, что макросы ***PACKAGE*** и ***LOCALE_PATH*** в самой программе не определены — мы можем задать их напрямую из командной строки. В нашем случае переводы искать надо прямо в текущем каталоге, а домен называется *sheepcounter*:

@user

```
[user@VM in-place]$ cc -DPACKAGE='"sheepcounter"' -D LOCALE_PATH='"."' src/
sheepcounter.c -o sheepcounter
[user@VM in-place]$ ./sheepcounter
Давай считать овец вместе!

Сколько овец ты хочешь посчитать?
10
Оу, я знаю только цифры, не числа, напиши что-то другое:
5
Мы посчитали 1 овцу
Мы посчитали 2 овцы
Мы посчитали 3 овцы
Мы посчитали 4 овцы
Мы посчитали 5 овец
[user@VM in-place]$ LC_MESSAGES=en_US.UTF-8 ./sheepcounter
Let's count sheeps together!

How many sheeps do you want to count?
4
We counted 1 sheep
```

```
We counted 2 sheeps
We counted 3 sheeps
We counted 4 sheeps
[user@VM in-place]$
```

10.2. Сборка пакета с поддержкой переводов

Соберём пакет, в котором будут поддерживаться переводы. При этом возникают некоторые особенности установки.

При установке пакета в систему файлы переводов должны быть установлены в специальную директорию хранения всех переводов приложений системы — **`/usr/share/locale`**. Расположение файла с доменом в ней стандартное — **`/usr/share/locale/<код_локали>/<класс_объекта>/<домен>.mo`** .

Для сборки напишем небольшой **`Makefile`**, в котором макрос ***`PACKAGE`*** зададим сразу, а вот расположение директории с локалями оставим на усмотрение автора пакета. Обычно **make** игнорирует описание переменных в **`Makefile`**, если пользователь переопределил их из командной строки. Для того чтобы дополнить описание в **`Makefile`** командной строкой, пометим его ключевым словом ***`override`*** и используем операцию «**`+=`**»:

@**user**: **`sheepcounter-pkg/Makefile`**

```
RU = ru/LC_MESSAGES
PROGRAM = sheepcounter
override CFLAGS += -Wall -O0 -g -DPACKAGE='"$(PROGRAM)"'
CC = cc

GENS = */*.mo $(PROGRAM) */*.pot
TRASH = *.o *~ */*~ o.*

all: $(RU)/$(PROGRAM).mo $(PROGRAM)

$(PROGRAM): src/$(PROGRAM).c
	$(CC) $(CFLAGS) $< -o $@

$(RU)/$(PROGRAM).mo: po/ru.po
	mkdir -p `dirname $@`
	msgfmt $< -o $@

po/$(PROGRAM).pot: src/$(PROGRAM).c
	xgettext -k_ -c $< -o $@

po/ru.po:  po/$(PROGRAM).pot
	msgmerge -U $@ $<

clean:
	 rm -f $(TRASH)

distclean:	clean
	 rm -f $(GENS)
```

Добавим ***`.gear/rules`*** и spec-файл:

@**user**: **`sheepcounter-pkg/.gear/rules`**

```
spec:   .gear/sheepcounter.spec
tar.gz: . name=@name@-@version@
```

В файл спецификации добавим определение недостающего макроса ***LOCALE_PATH*** таким образом, чтобы перевод находился после установки пакета. Поскольку в Makefile всё для конкатенации уже написано, здесь необходимо просто указать данные для конкатенации в соответствующей переменной:

@user: **sheepcounter-pkg/.gear/sheepcounter.spec**

```
Name: sheepcounter
Version: 0.0
Release: alt1

Summary: Test pkg with i18n

License: GPL-3.0-or-later
Group: Development/Other

Source0: %name-%version.tar.gz

%description
This is a small testing package, builded with i18n

%prep
%setup

%build
make CFLAGS=-DLOCALE_PATH=\'\"%_datadir/locale\"\'

%install
install -D %name %buildroot%_bindir/%name
install -D -m644 ru/LC_MESSAGES/%name.mo %buildroot%_datadir/locale/ru/
LC_MESSAGES/%name.mo


%files
%_bindir/%name
%_datadir/locale/*/*/*.mo

%changelog
* Wed Jul 30 2025 UsamG1t <usamg1t@altlinux.org> 0.0-alt1
- Initial build
```

В spec-файле производится ручная установка исполняемого файла и файла переводов, в директиве ***%files*** явно описаны итоговые файлы пакета.

Соберём пакет и попробуем запустить его:

@user

```
[user@VM sheepcounter-pkg]$ gear-hsh --lazy
<...>
[user@VM sheepcounter-pkg]$ cp ~/hasher/repo/x86_64/RPMS.hasher/sheepcounter-0.0-
alt1.x86_64.rpm ~/hasher/chroot/.in
[user@VM sheepcounter-pkg]$ hypersh --rooter
```

@rooter

```
[root@localhost .in]# rpm -i sheepcounter-0.0-alt1.x86_64.rpm
<13>Jul 30 05:59:13 rpm: sheepcounter-0.0-alt1 1753854986 installed

[root@localhost .in]# which sheepcounter
/usr/bin/sheepcounter

[root@localhost .in]# locale
LANG=C.UTF-8
LC_CTYPE="C.UTF-8"
LC_NUMERIC="C.UTF-8"
LC_TIME="C.UTF-8"
LC_COLLATE="C.UTF-8"
LC_MONETARY="C.UTF-8"
LC_MESSAGES="C.UTF-8"
LC_PAPER="C.UTF-8"
LC_NAME="C.UTF-8"
LC_ADDRESS="C.UTF-8"
LC_TELEPHONE="C.UTF-8"
LC_MEASUREMENT="C.UTF-8"
LC_IDENTIFICATION="C.UTF-8"
LC_ALL=

[root@localhost .in]# sheepcounter
Let's count sheeps together!

How many sheeps do you want to count?
10
Oh, I know only digits, not numbers, try again:
6
We counted 1 sheep
We counted 2 sheeps
We counted 3 sheeps
We counted 4 sheeps
We counted 5 sheeps
We counted 6 sheeps

[root@localhost .in]# LC_MESSAGES=ru_RU.UTF-8 sheepcounter
Давай считать овец вместе!

Сколько овец ты хочешь посчитать?
8
Мы посчитали 1 овцу
Мы посчитали 2 овцы
Мы посчитали 3 овцы
Мы посчитали 4 овцы
Мы посчитали 5 овец
Мы посчитали 6 овец
Мы посчитали 7 овец
Мы посчитали 8 овец
[root@localhost .in]#
```

10.3. Переводы с помощью autotools

Добавим autotools-поддержку переводов. Поскольку все действия над переводами — шаблонные, нет необходимости задавать собственный **po/Makefile.am**. Он и финальный **po/Makefile** целиком генерируются из файлов **po/Makevars** (с настройкой дополнительных переменных **po/Makefile**) и **po/POTFILES.in** (со списком источников сообщений, требующих перевода). Название файла с переводом выбирается автоматически — это код языка + расширение ***.po***, в нашем случае — ***ru.po***.

Файл **po/Makevars** подойдёт стандартный; в ALT он входит с состав пакета *gettext-tools*:

@user

```
[user@VM sheepcounter-pkg] cp /usr/share/gettext/po/Makevars.template po/Makevars
```

Перевод требуется только для строк в ***sheepcounter.c***:

@user: **sheepcounter-pkg/po/POTFILES.in**

```
src/sheepcounter.c
```

Классическая структура autotools предполагает отдельный **Makefile** в каждом компоненте проекта — в базовой директории, в **src/** и в **/po**; для двух из них напишем примитивные **Makefile.am**:

@user: **sheepcounter-pkg/Makefile.am**

```
SUBDIRS = src po
```

Макрос ***PACKAGE*** попадёт в файл **config.h**, который создаётся после **./configure**. Расположение директории с системной локалью тоже задаётся с помощью **configure** и доступно в **Makefile** под именем ***localedir***. В качестве макроса Си ***LOCALE_PATH*** мы его определим, как и раньше, самостоятельно — дополнив список параметров компилятора Си, которые ему подсовывает **automake**:

@user: **sheepcounter-pkg/src/Makefile.am**

```
CFLAGS = -Wall -O0 -g
AM_CFLAGS=-D'LOCALE_PATH="$(localedir)"'

bin_PROGRAMS=sheepcounter
```

Добавим **config.h** в список заголовочных файлов нашей программы:

@user: **sheepcounter-pkg/src/sheepcounter.c**

```
#include <stdio.h>
#include <stdlib.h>
#include <libgen.h>
#include <libintl.h>
#include <locale.h>
#include "config.h"
```

```
#define _(STRING) gettext(STRING)
…

…
```

Добавим сам **`configure.ac`**. В нем для локализации указываются зависимости на GNU gettext, языки переводов, а также генерат **`po/Makefile.in`** в макросе ***`AC_CONFIG_FILES`***.

В настройки программы также добавлен параметр для работы с числами, а не только с цифрами. Отслеживается он с помощью ключей ***`--{enable/disable}-numbers`***. Ключи присваивают макросу значения ***`yes`*** или ***`no`*** соответственно, в зависимости от значения с помощью макроса условия ***`AS_IF`*** производится его определение:

@**user**: **`sheepcounter-pkg/configure.ac`**

```
#                                               -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.


AC_INIT([sheepcounter], [1.0], [UsamG1t])
AM_INIT_AUTOMAKE([foreign])
AM_GNU_GETTEXT(external)
AM_GNU_GETTEXT_REQUIRE_VERSION(0.21)
AC_CONFIG_SRCDIR([src/sheepcounter.c])
AC_CONFIG_HEADERS([config.h])
ALL_LINGUAS="ru"

# Checks for programs.
AC_PROG_CC

# Checks for libraries.

# Checks for header files.
AC_CHECK_HEADERS([libintl.h locale.h stdlib.h])

# Optional clues
AC_ARG_ENABLE([numbers],
       AS_HELP_STRING([--enable-numbers],[Enable not only digits counting]),
       AS_IF([test "$enable_numbers" = "yes"],
              AC_DEFINE(NUMBERS, [], [Numbers counting acception]))
)

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.
AC_CHECK_FUNCS([realpath setlocale])
AC_FUNC_ERROR_AT_LINE

AC_CONFIG_FILES([Makefile po/Makefile.in src/Makefile])
AC_OUTPUT
```

```
.
├── configure.ac
├── Makefile.am
├── po
│   ├── Makevars
│   ├── POTFILES.in
```

```
│   └── ru.po
└── src
    ├── Makefile.am
    └── sheepcounter.c
```

Отредактируем spec-файл (теперь в нём нет необходимости доопределять какие-либо параметры):

@**user**: **sheepcounter-pkg/.gear/sheepcounter.spec**

```
%def_disable numbers

Name: sheepcounter
Version: 1.0
Release: alt1

Summary: Test pkg with i18n

License: GPL-3.0-or-later
Group: Development/Other

Source0: %name-%version.tar.gz

%description
This is a small testing package, builded with i18n

%prep
%setup

%build
%autoreconf
%configure %{subst_enable numbers}
%make_build

%install
%makeinstall_std

%files
%_bindir/%name
%_datadir/locale/*/*/*.mo

%changelog
* Wed Jul 30 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Autotools upgrade

* Wed Jul 30 2025 UsamG1t <usamg1t@altlinux.org> 0.0-alt1
- Initial build
```

Соберём пакет:

@**user**

```
[user@VM sheepcounter-pkg]$ gear-hsh
<...>
+ ./configure --build=x86_64-alt-linux --host=x86_64-alt-linux --prefix=/usr --
exec-prefix=/usr --bindir=/usr/bin --sbindir=/usr/sbin --sysconfdir=/etc --
datadir=/usr/share --includedir=/usr/include --libdir=/usr/lib64 --libexecdir=/
```

```
usr/lib --localstatedir=/var/lib --sharedstatedir=/var/lib --mandir=/usr/share/
man --infodir=/usr/share/info --disable-dependency-tracking --disable-silent-
rules --runstatedir=/var/run --without-included-gettext --disable-numbers
<...>
```

Отдельно заметим в выводе секцию по обработке переводов: производится автоматическая сборка **.mo**-файла с его последующей установкой в правильную директорию:

```
Making all in po
<...>
rm -f ru.gmo && /usr/bin/msgmerge --for-msgfmt -o ru.1po ru.po sheepcounter.pot
&& /usr/bin/msgfmt -c --statistics --verbose -o ru.gmo ru.1po && rm -f ru.1po
<...>
Making install in po
make[1]: Entering directory '/usr/src/RPM/BUILD/sheepcounter-1.0/po'
installing ru.gmo as /usr/src/tmp/sheepcounter-buildroot/usr/share/locale/ru/
LC_MESSAGES/sheepcounter.mo
<...>
Wrote: /usr/src/RPM/SRPMS/sheepcounter-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/sheepcounter-1.0-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/sheepcounter-debuginfo-1.0-alt1.x86_64.rpm
(w2.lzdio)
5.85user 4.55system 0:14.41elapsed 72%CPU (0avgtext+0avgdata 22040maxresident)k
128inputs+11632outputs (0major+541655minor)pagefaults 0swaps
[user@VM sheepcounter-pkg]$
```

Проверим работу пакета, после пересоберём его с указанием параметра и вновь проверим работу (язык в hasher из примера установлен английский по умолчанию):

@user

```
[user@VM ~]$ hypersh hasher/repo/SRPMS.hasher/sheepcounter-1.0-alt1.src.rpm
<...>
[user@VM ~]$ cp hasher/repo/x86_64/RPMS.hasher/sheepcounter-1.0-alt1.x86_64.rpm
hasher/chroot/.in/
[user@VM ~]$ hypersh --rooter
```

@rooter

```
[root@localhost .in]# rpm -i sheepcounter-1.0-alt1.x86_64.rpm
<13>Jul 30 10:03:41 rpm: sheepcounter-1.0-alt1 1753869570 installed

[root@localhost .in]# ls /usr/share/locale/ru/LC_MESSAGES/sheepcounter.mo
/usr/share/locale/ru/LC_MESSAGES/sheepcounter.mo
[root@localhost .in]# sheepcounter
Let's count sheeps together!

How many sheeps do you want to count?
30
Oh, I know only digits, not numbers, try again:
3
We counted 1 sheep
We counted 2 sheeps
We counted 3 sheeps

[root@localhost .in]# LC_MESSAGES=ru_RU.UTF-8 sheepcounter
Давай считать овец вместе!
```

```
Сколько овец ты хочешь посчитать?
5
Мы посчитали 1 овцу
Мы посчитали 2 овцы
Мы посчитали 3 овцы
Мы посчитали 4 овцы
Мы посчитали 5 овец
[root@localhost .in]#
```

@builder

```
[builder@localhost ~]$ tree -A
.
├── RPM
│   ├── BUILD
│   ├── RPMS
│   │   └── noarch
│   ├── SOURCES
│   │   └── sheepcounter-pkg-1.0.tar.gz
│   ├── SPECS
│   │   └── sheepcounter-pkg.spec
│   └── SRPMS
├── debug
├── in
│   ├── SOURCE_DATE_EPOCH
│   ├── nosrpm
│   │   └── sheepcounter-pkg-1.0-alt1.nosrc.rpm
│   └── srpm
│       └── sheepcounter-pkg-1.0-alt1.src.rpm
└── tmp

13 directories, 5 files
[builder@localhost ~]$ rpmbuild -ba -enable=numbers RPM/SPECS/sheepcounter.spec
<...>
+ ./configure --build=x86_64-alt-linux --host=x86_64-alt-linux --prefix=/usr --
exec-prefix=/usr --bindir=/usr/bin --sbindir=/usr/sbin --sysconfdir=/etc --
datadir=/usr/share --includedir=/usr/include --libdir=/usr/lib64 --libexecdir=/
usr/lib --localstatedir=/var/lib --sharedstatedir=/var/lib --mandir=/usr/share/
man --infodir=/usr/share/info --disable-dependency-tracking --disable-silent-
rules --runstatedir=/var/run --without-included-gettext --enable-numbers
<...>
Wrote: /usr/src/RPM/SRPMS/sheepcounter-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/sheepcounter-1.0-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/sheepcounter-debuginfo-1.0-alt1.x86_64.rpm
(w2.lzdio)
[builder@localhost ~]$
```

@rooter

```
[root@localhost .in]# rpm -i --replacefiles /usr/src/RPM/RPMS/x86_64/
sheepcounter-1.0-alt1.x86_64.rpm
<13>Jul 30 10:10:19 rpm: sheepcounter-1.0-alt1 1753870086 installed
[root@localhost .in]#
[root@localhost .in]# sheepcounter
Let's count sheeps together!

How many sheeps do you want to count?
10
```

```
We counted 1 sheep
We counted 2 sheeps
We counted 3 sheeps
We counted 4 sheeps
We counted 5 sheeps
We counted 6 sheeps
We counted 7 sheeps
We counted 8 sheeps
We counted 9 sheeps
We counted 10 sheeps
[root@localhost .in]# LC_MESSAGES=ru_RU.UTF-8 sheepcounter
Давай считать овец вместе!

Сколько овец ты хочешь посчитать?
21
Мы посчитали 1 овцу
Мы посчитали 2 овцы
Мы посчитали 3 овцы
Мы посчитали 4 овцы
Мы посчитали 5 овец
Мы посчитали 6 овец
Мы посчитали 7 овец
Мы посчитали 8 овец
Мы посчитали 9 овец
Мы посчитали 10 овец
Мы посчитали 11 овец
Мы посчитали 12 овец
Мы посчитали 13 овец
Мы посчитали 14 овец
Мы посчитали 15 овец
Мы посчитали 16 овец
Мы посчитали 17 овец
Мы посчитали 18 овец
Мы посчитали 19 овец
Мы посчитали 20 овец
Мы посчитали 21 овцу
[root@localhost .in]#
```

Глава 11. Модификация сторонних исходников

При работе со сторонними исходниками нередко необходимо вносить изменения в исходные тексты продукта для, например, его локализации, адаптации к системе или исправления зависимостей. В случае, когда разработчики не принимают изменения в основной код (или передача этих изменений должна быть ограничена некоторым кругом лиц и потому не может быть передана для добавления в основной код), можно локально изменить исходники, однако с обновлением основной версии необходимо будет переносить все изменения.

Для упрощения и отчасти автоматизации переноса изменений можно использовать **патчи** — сообщения специального формата, в которых описывается набор изменений объектов для их преобразования.

При создании первой версии изменений:

- исходники непосредственно исправляются под нужды автора / сообщества;
- исправления оформляются в виде патчей;
- патч-сет сохраняется;
- исправленная версия используется или выкладывается в открытый доступ в зависимости от нужд.

При *обновлении* исходников нужно:

- применить патчи к новой версии (при этом часть может примениться сразу, а часть — отвалиться из-за изменений);
- адаптировать оставшиеся патчи для их применимости;
- сохранить новый патч-сет;
- снова пользоваться надстройками.

С появлением распределённых систем контроля версий, таких как GIT, роль патч-сетов стали играть отдельные ветки разработки, и коммиты в них; однако ***diff / patch*** (или ***git diff / patch***) — всё ещё более гибкий инструмент, а явное хранение различий в **.patch**-файлах более наглядно.

11.1. Отслеживание изменений

Рассмотрим, как производится отслеживание изменений в файлах и описание патчей для их преобразования. С помощью потокового редактора **sed** сделаем три последовательных изменения файла. С помощью флага ***-i*** зададим постфикс для имени файла, в котором будет сохранена версия файла до изменения:

@user

```
[user@VM ~]$ mkdir calend-patches
[user@VM ~]$ cd calend-patches/

[user@VM calend-patches]$ cal -y > calend
[user@VM calend-patches]$ cat calend
                                2025

      Январь                Февраль                 Март
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
      1  2  3  4  5                   1  2                   1  2
6  7  8  9 10 11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9
13 14 15 16 17 18 19   10 11 12 13 14 15 16   10 11 12 13 14 15 16
20 21 22 23 24 25 26   17 18 19 20 21 22 23   17 18 19 20 21 22 23
27 28 29 30 31         24 25 26 27 28         24 25 26 27 28 29 30
                                              31
      Апрель                  Май                   Июнь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
   1  2  3  4  5  6              1  2  3  4                      1
7  8  9 10 11 12 13    5  6  7  8  9 10 11    2  3  4  5  6  7  8
14 15 16 17 18 19 20   12 13 14 15 16 17 18    9 10 11 12 13 14 15
21 22 23 24 25 26 27   19 20 21 22 23 24 25   16 17 18 19 20 21 22
```

```
28 29 30                26 27 28 29 30 31      23 24 25 26 27 28 29
                                              30
       Июль                    Август                  Сентябрь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
   1  2  3  4  5  6                  1  2  3    1  2  3  4  5  6  7
7  8  9 10 11 12 13    4  5  6  7  8  9 10    8  9 10 11 12 13 14
14 15 16 17 18 19 20   11 12 13 14 15 16 17   15 16 17 18 19 20 21
21 22 23 24 25 26 27   18 19 20 21 22 23 24   22 23 24 25 26 27 28
28 29 30 31            25 26 27 28 29 30 31   29 30

      Октябрь                 Ноябрь                  Декабрь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
      1  2  3  4  5                  1  2    1  2  3  4  5  6  7
6  7  8  9 10 11 12    3  4  5  6  7  8  9    8  9 10 11 12 13 14
13 14 15 16 17 18 19   10 11 12 13 14 15 16   15 16 17 18 19 20 21
20 21 22 23 24 25 26   17 18 19 20 21 22 23   22 23 24 25 26 27 28
27 28 29 30 31         24 25 26 27 28 29 30   29 30 31

[user@VM calend-patches]$ sed -i.old1 's/a/@/g' calend
[user@VM calend-patches]$ cat calend
                             2025

      Янв@рь                  Февр@ль                  М@рт
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
      1  2  3  4  5                  1  2                  1  2
6  7  8  9 10 11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9
13 14 15 16 17 18 19   10 11 12 13 14 15 16   10 11 12 13 14 15 16
20 21 22 23 24 25 26   17 18 19 20 21 22 23   17 18 19 20 21 22 23
27 28 29 30 31         24 25 26 27 28          24 25 26 27 28 29 30
                                              31
      Апрель                    М@й                   Июнь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
   1  2  3  4  5  6               1  2  3  4                     1
7  8  9 10 11 12 13    5  6  7  8  9 10 11    2  3  4  5  6  7  8
14 15 16 17 18 19 20   12 13 14 15 16 17 18    9 10 11 12 13 14 15
21 22 23 24 25 26 27   19 20 21 22 23 24 25   16 17 18 19 20 21 22
28 29 30               26 27 28 29 30 31      23 24 25 26 27 28 29
                                              30
       Июль                    Август                  Сентябрь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
   1  2  3  4  5  6                  1  2  3    1  2  3  4  5  6  7
7  8  9 10 11 12 13    4  5  6  7  8  9 10    8  9 10 11 12 13 14
14 15 16 17 18 19 20   11 12 13 14 15 16 17   15 16 17 18 19 20 21
21 22 23 24 25 26 27   18 19 20 21 22 23 24   22 23 24 25 26 27 28
28 29 30 31            25 26 27 28 29 30 31   29 30

      Октябрь                 Ноябрь                  Дек@брь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
      1  2  3  4  5                  1  2    1  2  3  4  5  6  7
6  7  8  9 10 11 12    3  4  5  6  7  8  9    8  9 10 11 12 13 14
13 14 15 16 17 18 19   10 11 12 13 14 15 16   15 16 17 18 19 20 21
20 21 22 23 24 25 26   17 18 19 20 21 22 23   22 23 24 25 26 27 28
27 28 29 30 31         24 25 26 27 28 29 30   29 30 31

[user@VM calend-patches]$ sed -i.old2 's/0/ /g' calend
[user@VM calend-patches]$ cat calend
                              2 25

      Янв@рь                  Февр@ль                  М@рт
```

```
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
      1  2  3  4  5                  1  2                  1  2
6  7  8  9 1  11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9
13 14 15 16 17 18 19   1  11 12 13 14 15 16   1  11 12 13 14 15 16
2  21 22 23 24 25 26   17 18 19 2  21 22 23   17 18 19 2  21 22 23
27 28 29 3  31         24 25 26 27 28         24 25 26 27 28 29 3
                                             31
      Апрель                  М@й                    Июнь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
   1  2  3  4  5  6            1  2  3  4                     1
7  8  9 1  11 12 13    5  6  7  8  9 1  11    2  3  4  5  6  7  8
14 15 16 17 18 19 2    12 13 14 15 16 17 18    9 1  11 12 13 14 15
21 22 23 24 25 26 27   19 2  21 22 23 24 25   16 17 18 19 2  21 22
28 29 3                26 27 28 29 3  31      23 24 25 26 27 28 29
                                             3
       Июль                   Август                Сентябрь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
   1  2  3  4  5  6               1  2  3    1  2  3  4  5  6  7
7  8  9 1  11 12 13    4  5  6  7  8  9 1    8  9 1  11 12 13 14
14 15 16 17 18 19 2    11 12 13 14 15 16 17   15 16 17 18 19 2  21
21 22 23 24 25 26 27   18 19 2  21 22 23 24   22 23 24 25 26 27 28
28 29 3  31            25 26 27 28 29 3  31   29 3

      Октябрь                 Ноябрь                 Дек@брь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
      1  2  3  4  5                  1  2    1  2  3  4  5  6  7
6  7  8  9 1  11 12    3  4  5  6  7  8  9    8  9 1  11 12 13 14
13 14 15 16 17 18 19   1  11 12 13 14 15 16   15 16 17 18 19 2  21
2  21 22 23 24 25 26   17 18 19 2  21 22 23   22 23 24 25 26 27 28
27 28 29 3  31         24 25 26 27 28 29 3    29 3  31

[user@VM calend-patches]$ sed -i.old3 's/@/ю/g' calend
[user@VM calend-patches]$ cat calend
                            2 25

      Янвюрь                  Феврюль                 Мюрт
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
      1  2  3  4  5                  1  2                  1  2
6  7  8  9 1  11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9
13 14 15 16 17 18 19   1  11 12 13 14 15 16   1  11 12 13 14 15 16
2  21 22 23 24 25 26   17 18 19 2  21 22 23   17 18 19 2  21 22 23
27 28 29 3  31         24 25 26 27 28         24 25 26 27 28 29 3
                                             31
      Апрель                  Мюй                    Июнь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
   1  2  3  4  5  6            1  2  3  4                     1
7  8  9 1  11 12 13    5  6  7  8  9 1  11    2  3  4  5  6  7  8
14 15 16 17 18 19 2    12 13 14 15 16 17 18    9 1  11 12 13 14 15
21 22 23 24 25 26 27   19 2  21 22 23 24 25   16 17 18 19 2  21 22
28 29 3                26 27 28 29 3  31      23 24 25 26 27 28 29
                                             3
       Июль                   Август                Сентябрь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
   1  2  3  4  5  6               1  2  3    1  2  3  4  5  6  7
7  8  9 1  11 12 13    4  5  6  7  8  9 1    8  9 1  11 12 13 14
14 15 16 17 18 19 2    11 12 13 14 15 16 17   15 16 17 18 19 2  21
21 22 23 24 25 26 27   18 19 2  21 22 23 24   22 23 24 25 26 27 28
28 29 3  31            25 26 27 28 29 3  31   29 3
```

```
      Октябрь                 Ноябрь                  Декюбрь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
      1  2  3  4  5                  1  2    1  2  3  4  5  6  7
6  7  8  9 1  11 12    3  4  5  6  7  8  9    8  9 1  11 12 13 14
13 14 15 16 17 18 19   1  11 12 13 14 15 16   15 16 17 18 19 2  21
2  21 22 23 24 25 26   17 18 19 2  21 22 23   22 23 24 25 26 27 28
27 28 29 3  31         24 25 26 27 28 29 3    29 3  31

[user@VM calend-patches]$ ls
calend  calend.old1  calend.old2  calend.old3
[user@VM calend-patches]$
```

Глазами найти изменения довольно трудно, поэтому воспользуемся утилитой diff. При этом добавим флаг ***-u***, который оформляет найденные различия в виде единого контекстного блока с указанными изменениями (т. н. «unified diff»). Именно такой формат в подавляющем большинстве случаев применяется для описания патчей:

@user

```
[user@VM calend-patches]$ diff -u calend.old1 calend.old2 > p0.patch
[user@VM calend-patches]$ diff -u calend.old2 calend.old3 > p1.patch
[user@VM calend-patches]$ diff -u calend.old3 calend > p2.patch

[user@VM calend-patches]$ cat p0.patch
--- calend.old1 2025-07-31 15:22:20.207687635 +0300
+++ calend.old2 2025-07-31 15:33:38.576477837 +0300
@@ -1,6 +1,6 @@
                               2025

-       Январь                  Февраль                 Март
+       Янв@рь                  Февр@ль                 М@рт
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
       1  2  3  4  5                  1  2                    1  2
 6  7  8  9 10 11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9
@@ -8,7 +8,7 @@
20 21 22 23 24 25 26   17 18 19 20 21 22 23   17 18 19 20 21 22 23
27 28 29 30 31         24 25 26 27 28         24 25 26 27 28 29 30
                                              31
-       Апрель                  Май                     Июнь
+       Апрель                  М@й                     Июнь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
    1  2  3  4  5  6            1  2  3  4                      1
 7  8  9 10 11 12 13    5  6  7  8  9 10 11    2  3  4  5  6  7  8
@@ -24,7 +24,7 @@
21 22 23 24 25 26 27   18 19 20 21 22 23 24   22 23 24 25 26 27 28
28 29 30 31            25 26 27 28 29 30 31   29 30

-       Октябрь                 Ноябрь                  Декабрь
+       Октябрь                 Ноябрь                  Дек@брь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
       1  2  3  4  5                  1  2    1  2  3  4  5  6  7
 6  7  8  9 10 11 12    3  4  5  6  7  8  9    8  9 10 11 12 13 14
[user@VM calend-patches]$
```

С помощью утилиты patch можно применять изменения к файлам для их преобразования:

@user

```
[user@VM calend-patches]$ cp calend.old1 work-copy
[user@VM calend-patches]$ patch work-copy < p0.patch
patching file work-copy
[user@VM calend-patches]$ diff -u work-copy calend.old2
[user@VM calend-patches]$
```

Так как патчи накладываются на обновляющуюся версию исходников, возможна ситуация, когда контекст наложения патча изменяется. В зависимости от того, как сильно обновление исходников изменило контекст, рассчитывается некоторый уровень применимости патча, с какого-то уровня патч не применяется:

@user

```
[user@VM calend-patches]$ cp calend.old2 work-copy

[user@VM calend-patches]$ patch --verbose work-copy < p2.patch
Hmm...  Looks like a unified diff to me...
The text leading up to this was:
--------------------------
|--- calend.old3        2025-07-31 15:34:51.901347072 +0300
|+++ calend     2025-07-31 15:35:51.631240548 +0300
--------------------------
patching file work-copy
Using Plan A...
Hunk #1 succeeded at 1 with fuzz 2.
Hunk #2 succeeded at 8 with fuzz 2.
Hunk #3 succeeded at 24 with fuzz 2.
done
[user@VM calend-patches]$ cat work-copy
                              2025

      Январь                 Феврюль                  Мюрт
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
      1  2  3  4  5                   1  2                   1  2
6  7  8  9 10 11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9
13 14 15 16 17 18 19   10 11 12 13 14 15 16   10 11 12 13 14 15 16
20 21 22 23 24 25 26   17 18 19 20 21 22 23   17 18 19 20 21 22 23
27 28 29 30 31         24 25 26 27 28         24 25 26 27 28 29 30
                                              31
      Апрель                   Мюй                  Июнь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
   1  2  3  4  5  6             1  2  3  4                      1
7  8  9 10 11 12 13    5  6  7  8  9 10 11    2  3  4  5  6  7  8
14 15 16 17 18 19 20   12 13 14 15 16 17 18    9 10 11 12 13 14 15
21 22 23 24 25 26 27   19 20 21 22 23 24 25   16 17 18 19 20 21 22
28 29 30               26 27 28 29 30 31      23 24 25 26 27 28 29
                                              30
       Июль                   Август                Сентябрь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
   1  2  3  4  5  6                1  2  3    1  2  3  4  5  6  7
7  8  9 10 11 12 13    4  5  6  7  8  9 10    8  9 10 11 12 13 14
14 15 16 17 18 19 20   11 12 13 14 15 16 17   15 16 17 18 19 20 21
21 22 23 24 25 26 27   18 19 20 21 22 23 24   22 23 24 25 26 27 28
28 29 30 31            25 26 27 28 29 30 31   29 30

      Октябрь                Ноябрь                 Декюбрь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
      1  2  3  4  5                   1  2    1  2  3  4  5  6  7
```

```
6  7  8  9 10 11 12    3  4  5  6  7  8  9    8  9 10 11 12 13 14
13 14 15 16 17 18 19   10 11 12 13 14 15 16   15 16 17 18 19 20 21
20 21 22 23 24 25 26   17 18 19 20 21 22 23   22 23 24 25 26 27 28
27 28 29 30 31         24 25 26 27 28 29 30   29 30 31

[user@VM calend-patches]$
```

@user

```
[user@VM calend-patches]$ cp calend.old2 work-copy
[user@VM calend-patches]$ sed -i "s/0/Q/g" work-copy

[user@VM calend-patches]$ patch --verbose work-copy < p1.patch
Hmm...  Looks like a unified diff to me...
The text leading up to this was:
--------------------------
|--- calend.old2        2025-07-31 15:33:38.576477837 +0300
|+++ calend.old3        2025-07-31 15:34:51.901347072 +0300
--------------------------
patching file work-copy
Using Plan A...
Hunk #1 FAILED at 1.
1 out of 1 hunk FAILED -- saving rejects to file work-copy.rej
done
[user@VM calend-patches]$
```

В случае неприменимости каких-то изменений создаётся **`.rej`**-файл для описания неприменённых изменений; эти изменения необходимо вносить руками, после чего имеет смысл обновить исходный патч:

@user

```
[user@VM calend-patches]$ ls
calend  calend.old1  calend.old2  calend.old3  p0.patch  p1.patch  p2.patch
work-copy  work-copy.orig  work-copy.rej
[user@VM calend-patches]$ cat work-copy.rej
--- calend.old2 2025-07-31 15:33:38.576477837 +0300
+++ calend.old3 2025-07-31 15:34:51.901347072 +0300
@@ -1,34 +1,34 @@
-                               2025
+                               2 25

       Янв@рь                  Февр@ль                  М@рт
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
       1  2  3  4  5                   1  2                   1  2
- 6  7  8  9 10 11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9
-13 14 15 16 17 18 19   10 11 12 13 14 15 16   10 11 12 13 14 15 16
-20 21 22 23 24 25 26   17 18 19 20 21 22 23   17 18 19 20 21 22 23
-27 28 29 30 31         24 25 26 27 28         24 25 26 27 28 29 30
+ 6  7  8  9 1  11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9
+13 14 15 16 17 18 19   1  11 12 13 14 15 16   1  11 12 13 14 15 16
+2  21 22 23 24 25 26   17 18 19 2  21 22 23   17 18 19 2  21 22 23
+27 28 29 3  31         24 25 26 27 28         24 25 26 27 28 29 3
                                              31
       Апрель                   М@й                    Июнь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
    1  2  3  4  5  6             1  2  3  4                      1
- 7  8  9 10 11 12 13    5  6  7  8  9 10 11    2  3  4  5  6  7  8
```

```
-14 15 16 17 18 19 20   12 13 14 15 16 17 18    9 10 11 12 13 14 15
-21 22 23 24 25 26 27   19 20 21 22 23 24 25   16 17 18 19 20 21 22
-28 29 30               26 27 28 29 30 31      23 24 25 26 27 28 29
-                                              30
+ 7  8  9 1  11 12 13    5  6  7  8  9 1  11    2  3  4  5  6  7  8
+14 15 16 17 18 19 2    12 13 14 15 16 17 18    9 1  11 12 13 14 15
+21 22 23 24 25 26 27   19 2  21 22 23 24 25   16 17 18 19 2  21 22
+28 29 3                26 27 28 29 3  31      23 24 25 26 27 28 29
+                                              3
        Июль                   Август                 Сентябрь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
    1  2  3  4  5  6                  1  2  3    1  2  3  4  5  6  7
- 7  8  9 10 11 12 13    4  5  6  7  8  9 10    8  9 10 11 12 13 14
-14 15 16 17 18 19 20   11 12 13 14 15 16 17   15 16 17 18 19 20 21
-21 22 23 24 25 26 27   18 19 20 21 22 23 24   22 23 24 25 26 27 28
-28 29 30 31            25 26 27 28 29 30 31   29 30
+ 7  8  9 1  11 12 13    4  5  6  7  8  9 1     8  9 1  11 12 13 14
+14 15 16 17 18 19 2    11 12 13 14 15 16 17   15 16 17 18 19 2  21
+21 22 23 24 25 26 27   18 19 2  21 22 23 24   22 23 24 25 26 27 28
+28 29 3  31            25 26 27 28 29 3  31   29 3

       Октябрь                Ноябрь                 Дек@брь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
       1  2  3  4  5                     1  2    1  2  3  4  5  6  7
- 6  7  8  9 10 11 12    3  4  5  6  7  8  9    8  9 10 11 12 13 14
-13 14 15 16 17 18 19   10 11 12 13 14 15 16   15 16 17 18 19 20 21
-20 21 22 23 24 25 26   17 18 19 20 21 22 23   22 23 24 25 26 27 28
-27 28 29 30 31         24 25 26 27 28 29 30   29 30 31
+ 6  7  8  9 1  11 12    3  4  5  6  7  8  9    8  9 1  11 12 13 14
+13 14 15 16 17 18 19   1  11 12 13 14 15 16   15 16 17 18 19 2  21
+2  21 22 23 24 25 26   17 18 19 2  21 22 23   22 23 24 25 26 27 28
+27 28 29 3  31         24 25 26 27 28 29 3    29 3  31

[user@VM calend-patches]$ diff work-copy.rej p1.patch
[user@VM calend-patches]$
```

11.2. Patchutils

Для удобной работы с патчами существует отдельный набор утилит patchutils.

11.2.1. Combinediff

Утилита **combinediff** объединяет патчи в единый блок, при этом по возможности объединяя последовательные действия.

Рассмотрим работу этой утилиты на примере объединения последовательных патчей:

@user

```
[user@VM calend-patches]$ cal
    Август 2025
Пн Вт Ср Чт Пт Сб Вс
            1  2  3
4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

```
[user@VM calend-patches]$ cal > base
[user@VM calend-patches]$ sed -i.start "s/ 2/%II/g" base
[user@VM calend-patches]$ sed -i.middle "s/II/2/g" base
[user@VM calend-patches]$ cat base.start base.middle base
    Август 2025
Пн Вт Ср Чт Пт Сб Вс
            1  2  3
4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

    Август%II025
Пн Вт Ср Чт Пт Сб Вс
            1 %II  3
4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19%II0%II1%II2%II3%II4
25%II6%II7%II8%II9 30 31

    Август%2025
Пн Вт Ср Чт Пт Сб Вс
            1 %2  3
4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19%20%21%22%23%24
25%26%27%28%29 30 31

[user@VM calend-patches]$ diff -u base.start base.middle > 1st.patch
[user@VM calend-patches]$ diff -u base.middle base > 2nd.patch
```

В описании патчей кроме самих изменений также фиксируются имена сравниваемых файлов. Это необходимо, поскольку **patch** умеет применять патчи к итоговому файлу, указанному в его ***output-namefile***. В случае **combinediff** указание разных имён будет определяться как одновременное применение изменений к разным файлам, а не последовательное к одному. Поэтому поменяем ***input-filename*** и ***output-filename*** на одинаковые и применим утилиту:

@user

```
[user@VM calend-patches]$ vim 1st.patch
[user@VM calend-patches]$ vim 2nd.patch
[user@VM calend-patches]$ cat 1st.patch
--- base        2025-08-02 14:33:55.057478757 +0300
+++ base        2025-08-02 14:35:16.045334090 +0300
@@ -1,8 +1,8 @@
-     Август 2025
+     Август%II025
Пн Вт Ср Чт Пт Сб Вс
-             1  2  3
+             1 %II  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
-18 19 20 21 22 23 24
-25 26 27 28 29 30 31
+18 19%II0%II1%II2%II3%II4
+25%II6%II7%II8%II9 30 31
```

```
[user@VM calend-patches]$ cat 2nd.patch
--- base        2025-08-02 14:35:16.045334090 +0300
+++ base        2025-08-02 14:35:37.973294915 +0300
@@ -1,8 +1,8 @@
-     Август%II025
+     Август%2025
Пн Вт Ср Чт Пт Сб Вс
-             1 %II  3
+             1 %2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
-18 19%II0%II1%II2%II3%II4
-25%II6%II7%II8%II9 30 31
+18 19%20%21%22%23%24
+25%26%27%28%29 30 31

[user@VM calend-patches]$ combinediff 1st.patch 2nd.patch > all.patch
[user@VM calend-patches]$ cat all.patch
diff -u base base
--- base        2025-08-02 14:35:16.045334090 +0300
+++ base        2025-08-02 14:35:37.973294915 +0300
@@ -1,8 +1,8 @@
-     Август 2025
+     Август%2025
Пн Вт Ср Чт Пт Сб Вс
-             1  2  3
+             1 %2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
-18 19 20 21 22 23 24
-25 26 27 28 29 30 31
+18 19%20%21%22%23%24
+25%26%27%28%29 30 31

[user@VM calend-patches]$
```

Как видно из примера, последовательное изменение файла «склеилось» в единое изменение.

11.2.2. Interdiff

Утилита **interdiff** показывает операции, которые надо провести над файлом, преобразованным только первым патчем, чтобы получить файл, преобразованный только вторым патчем (Преобразование ***base + patch1 -> base + patch2***):

@user

```
[user@VM calend-patches]$ cal > base
[user@VM calend-patches]$ sed "s/3/$/g" < base > upd1
[user@VM calend-patches]$ cat upd1
    Август 2025
Пн Вт Ср Чт Пт Сб Вс
            1  2  $
4  5  6  7  8  9 10
11 12 1$ 14 15 16 17
18 19 20 21 22 2$ 24
25 26 27 28 29 $0 $1

[user@VM calend-patches]$ sed -E "s/(П|т)/\!/g" < base > upd2
```

```
[user@VM calend-patches]$ cat upd2
    Авгус! 2025
!н В! Ср Ч! !! Сб Вс
            1  2  3
4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

[user@VM calend-patches]$ diff -u base upd1 > inter1.patch
[user@VM calend-patches]$ diff -u base upd2 > inter2.patch
[user@VM calend-patches]$ interdiff inter1.patch inter2.patch
diff -u upd1 upd2
--- upd1        2025-08-02 16:30:18.451334391 +0300
+++ upd2        2025-08-02 16:32:41.818068884 +0300
@@ -1,8 +1,8 @@
-     Август 2025
-Пн Вт Ср Чт Пт Сб Вс
-             1  2  $
+     Авгус! 2025
+!н В! Ср Ч! !! Сб Вс
+             1  2  3
 4  5  6  7  8  9 10
-11 12 1$ 14 15 16 17
-18 19 20 21 22 2$ 24
-25 26 27 28 29 $0 $1
+11 12 13 14 15 16 17
+18 19 20 21 22 23 24
+25 26 27 28 29 30 31

[user@VM calend-patches]$
```

11.2.3. Rediff

Утилита **rediff** исправляет ошибки, которые допускаются при ручном исправлении патча. При этом редактирование всё также просто основывается на контексте самого патча и никак не зависит от исходников:

@user

```
[user@VM calend-patches]$ cat all.patch
diff -u base base
--- base        2025-08-02 14:35:16.045334090 +0300
+++ base        2025-08-02 14:35:37.973294915 +0300
@@ -1,8 +1,8 @@
-     Август 2025
+     Август%2025
Пн Вт Ср Чт Пт Сб Вс
-             1  2  3
+             1 %2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
-18 19 20 21 22 23 24
-25 26 27 28 29 30 31
+18 19%20%21%22%23%24
+25%26%27%28%29 30 31

[user@VM calend-patches]$ vim all.patch
```

```
[user@VM calend-patches]$ cat all.patch
diff -u base base
--- base        2025-08-02 14:35:16.045334090 +0300
+++ base        2025-08-02 14:35:37.973294915 +0300
@@ -1,8 +1,8 @@
     Август 2025
Пн Вт Ср Чт Пт Сб Вс
-             1  2  3
+             1 %2  3
 4  5  6  7  8  9 10
+~~Middle-line text~~
11 12 13 14 15 16 17
-18 19 20 21 22 23 24
-25 26 27 28 29 30 31
+18 19%20%21%22%23%24
+25%26%27%28%29 30 31

[user@VM calend-patches]$ rediff all.patch
diff -u base base
--- base        2025-08-02 14:35:16.045334090 +0300
+++ base        2025-08-02 14:35:37.973294915 +0300
@@ -1,8 +1,9 @@
     Август 2025
Пн Вт Ср Чт Пт Сб Вс
-             1  2  3
+             1 %2  3
 4  5  6  7  8  9 10
+~~Middle-line text~~
11 12 13 14 15 16 17
-18 19 20 21 22 23 24
-25 26 27 28 29 30 31
+18 19%20%21%22%23%24
+25%26%27%28%29 30 31

[user@VM calend-patches]$
```

11.3. Патчи при сборке пакетов

При разработке с помощью Gear существует правило не изменять исходники в части git-репозитория, соответствующей авторским исходным текстам. Изменения разработчиков, сопровождающих пакет, надо держать отдельно; один из вариантов — в виде патчей. Этот вариант хорош тем, что явно поддерживается в структуре файла спецификации.

Соберём примитивный пакет, который затем преобразуем с помощью патча:

@**user**: **`hello-upgrade-pkg/prog.c`**

```
#include <stdio.h>

char str[] = "Hello, Packager!";

int main(void) {
       printf("%s\n", str);
       return 0;
}
```

@**user**: **`hello-upgrade-pkg/Makefile`**

```
TRASH = *.o *~ .gear/*~
GENS = hello-upgrade
CC = cc
CFLAGS = -O0 -g -Wall

hello-upgrade:  prog.o
	$(CC) $(CFLAGS) $< -o $@

clean:
	rm -f $(TRASH)

distclean:	clean
	rm -f $(GENS)
```

@**user**: **hello-upgrade-pkg/.gear/rules**

```
spec:   .gear/hello-upgrade.spec
tar.gz: . name=@name@-@version@
```

@**user**: **hello-upgrade-pkg/.gear/hello-upgrade.spec**

```
Name: hello-upgrade
Version: 1.0
Release: alt1

Summary: Test pkg with patch

License: GPL-3.0-or-later
Group: Development/Other

Source0: %name-%version.tar.gz

%description
This is a small testing package, builded with patch

%prep
%setup

%build
%make_build

%install
install -D %name %buildroot%_bindir/%name

%files
%_bindir/%name


%changelog
* Sat Aug 02 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial Build
```

Программа пакета просто печатает сообщение на экран:

@**user**

```
[user@VM hello-upgrade-pkg]$ gear-hsh --lazy
<...>
[user@VM hello-upgrade-pkg]$ cp ~/hasher/repo/x86_64/RPMS.hasher/hello-
upgrade-1.0-alt1.x86_64.rpm ~/hasher/chroot/.in/
[user@VM hello-upgrade-pkg]$ hsh-shell --rooter
```

@**rooter**

```
[root@localhost .in]# rpm -i hello-upgrade-1.0-alt1.x86_64.rpm
<13>Aug  2 15:58:59 rpm: hello-upgrade-1.0-alt1 1754150276 installed

[root@localhost .in]# hello-upgrade
Hello, Packager!
[root@localhost .in]#
```

Теперь добавим патч с изменениями. Для его создания воспользуемся **git diff** — встроенной в GIT утилитой определения изменений между коммитами. Её формат совместим с **unified diff** («лишние» строки с полями, относящимися к коммиту, утилита **patch** пропустит). Для начала добавим все изменения. Добавим новую функцию, которая будет возвращать сообщение для вывода. Отредактируем исходный код программы и **`Makefile`** для сборки:

@**user**: **`hello-upgrade-pkg/fun.c`**

```
#include <stdio.h>
#include <stdlib.h>

void return_str(char* str) {
       snprintf(str, 30, "Hello, Upgraded Packager!");
}
```

@**user**: **`hello-upgrade-pkg/fun.h`**

```
void return_str(char* str);
```

@**user**: **`hello-upgrade-pkg/prog.c`**

```
#include <stdio.h>
#include <stdlib.h>
#include "fun.h"

int main(void) {
       char* result = malloc(30 * sizeof(char));
       return_str(result);
       printf("%s\n", result);
       free(result);
       return 0;
}
```

@**user**: **`hello-upgrade-pkg/Makefile`**

```
TRASH = *.o *~ .gear/*~
GENS = hello-upgrade
CC = cc
CFLAGS = -O0 -g -Wall

hello-upgrade:  prog.o fun.o
```

```
	$(CC) $(CFLAGS) $^ -o $@
clean:
	rm -f $(TRASH)

distclean:	clean
	rm -f $(GENS)
```

Теперь сделаем единый коммит и сохраним в виде патча все изменения (можно было сделать несколько коммитов, тогда при описании изменений рассматривались бы не соседние, а разнесённые в ветке коммиты, однако результат был бы таким же):

@user

```
[user@VM hello-upgrade-pkg]$ ls
fun.c  fun.h  Makefile  prog.c
[user@VM hello-upgrade-pkg]$ git status
On branch master
Changes not staged for commit:
 (use "git add <file>..." to update what will be committed)
 (use "git restore <file>..." to discard changes in working directory)
	modified:   Makefile
	modified:   prog.c

Untracked files:
 (use "git add <file>..." to include in what will be committed)
	fun.c
	fun.h

no changes added to commit (use "git add" and/or "git commit -a")

[user@VM hello-upgrade-pkg]$ git add fun.*
[user@VM hello-upgrade-pkg]$ git commit -a -m "Patch commit"
[master 9013147] Patch commit
4 files changed, 14 insertions(+), 4 deletions(-)
create mode 100644 fun.c
create mode 100644 fun.h

[user@VM hello-upgrade-pkg]$ git log --oneline
9013147 (HEAD -> master) Patch commit
fa6269f 1.0-alt1
e30b38f 1.0-alt1
9be6d7f 1.0-alt1
[user@VM hello-upgrade-pkg]$ git diff fa6269f 9013147 --patch --output=hello-
upgrade-1.0-alt1.patch

[user@VM hello-upgrade-pkg]$ cat hello-upgrade-1.0-alt1.patch
index 3f543ad..c39cafe 100644
--- a/Makefile
+++ b/Makefile
@@ -3,8 +3,8 @@ GENS = hello-upgrade
CC = cc
CFLAGS = -O0 -g -Wall

-hello-upgrade: prog.o
-	$(CC) $(CFLAGS) $< -o $@
+hello-upgrade: prog.o fun.o
+	$(CC) $(CFLAGS) $^ -o $@
clean:
	rm -f $(TRASH)
```

```
diff --git a/fun.c b/fun.c
new file mode 100644
index 0000000..5460923
--- /dev/null
+++ b/fun.c
@@ -0,0 +1,6 @@
+#include <stdio.h>
+#include <stdlib.h>
+
+void return_str(char* str) {
+       snprintf(str, 30, "Hello, Upgraded Packager!");
+}
diff --git a/fun.h b/fun.h
new file mode 100644
index 0000000..444a555
--- /dev/null
+++ b/fun.h
@@ -0,0 +1 @@
+void return_str(char* str);
diff --git a/prog.c b/prog.c
index 5c41659..c05ada9 100644
--- a/prog.c
+++ b/prog.c
@@ -1,8 +1,11 @@
#include <stdio.h>
-
-char str[] = "Hello, Packager!";
+#include <stdlib.h>
+#include "fun.h"

int main(void) {
-       printf("%s\n", str);
+       char* result = malloc(30 * sizeof(char));
+       return_str(result);
+       printf("%s\n", result);
+       free(result);
       return 0;
}
[user@VM hello-upgrade-pkg]$
```

Теперь откатим репозиторий на коммит до изменений и просто сохраним патч:

@user

```
[user@VM hello-upgrade-pkg]$ git reset --hard HEAD~
HEAD is now at fa6269f 1.0-alt1
[user@VM hello-upgrade-pkg]$ tree . .gear/
.
├── hello-upgrade-1.0-alt1.patch
├── Makefile
└── prog.c
.gear/
├── hello-upgrade.spec
└── rules

2 directories, 5 files
[user@VM hello-upgrade-pkg]$ git status
On branch master
```

```
Untracked files:
 (use "git add <file>..." to include in what will be committed)
       hello-upgrade-1.0-alt1.patch

nothing added to commit but untracked files present (use "git add" to track)
[user@VM hello-upgrade-pkg]$ git add hello-upgrade-1.0-alt1.patch
[user@VM hello-upgrade-pkg]$ vim .gear/hello-upgrade.spec
[user@VM hello-upgrade-pkg]$ gear-commit -a
[master 157f524] 1.0-alt1
2 files changed, 54 insertions(+)
create mode 100644 hello-upgrade-1.0-alt1.patch
[user@VM hello-upgrade-pkg]$
```

Добавим описание патча в spec-файл и **`.gear/rules`** для его переноса в hasher при сборке. В spec-файле необходимо указать название патча, а также команду применения патча после распаковки исходников (в директиву ***`%prep`*** добавляется специальный макрос ***`%autopatch`***):

@**user**: **`hello-upgrade-pkg/.gear/hello-upgrade.spec`**

```
Name: hello-upgrade
Version: 1.0
Release: alt1

Summary: Test pkg with patch

License: GPL-3.0-or-later
Group: Development/Other

Source0: %name-%version.tar.gz
Patch: %name-%version-%release.patch

%description
This is a small testing package, builded with patch

%prep
%setup
%autopatch -p1

%build
%make_build

%install
install -D %name %buildroot%_bindir/%name

%files
%_bindir/%name


%changelog
* Sat Aug 02 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial Build
- Add Patch
```

При сборке ***`%autopatch`*** раскрывается в последовательное применение патчей, заданных директивами ***`Patch:`*** и ***`Patch№:`***. Ключ ***`-p1`*** передаётся каждому запуску утилиты **Patch** — он означает, что в описании имён файлов внутри патча присутствует один «лишний» каталог. В самом деле, в описании патча, полученного с помощью **`git diff`** все изменённые файлы принадлежат псевдодиректориям **`a/`** и **`b/`**:

@**user**

```
[user@VM hello-upgrade-pkg]$ cat hello-upgrade-1.0-alt1.patch | grep -E "(a/|b/)"
diff --git a/Makefile b/Makefile
--- a/Makefile
+++ b/Makefile
diff --git a/fun.c b/fun.c
+++ b/fun.c
diff --git a/fun.h b/fun.h
+++ b/fun.h
diff --git a/prog.c b/prog.c
--- a/prog.c
+++ b/prog.c
[user@VM hello-upgrade-pkg]$
```

a/ — обозначает «старую» версию файла, **b/** — «новую» версию, между которыми производится сравнение. При применении патча эти директории будут проигнорированы утилитой (как раз благодаря флагу ***-p1***), но при этом повышается читаемость самого патча.

Соберём пакет и проверим, что патч применился:

@**user**

```
[user@VM hello-upgrade-pkg]$ gear-hsh --lazy
<...>
Executing(%prep): /bin/sh -e /usr/src/tmp/rpm-tmp.84343
+ umask 022
+ /bin/mkdir -p /usr/src/RPM/BUILD
+ cd /usr/src/RPM/BUILD
+ cd /usr/src/RPM/BUILD
+ rm -rf hello-upgrade-1.0
+ echo 'Source #0 (hello-upgrade-1.0.tar.gz):'
Source #0 (hello-upgrade-1.0.tar.gz):
+ /bin/tar -xf -
+ /usr/bin/gzip -dc /usr/src/RPM/SOURCES/hello-upgrade-1.0.tar.gz
+ cd hello-upgrade-1.0
+ /bin/chmod -c -Rf u+rwX,go-w .
+ echo 'Patch #0 (hello-upgrade-1.0-alt1.patch):'
Patch #0 (hello-upgrade-1.0-alt1.patch):
+ /usr/bin/patch -p1
patching file Makefile
patching file fun.c
patching file fun.h
patching file prog.c
+ exit 0
<...>
[user@VM hello-upgrade-pkg]$ cp ~/hasher/repo/x86_64/RPMS.hasher/hello-
upgrade-1.0-alt1.x86_64.rpm ~/hasher/chroot/.in/
[user@VM hello-upgrade-pkg]$ hsh-shell --rooter
```

@**rooter**

```
[root@localhost .in]# rpm -i hello-upgrade-1.0-alt1.x86_64.rpm
<13>Aug  3 07:55:29 rpm: hello-upgrade-1.0-alt1 1754206866 installed
[root@localhost .in]# hello-upgrade
Hello, Upgraded Packager!
[root@localhost .in]#
```

Глава 12. Автоматизация сборки библиотек и версионирование

При работе с проектами частой задачей становится создание и подключение библиотек. В библиотеки обычно выносится логика, общая для нескольких создаваемых исполняемых файлов, или публичное API, позволяющее использовать функции проекта в других разработках. Использование библиотек делает процесс сборки многоступенчатым: сначала надо скомпилировать исходники, затем из некоторых объектных файлов собрать библиотеку, а только затем — готовую программу. Если библиотека разделяемая, для запуска программы, которая её использует, необходимо вручную указывать место её расположения через ***LD_PRELOAD*** и ***LD_LIBRARY_PATH***. Для удобной работы с библиотеками и их тестирования необходимо использовать специальные инструменты по автосборке. Классическим инструментом сборки библиотек является GNU Libtool.

12.1. Использование Libtool вручную

Рассмотрим простую программу, которая прибавляет константу к числу, разобьём её на основной код, файл с основной функцией, файл с глобальной переменной-константой и заголовочный файл:

@user: **`inc-pkg/main.c`**

```
#include <stdio.h>
#include <stdlib.h>
#include "libinc.h"

int main(int argc, char *argv[]) {
        int n;

        if(argc < 2) {
          fprintf(stderr, "Usage: %s NUMBER\n", argv[0]);
          return 1;
        }

        n = atoi(argv[1]);
        printf("%d\n", inc(n));

 return 0;
}
```

@user: **`inc-pkg/fun.c`**

```
#include "libinc.h"

int inc(int var) {
  return var + inc_var;
}
```

@**user**: **inc-pkg/global.c**

```
#include "libinc.h"
int inc_var = 1;
```

@**user**: **inc-pkg/libinc.h**

```
int inc(int);
extern int inc_var;
```

Опишем **Makefile** с использованием **libtool** и рассмотрим подробнее используемые команды:

@**user**: **inc-pkg/Makefile**

```
CFLAGS = -g -O
LTFLAGS = --tag=CC

all:    inc

%.lo:   %.c
	libtool --mode=compile $(LTFLAGS) $(CC) -c $<
libinc.la: fun.lo global.lo
	libtool --mode=link $(LTFLAGS) $(CC) -o $@ $^ -rpath /usr/lib64
inc:    main.o libinc.la
	libtool --mode=link $(LTFLAGS) $(CC) -o $@ $^
check:  inc
	test "`./$< 123`" = "124"
clean:
	rm -rf *.so inc .libs *.l? *.o
```

Утилита **libtool** управляет и компиляцией, и компоновкой (определяется флагом ***--mode=***), и ей необходимо передавать дополнительные параметры (например, информацию о компиляторе), такие параметры мы храним в переменной ***LTFLAGS***. При этом базовые флаги компиляции и компоновки (например, ***-fPIC*** для разделяемых библиотек) добавляются автоматически.

В результате компиляции порождаются **.lo**-файлы (так называемые «libtool objects»). Libtool object — это shell-сценарий с мета-информацией о том, где лежат «настоящие» объектные файлы (причём как собранные с флагом ***-fPIC*** для динамической компоновки, так и без него для статической).

На основе **.lo**-файлов создаётся «libtool archive» (**.la**). Это тоже shell-сценарии с мета-информацией о собранных вариантах библиотек (статическом и динамическом).

Исполняемый файл, который порождается **libtool** — это, конечно, тоже shell-сценарий, в котором вся информация перерабатывается, выбирается тип сборки, при необходимости указывается место расположения «настоящей» динамической библиотеки, после чего вызывается соответствующая программа. Таким образом, результат работы **libtool** можно запускать сразу, минуя все эти стадии.

Простейшие примеры запуска **libtool** для получения **.lo**, **.la** и исполняемых файлов приведены выше.

Соберём проект и рассмотрим ближе некоторые файлы:

@**user**

```
[user@VM inc-pkg]$ make
cc -g -O   -c -o main.o main.c
libtool --mode=compile --tag=CC cc -c fun.c
libtool-default: compile:  cc -c fun.c  -fPIC -DPIC -o .libs/fun.o
libtool-default: compile:  cc -c fun.c -o fun.o >/dev/null 2>&1
libtool --mode=compile --tag=CC cc -c global.c
libtool-default: compile:  cc -c global.c  -fPIC -DPIC -o .libs/global.o
libtool-default: compile:  cc -c global.c -o global.o >/dev/null 2>&1
libtool --mode=link --tag=CC cc -o libinc.la fun.lo global.lo -rpath /usr/lib64
libtool-default: link: x86_64-alt-linux-gcc -shared  -fPIC -DPIC  .libs/
fun.o .libs/global.o      -Wl,-soname -Wl,libinc.so.0 -o .libs/libinc.so.0.0.0
libtool-default: link: (cd ".libs" && rm -f "libinc.so.0" && ln -s
"libinc.so.0.0.0" "libinc.so.0")
libtool-default: link: (cd ".libs" && rm -f "libinc.so" && ln -s
"libinc.so.0.0.0" "libinc.so")
libtool-default: link: ar cr .libs/libinc.a  fun.o global.o
libtool-default: link: ranlib .libs/libinc.a
libtool-default: link: ( cd ".libs" && rm -f "libinc.la" && ln -s "../libinc.la"
"libinc.la" )
libtool --mode=link --tag=CC cc -o inc main.o libinc.la
libtool-default: link: cc -o .libs/inc main.o  ./.libs/libinc.so
[user@VM inc-pkg]$
```

```
.
├── fun.c
├── fun.lo
├── fun.o
├── global.c
├── global.lo
├── global.o
├── inc
├── libinc.h
├── libinc.la
├── main.c
├── main.o
└── Makefile
.libs/
├── fun.o
├── global.o
├── inc
├── libinc.a
├── libinc.la -> ../libinc.la
├── libinc.lai
├── libinc.so -> libinc.so.0.0.0
├── libinc.so.0 -> libinc.so.0.0.0
└── libinc.so.0.0.0
```

Во-первых, среди получившихся файлов появились два файла **`inc`**. Находящийся в **`.libs/`** — непосредственно исполняемый файл. Находящийся в текущем каталоге — на самом деле shell-сценарий, в котором автоматически прописываются пути к динамическим библиотекам, и программа вызывается с их неявным указыванием:

@**user**

```
[user@VM inc-pkg]$ diff inc .libs/inc
Двоичные файлы inc и .libs/inc различаются
[user@VM inc-pkg]$ head -n1 inc
#! /bin/sh
```

```
[user@VM inc-pkg]$ grep "LD_LIBRARY_PATH" inc
relink_command="(cd /home/user/inc-pkg; { test -z \"\${LIBRARY_PATH+set}\" ||
unset LIBRARY_PATH || { LIBRARY_PATH=; export LIBRARY_PATH; }; }; { test -z \"\$
{COMPILER_PATH+set}\" || unset COMPILER_PA
TH || { COMPILER_PATH=; export COMPILER_PATH; }; }; { test -z \"\$
{GCC_EXEC_PREFIX+set}\" || unset GCC_EXEC_PREFIX || { GCC_EXEC_PREFIX=; export
GCC_EXEC_PREFIX; }; }; { test -z \"\${LD_RUN_PATH+set}\" || unset
LD_RUN_PATH || { LD_RUN_PATH=; export LD_RUN_PATH; }; }; { test -z \"\$
{LD_LIBRARY_PATH+set}\" || unset LD_LIBRARY_PATH || { LD_LIBRARY_PATH=; export
LD_LIBRARY_PATH; }; }; PATH=/home/user/.gemie/bin
:/home/user/bin:/usr/bin:/bin:/usr/local/bin:/usr/games; export PATH; cc -o \
$progdir/\$file main.o  ./.libs/libinc.so -Wl,-rpath -Wl,/home/user/inc-
pkg/.libs)"
[user@VM inc-pkg]$
[user@VM inc-pkg]$ ./inc 123
124
[user@VM inc-pkg]$ .libs/inc 123
.libs/inc: error while loading shared libraries: libinc.so.0: cannot open shared
object file: No such file or directory
[user@VM inc-pkg]$ LD_LIBRARY_PATH=`pwd`/.libs .libs/inc 123
124
[user@VM inc-pkg]$
```

.la- и **.lo**-файлы также не являются объектными, а лишь описывают параметры итоговой библиотеки для сборки:

@**user**

```
[user@VM inc-pkg]$ cat libinc.la
# libinc.la - a libtool library file
# Generated by libtool (GNU libtool) 2.4.7
#
# Please DO NOT delete this file!
# It is necessary for linking the library.

# The name that we can dlopen(3).
dlname='libinc.so.0'

# Names of this library.
library_names='libinc.so.0.0.0 libinc.so.0 libinc.so'

# The name of the static archive.
old_library='libinc.a'

# Linker flags that cannot go in dependency_libs.
inherited_linker_flags=''

# Libraries that this one depends upon.
dependency_libs=''

# Names of additional weak libraries provided by this library
weak_library_names=''

# Version information for libinc.
current=0
age=0
revision=0

# Is this an already installed library?
```

```
installed=no

# Should we warn about portability when linking against -modules?
shouldnotlink=no

# Files to dlopen/dlpreopen
dlopen=''
dlpreopen=''

# Directory that this library needs to be installed in:
libdir='/usr/lib64'
[user@VM inc-pkg]$ cat *.lo
# fun.lo - a libtool object file
# Generated by libtool (GNU libtool) 2.4.7
#
# Please DO NOT delete this file!
# It is necessary for linking the library.

# Name of the PIC object.
pic_object='.libs/fun.o'

# Name of the non-PIC object
non_pic_object='fun.o'

# global.lo - a libtool object file
# Generated by libtool (GNU libtool) 2.4.7
#
# Please DO NOT delete this file!
# It is necessary for linking the library.

# Name of the PIC object.
pic_object='.libs/global.o'

# Name of the non-PIC object
non_pic_object='global.o'

[user@VM inc-pkg]$
```

12.2. Версионирование

Подробнее рассмотрим получившуюся библиотеку для обсуждения версионирования:

@user

```
[user@VM inc-pkg]$ ls -la .libs/*.so*
lrwxrwxrwx 1 user user    15 авг  4 16:08 .libs/libinc.so -> libinc.so.0.0.0
lrwxrwxrwx 1 user user    15 авг  4 16:08 .libs/libinc.so.0 -> libinc.so.0.0.0
-rwxr-xr-x 1 user user 15416 авг  4 16:08 .libs/libinc.so.0.0.0
[user@VM inc-pkg]$
```

При работе с библиотеками (и, в целом, любыми объектами, требующими явный параметр для отслеживания изменений и совместимости) принято использовать специальные правила версионирования.

Во-первых, это формат хранения библиотек в виде одного файла с сопутствующими символьными ссылками. Бесчисловая версия используется для создания зависимостей на библиотеку при сборке (если не требуется строгая зависимость на конкретную версию), мажорная («основная», с одним числом) используется для указания на основную поддерживаемую версию, а последняя (единственная реальная библиотека) является самим рабочим объектом.

Во-вторых, это параметры, по которым ведётся отслеживание изменений, и правила версионирования. В случае с библиотеками версионирование описывается через изменение ABI — Application Binary Interface. Если API описывает, с какими параметрами следует вызывать функции библиотеки в *исходных текстах* программ, и какие значения они возвращают, то ABI отражает, во что именно эта договорённость превращается *после компиляции*. Таким образом, очень важно, чтобы исполняемый файл и разделяемая библиотека были *совместимы по ABI*: если такая совместимость есть, пускай даже при сборке программы использовалась библиотека слегка другой версии, программа будет работать штатно. Если совместимости нет, то *лучшее*, что может сделать библиотека — это немедленно завершить программу с ошибкой.

Повсеместно используемые большие библиотеки, например, GLibc, могут быть совместимы сразу с несколькими ABI, для более простых случаев **libtool** предлагает схему из трёх счётчиков — current:revision:age:

- ***current*** увеличивается при каждом изменении ABI — это т. н. «номер интерфейса»;
- ***revision*** увеличивается при каждом изменении в библиотеке, которое не отразилось на ABI (исправление ошибок, изменение функциональности и т. п.); если ***current*** увеличилось, ***revision*** обнуляется;
- ***age*** изменяется вместе с ***current***; он увеличивается при каждом *обратно совместимом* изменении (например, при добавлении новой функции или при изменении, не затрагивающем ABI), а в противном случае — обнуляется.

Таким образом, libtool versioning решает сразу три задачи:

1. При *любом* изменении версия увеличивается;
2. При изменении ABI версия увеличивается настолько, что становится больше любых версий, в которых ABI не поменялось;
3. По версии можно узнать *диапазон* предоставляемых «номеров интерфейса», с которыми гарантирована обратная совместимость — это диапазон от current до ***current*** — ***age***.

При более простом *семантическом* версионировании описание ведётся значениями ***MAJOR.MINOR.PATCH*** и меняется по следующим правилам:

- обратно **не**совместимые изменения ABI (удаление / изменение) — ***MAJOR++.MINOR=0.PATCH=0***;
- обратно совместимые изменения ABI (добавление) — ***MAJOR.MINOR++.PATCH=0***;
- обратно совместимые исправления ABI и изменения, не затрагивающие ABI — ***MAJOR.MINOR.PATCH++***.

12.3. Автоматизация Libtool

Поскольку **libtool** является классическим инструментом автосборки библиотек, в **autotools** есть поддержка его обработки.

Для начала соберём классический **`configure.ac`**:

@user: **`inc-pkg/configure.ac`**

```
AC_INIT([inc], [1.0], [UsamG1t])
AC_CONFIG_SRCDIR([src/main.c])
AM_INIT_AUTOMAKE([foreign subdir-objects])
AC_CONFIG_HEADERS([config.h])

# Checks for programs.
AC_PROG_CC

# Checks for libraries.

# Checks for header files.
AC_CHECK_HEADERS([stdlib.h])

# Optional clues

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.
AC_FUNC_ERROR_AT_LINE

AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

Перенесём исходники в отдельную поддиректорию и сделаем **`Makefile.am`** для обработки:

```
.
├── configure.ac
├── Makefile.am
└── src
    ├── fun.c
    ├── global.c
    ├── libinc.h
    ├── main.c
    └── Makefile.am
```

@user: **`inc-pkg/Makefile.am`**

```
SUBDIRS = src
```

@user: **`inc-pkg/src/Makefile.am`**

```
CFLAGS  = -Wall -O0 -g

bin_PROGRAMS = inc
inc_SOURCES = main.c fun.c global.c
```

```
fun.c global.c: libinc.h

check:  inc
       test "`./$< 123`" = "124"
```

Сборка уже работает корректно, однако получился просто многофайловый проект. Добавим обработку **libtool**: в **`configure.ac`** добавим инициализацию **libtool** и укажем явный запрет сборки статической библиотеки, в **`src/Makefile.am`** укажем сборку библиотеки из исходников (***`lib_LTLIBRARIES`***), а также связь библиотеки с исполняемым файлом (**`inc_LDADD`**):

@**user**: **`inc-pkg/configure.ac`**

```
AC_INIT([inc], [1.0], [UsamG1t])
AC_CONFIG_SRCDIR([src/main.c])
AM_INIT_AUTOMAKE([foreign subdir-objects])
LT_INIT([disable-static])
AC_CONFIG_HEADERS([config.h])

# Checks for programs.
AC_PROG_CC

# Checks for libraries.

# Checks for header files.
AC_CHECK_HEADERS([stdlib.h])

# Optional clues

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.
AC_FUNC_ERROR_AT_LINE

AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

@**user**: **`inc-pkg/src/Makefile.am`**

```
CFLAGS = -Wall -O0 -g

lib_LTLIBRARIES = libinc.la
libinc_la_SOURCES = fun.c global.c

bin_PROGRAMS = inc
inc_SOURCES = main.c
inc_LDADD = libinc.la

BUILT_SOURCES = libinc.h

check: inc
        test "`./$< 123`" = "124"
```

Добавим ***`.gear/rules`*** и spec-файл для сборки пакета:

@**user**: **`inc-pkg/.gear/rules`**

```
spec:   .gear/inc.spec
tar.gz: . name=@name@-@version@
```

@**user**: **`inc-pkg/.gear/inc.spec`**

```
Name: inc
Version: 1.0
Release: alt1

Summary: Test pkg with libtool

License: GPL-3.0-or-later
Group: Development/Other

Source0: %name-%version.tar.gz

%description
This is a small testing package, builded with libtool

%prep
%setup

%build
%autoreconf
%configure
%make_build

%install
%makeinstall_std

%check
make check

%files
%_bindir/%name
%_libdir/*

%changelog
* Tue Aug 05 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial Build
```

Напомним, что в **autotools** задаются места для установки всех компонентов проекта, а в макросе **`%configure`** они определены в соответствии с дисциплиной оформления пакетов ALT. Библиотека **`libinc.so`** автоматически установится в **`/usr/lib64`**, и при описании файлов пакета (директива **`%files`**) необходимо будет просто указать директорию **`%_libdir`**:

```
.
├── configure.ac
├── Makefile.am
└── src
    ├── fun.c
    ├── global.c
    ├── libinc.h
    ├── main.c
```

```
   └── Makefile.am
.gear/
├── inc.spec
└── rules
```

@user

```
[user@VM inc-pkg]$ gear-hsh --lazy
<...>
Executing(%build): /bin/sh -e /usr/src/tmp/rpm-tmp.60211
<...>
+ make -j2
<...>
/bin/sh ../libtool  --tag=CC   --mode=compile x86_64-alt-linux-gcc
-DHAVE_CONFIG_H -I. -I..     -Wall -O0 -g -c -o fun.lo fun.c
libtool: compile:  x86_64-alt-linux-gcc -DHAVE_CONFIG_H -I. -I.. -Wall -O0 -g -c
fun.c  -fPIC -DPIC -o .libs/fun.o
/bin/sh ../libtool  --tag=CC   --mode=compile x86_64-alt-linux-gcc
-DHAVE_CONFIG_H -I. -I..     -Wall -O0 -g -c -o global.lo global.c
libtool: compile:  x86_64-alt-linux-gcc -DHAVE_CONFIG_H -I. -I.. -Wall -O0 -g -c
global.c  -fPIC -DPIC -o .libs/global.o
/bin/sh ../libtool  --tag=CC   --mode=link x86_64-alt-linux-gcc  -Wall -O0 -g
-o libinc.la -rpath /usr/lib64 fun.lo global.lo
libtool: link: x86_64-alt-linux-gcc -shared  -fPIC -DPIC  .libs/fun.o .libs/
global.o    -O0 -g   -Wl,-soname -Wl,libinc.so.0 -o .libs/libinc.so.0.0.0
libtool: link: (cd ".libs" && rm -f "libinc.so.0" && ln -s "libinc.so.0.0.0"
"libinc.so.0")
libtool: link: (cd ".libs" && rm -f "libinc.so" && ln -s "libinc.so.0.0.0"
"libinc.so")
libtool: link: ( cd ".libs" && rm -f "libinc.la" && ln -s "../libinc.la"
"libinc.la" )
/bin/sh ../libtool  --tag=CC   --mode=link x86_64-alt-linux-gcc  -Wall -O0 -g
-o inc main.o libinc.la
libtool: link: x86_64-alt-linux-gcc -Wall -O0 -g -o .libs/inc main.o  ./.libs/
libinc.so
<...>
```

```
Executing(%install): /bin/sh -e /usr/src/tmp/rpm-tmp.31330
<...>
make[3]: Entering directory '/usr/src/RPM/BUILD/inc-1.0/src'
/usr/bin/mkdir -p '/usr/src/tmp/inc-buildroot/usr/lib64'
/bin/sh ../libtool   --mode=install /usr/libexec/rpm-build/install -p   libinc.la
'/usr/src/tmp/inc-buildroot/usr/lib64'
libtool: install: /usr/libexec/rpm-build/install -p .libs/libinc.so.0.0.0 /usr/
src/tmp/inc-buildroot/usr/lib64/libinc.so.0.0.0
libtool: install: (cd /usr/src/tmp/inc-buildroot/usr/lib64 && { ln -s -f
libinc.so.0.0.0 libinc.so.0 || { rm -f libinc.so.0 && ln -s libinc.so.0.0.0
libinc.so.0; }; })
libtool: install: (cd /usr/src/tmp/inc-buildroot/usr/lib64 && { ln -s -f
libinc.so.0.0.0 libinc.so || { rm -f libinc.so && ln -s libinc.so.0.0.0
libinc.so; }; })
libtool: install: /usr/libexec/rpm-build/install -p .libs/libinc.lai /usr/src/
tmp/inc-buildroot/usr/lib64/libinc.la
libtool: warning: remember to run 'libtool --finish /usr/lib64'
/usr/bin/mkdir -p '/usr/src/tmp/inc-buildroot/usr/bin'
 /bin/sh ../libtool   --mode=install /usr/libexec/rpm-build/install -p inc '/usr/
src/tmp/inc-buildroot/usr/bin'
libtool: warning: 'libinc.la' has not been installed in '/usr/lib64'
```

```
libtool: install: /usr/libexec/rpm-build/install -p .libs/inc /usr/src/tmp/inc-
buildroot/usr/bin/inc
<...>
Adjusting library links in /usr/src/tmp/inc-buildroot
./usr/lib64: (from <cmdline>:0)
       libinc.so.0 -> libinc.so.0.0.0
Verifying ELF objects in /usr/src/tmp/inc-buildroot
(arch=normal,fhs=normal,lfs=relaxed,lint=relaxed,rpath=normal,stack=normal,textre
l=normal,unresolved=normal)
Splitting links to aliased files under /{,s}bin in /usr/src/tmp/inc-buildroot
```

```
Executing(%check): /bin/sh -e /usr/src/tmp/rpm-tmp.55059
<...>
test "`./inc 123`" = "124"
<...>
Wrote: /usr/src/RPM/SRPMS/inc-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/inc-1.0-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/inc-debuginfo-1.0-alt1.x86_64.rpm (w2.lzdio)
7.09user 5.34system 0:15.58elapsed 79%CPU (0avgtext+0avgdata 22740maxresident)k
0inputs+17184outputs (0major+731782minor)pagefaults 0swaps
```

@user

```
[user@VM inc-pkg]$ cp ~/hasher/repo/x86_64/RPMS.hasher/inc-1.0-alt1.x86_64.rpm ~/
hasher/chroot/.in/
[user@VM inc-pkg]$ hsh-shell --rooter
```

@rooter

```
[root@localhost .in]# rpm -i inc-1.0-alt1.x86_64.rpm
<13>Aug  5 12:39:54 rpm: inc-1.0-alt1 1754397543 installed
[root@localhost .in]# which inc
/usr/bin/inc
[root@localhost .in]# inc 123
124
[root@localhost .in]# ls -la /usr/lib64/libinc*
lrwxrwxrwx 1 root root    15 Aug  5 12:37 /usr/lib64/libinc.so -> libinc.so.0.0.0
lrwxrwxrwx 1 root root    15 Aug  5 12:37 /usr/lib64/libinc.so.0 ->
libinc.so.0.0.0
-rw-r--r-- 1 root root 14232 Aug  5 12:37 /usr/lib64/libinc.so.0.0.0
[root@localhost .in]#
```

Стоит отдельно заметить, что установка **`.la`**- и **`.lo`**-файлов в систему запрещена дисциплиной оформления пакетов ALT Linux Team (в том числе и из-за соображений безопасности). Библиотеки и использующие их приложения должны быть скомпилированы так, чтобы работать без дополнительной прослойки, требующей запуска Shell. Случайно установленные в ***%buildroot*** файлы таких типов просто удаляются на этапе формирования пакета.

Глава 13. Сборка большого проекта и тестирование

13.1. Сборка большого проекта

Изучив основной спектр инструментов по разработке, отладке и автоматизации сборки пакетов, уже можно заниматься разработкой проектов под Альт. Вследствие этого последующие главы будут описывать работу с одним проектом.

За основу взят проект syscall, позволяющий работать с системными вызовами из командной строки. В рамках адаптации проекта под формат Альт-пакета необходимо добавить автоматическую конфигурацию для автосборки (**autotools** для конфигурации и **libtool** — после того, как выделим библиотеку из монолитного исходного текста). Также необходимо распределить файлы по каталогам в соответствии со общепринятой практикой.

Итоговое дерево директорий и пакет с исходными текстами преобразованного проекта:

```
.
├── configure.ac
├── doc
│   ├── Makefile.am
│   └── syscall.pod
├── LICENSE
├── Makefile.am
└── src
    ├── basic.c
    ├── globals.c
    ├── lssyscalls
    ├── Makefile.am
    ├── syscall.c
    ├── syscall.h
    └── utility.c
.gear/
├── rules
└── syscall.spec
```

Для работы с данным проектом его необходимо развернуть в gear-репозиторий с помощью ***`gear-srpmimport`*** (как описано в главе о работе со сценариями).

13.2. Тестирование проекта

Тестирование является одним из важнейших этапов разработки. Оно позволяет разобрать семантику кода, проверить его работу, а также работу и состояние того информационного пространства, которое этот код создаёт и с которым взаимодействует (глобальные переменные, макросы, переменные окружения).

Выделяют 4 вида тестирования:

- **модульное** — проверяет работу отдельных модулей написанного продукта;
- **системное** — проверяет работу всей системы изолированно от внешних зависимостей;
- **интеграционное** — проверяет работу в рамках среды, в которой она запускается, и на, вообще говоря, не нами задаваемых условиях среды;
- **приёмочное тестирование** — проверяет внешнее воздействие на проект, внесение не встроенных тестов и проверка поведения системы вообще на её взаимодействие с программами.

Объектом обсуждения этой главы является модульное тестирование по дисциплине xUnit, задающее собственный путь тестирования системы.

В рамках Unit-тестов определён **раннер** (test **runner**) — основная программа, запускающая тесты. Он отвечает за проверку подключаемости тестирующего кода, фильтрацию тестов, сбор информации по запускам и т. д. Все **тесты (test)** объединены в отдельные **тестовые комплекты (test case)**, проверяющие какое-то конкретное свойство проекта. Множества комплектов, описывающих один раздел работы с проектом объединены в **тестовые случаи (test suite)**. Как правило, в один тестовый случай входят тесты, работающие в одинаковых или похожих условиях, а если условия существенно разные (например, бэкенд в виде файла и бэкенд в базе данных) — в отдельные.

Для более точного понимания можно запомнить, что случай показывает, «в каких условиях сломалось?», комплект — «какая функциональность сломалась?", а сам тест — «что именно сломалось?»

Следующий важный блок любого тестирования — подготовка и ликвидация тестируемого окружения — **test fixture**, или *фикстуры*. Фикстуры могут создаваться для каждого из уровней тестирования и описывать все временно создаваемые данные и предварительно выполняемые действия для проверки работоспособности системы.

Одним из критериев качества тестирования является **test coverage** — покрытие, описывающее полноту тестирования программы. Самый простой показатель — процент строк исходного текста проекта, которые были выполнены в ходе тестирования. Более сложные показатели включают в себя проверку различных **execution path** — путей выполнения кода: например, если в одно и то же место программы можно попасть, предварительно пройдя и клаузу **`if`** условного оператора, и клаузу **`else`**, нужно иметь *два набора* тестов соответствующего места.

Одна из классических подсистем тестирования проектов на Си — Check. Работа с ней также автоматизирована в **autotools**, что позволяет встраивать её в проекты и проводить тестирование при сборке пакетов без дополнительных действий.

Добавим тестирование в проект. Для начала укажем зависимость на ***`check`***, воспользовавшись тем, что соответствующие проверки пакет *check* добавляет в утилиту определения зависимостей pkgconfig:

@user: **`syscall-master/configure.ac`**

```
@@ 11,6 11,9 @@ AC_CONFIG_HEADERS([config.h])
 # Checks for programs.
 AC_PROG_CC

+# Joint pkgconfig library/include check and variable definition.
+PKG_CHECK_MODULES([CHECK],[check])
+
 # Checks for libraries.

 # Checks for header files.
```

Поскольку check — это просто библиотека, тесты — это обычные функции на Си. Каждая тестирующая программа состоит из нескольких частей:

- сами тесты;
- задание тестового комплекта, тестовых случаев, раннера и, возможно, фикстур;

- регистрация: какие случаи принадлежат каким комплектам, какие тесты принадлежат каким случаям;
- запуск раннера;
- освобождение ресурсов.

Рассмотрим минимальный пример с простейшим тестом — проверкой на наличие соответствующей функции:

```
#include <stdio.h>
#include <assert.h>
#include <check.h>
#include "syscall.h"

START_TEST(parse_arg_incl) {
        assert(parse_arg != NULL);
}
END_TEST

int main(int argc, char *argv[]) {
        Suite *suite = suite_create("incl");
        TCase *testcase = tcase_create("incl");
        SRunner *runner = srunner_create(suite);
        int ret;

        suite_add_tcase(suite, testcase);
        tcase_add_test(testcase, parse_arg_incl);
        srunner_run_all(runner, CK_ENV);
        ret = srunner_ntests_failed(runner);
        srunner_free(runner);
        return ret != 0;
}
```

В примере видно, что:

- тесты описываются внутри специальных макросов ***START_TEST/END_TEST*** (превращаются в функции);
- создание объектов тестирования, их регистрация друг в друге, запуск раннера и анализ результатов — отдельные атомарные операции;
- раннер создаётся на базе хотя бы одного тестового комплекта (потом туда можно добавить ещё);
- по окончании тестирования программист обязан вызвать ***srunner_free()***, освобождая память не только самого раннера, но и рекурсивно всех зарегистрированных в нём объектов.

Ручное составление тестовых файлов — довольно монотонная и долгая задача. Однако большая часть тестового файла вполне может быть сгенерирована, для этого в пакет *libcheck* входит утилита **checkmk**. Тестовые файлы пишутся всё так же на Си, но общая часть задаётся директивами, похожими на Си-препроцессор, а в **`Makefile.am`** добавляется вызов **checkmk**:

@**user**: **`syscall-master/tests/Makefile.am`**

```
TESTS = include upstream
check_PROGRAMS = include upstream

.ts.c:
	checkmk $< > $@

AM_CFLAGS = -I$(top_builddir)/src @CHECK_CFLAGS@
LDADD = $(top_builddir)/src/libsyscall.la @CHECK_LIBS@
```

Традиционно исходники для **checkmk** имеют расширение **`.ts`** (от test suite), поэтому многие текстовые редакторы пытаются включить подсветку синтаксиса *TypeScript* — она, конечно, плохо подходит.

В самих тестах используются специальные функции-макросы для проверки значений. Проверка включает в себя работу как с локальными переменными тестов, так и с глобальными значениями, как, например, указатели на функции библиотеки.

Воспроизведём пример выше с использованием макросов, и также добавим проверку на наличие ещё одной функции:

@**user**: **`syscall-master/tests/include.ts`**

```
#include <check.h>
#include "syscall.h"

#test include
	ck_assert_ptr_nonnull(parse_arg);
	ck_assert_ptr_nonnull(lookup);
```

В более существенном примере проверим работоспособность некоторых функций:

- тестовые случаи описывают функциональность программы из разных файлов библиотеки;
- тестовые комплекты проверяют функциональность конкретной функции;
- тесты описывают ситуации с конкретными параметрами:
  - разные или одинаковые имена системных вызовов;
  - бинарный поиск системного вызова в таблице;
  - обработку аргумента системного вызова.

В качестве параметров тестируемых функций могут выступать как локальные переменные отдельного теста, так и глобальные переменные:

@**user**: **`syscall-master/tests/upstream.ts`**

```
#include <check.h>
#include "syscall.h"

int res;
unsigned long ulres;

#suite utility
#tcase scomp
#test diff_syscalls
	Syscall sys1 = {"write", 0};
```

```
       Syscall sys2 = {"read", 0};
       res = scomp(&sys1, &sys2);
       ck_assert_int_gt(res, 0);

#test same_syscalls
       Syscall sys1 = {"open", 0};
       Syscall sys2 = {"open", 0};
       res = scomp(&sys1, &sys2);
       ck_assert_int_eq(res, 0);

#tcase lookup
#test found
       char name[] = "write";
       res = lookup(name);
       ck_assert_int_eq(res, 1);

char syscall_name[] = "write";

#suite basic
#tcase parse_arg
#test arg_length
       char arg[] = "#hello";
       ulres = parse_arg(syscall_name, arg);
       ck_assert_uint_eq(ulres, 5);

#test arg_retval
       ret_values[12] = 42;

       char arg[] = "$12";
       ulres = parse_arg(syscall_name, arg);
       ck_assert_uint_eq(ulres, 42);

#test arg_number
       char arg[] = "100500";
       ulres = parse_arg(syscall_name, arg);
       ck_assert_uint_eq(ulres, 100500);
```

Подробнее рассмотрим обновлённый spec-файл:

@**user**: **`syscall-master/.gear/syscall.spec`**

```
Name: syscall
Version: 1.1
Release: alt1

Summary: send system calls from your shell
URL: https://github.com/oliwer/syscall
License: ISC
Group: Other

Source0: %name-%version.tar.gz

# Automatically added by buildreq on Fri Aug 08 2025
# optimized out: glibc-kernheaders-generic glibc-kernheaders-x86 gnu-config
libgpg-error perl perl-Encode perl-Pod-Escapes perl-Pod-Simple perl-parent perl-
podlators sh5
BuildRequires: perl-Pod-Usage libcheck-devel check

%description
```

```
Execute a list of raw system calls. All the system calls listed in your system's
unistd.h are supported, with up to 5 arguments. A maximum of 20 calls can be
executed per invocation, each separated by a comma.

Arguments starting by a # symbol are used to give a string length.
For instance, #hello would be evaluated as 5.

Arguments starting by a $ followed by a number from 0 to 19 refer to a previous
system call return code. For instance, $0 refers to to the return code of the
first system call executed.
To display those values, use the echo built-in command.

The echo command can be used like any other system call
to easily display $ or # values, or any string or number.

%prep
%setup

%build
%autoreconf
%configure
%make_build

%install
%makeinstall_std

%check
make check

%files
%_bindir/%name
%_libdir/*
%_man1dir/*

%changelog
* Fri Aug 08 2025 UsamG1t <usamg1t@altlinux.org> 1.1-alt1
- Add xUnit check

* Fri Aug 08 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial Build
```

Поскольку исходный текст проекта расположен на отдельном сетевом ресурсе, в spec-файле должно быть добавлено указание на него, для этого используется директива ***URL***. Также, поскольку в проекте уже указана лицензия (о том, какая именно лицензия, а также их разновидности и отличия, будет рассказано в будущей главе), в spec-файле указывается она (заметим, что смена лицензии *конкретно в этом случае* была бы допустима, но явной необходимости для этого нет). Наконец, к зависимости на генератор документации нужно также добавить сборочные зависимости на пакет *check* и на его C-библиотеку.

Итоговый Gear-репозиторий выглядит так:

```
.
├── configure.ac
├── doc
│   ├── Makefile.am
│   └── syscall.pod
├── LICENSE
├── Makefile.am
```

```
├── src
│   ├── basic.c
│   ├── globals.c
│   ├── lssyscalls
│   ├── Makefile.am
│   ├── syscall.c
│   ├── syscall.h
│   └── utility.c
└── tests
    ├── include.ts
    ├── Makefile.am
    └── upstream.ts
.gear/
├── rules
└── syscall.spec
```

@user

```
[user@VM syscall-master]$ gear-hsh --lazy
<...>
Executing(%check): /bin/sh -e /usr/src/tmp/rpm-tmp.70349
+ umask 022
+ /bin/mkdir -p /usr/src/RPM/BUILD
+ cd /usr/src/RPM/BUILD
+ cd syscall-1.1
+ make check
<...>
checkmk include.ts > include.c
<...>
PASS: include
PASS: upstream
============================================================================
Testsuite summary for syscall 1.0
============================================================================
# TOTAL: 2
# PASS:  2
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
<...>
Wrote: /usr/src/RPM/SRPMS/syscall-1.1-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/syscall-1.1-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/syscall-debuginfo-1.1-alt1.x86_64.rpm (w2.lzdio)
11.68user 11.20system 0:23.99elapsed 95%CPU (0avgtext+0avgdata 24184maxresident)k
760inputs+19696outputs (0major+888811minor)pagefaults 0swaps
[user@VM syscall-master]$

[user@VM syscall-master]$ cp ~/hasher/repo/x86_64/RPMS.hasher/syscall-1.1-
alt1.x86_64.rpm ~/hasher/chroot/.in/
[user@VM syscall-master]$ hsh-shell --rooter
```

@rooter

```
[root@localhost .in]# rpm -i syscall-1.1-alt1.x86_64.rpm
<13>Aug  8 11:55:52 rpm: syscall-1.1-alt1 1754653927 installed
[root@localhost .in]# which syscall
/usr/bin/syscall
[root@localhost .in]# ls /usr/share/man/man1/syscall.1.xz
/usr/share/man/man1/syscall.1.xz

[root@localhost .in]# cd
[root@localhost ~]# syscall open my.file e101 0755 , write '$0' hello '#hello' ,
close '$0'
[root@localhost ~]# cat my.file
hello
[root@localhost ~]#
```

13.3. Тестовое покрытие

Теперь для версии 1.2 добавим измерения покрытия программы тестами. Для этого добавим в **`Makefile.am`** дополнительный рецепт для запуска утилиты gcov, которая обрабатывает результаты тестирования:

@**user**: **`syscall-master/Makefile.am`**

```
SUBDIRS = src doc tests

checklog:       check
        cat tests/*.log

gcov:   check
        $(MAKE) -C src gcov
```

- для ручного показа статистики по тестированию добавлен рецепт ***`checklog`***;
- рецепт ***`make gcov`*** в основном **`Makefile`** требует предварительного запуска тестов (***`check`***), после чего вызывает ***`make gcov`*** в каталоге **`src`**.

Собирать статистику о выполнении *строк исходного текста* можно только если программа особым образом скомпилирована: в её код добавлено постоянное обновление многочисленных счётчиков, проверка достижимости и т. п. Исполняемые файлы, скомпилированные для проверки покрытия, непригодны к *эксплуатации*: они работают гораздо медленнее, потребляют больше ресурсов и при запуске создают множество временных файлов с отчётами:

@**user**: **`syscall-master/src/Makefile.am`**

```
CFLAGS  = -Wall -O0 -g
if COND_GCOV
CFLAGS  += -fprofile-arcs -ftest-coverage
endif

lib_LTLIBRARIES=libsyscall.la
libsyscall_la_SOURCES=globals.c utility.c basic.c
BUILT_SOURCES = syscall.h systab.h

bin_PROGRAMS = syscall
syscall_SOURCES = syscall.c
syscall_LDADD = libsyscall.la
```

```
systab.h:
	sh lssyscalls | awk '{printf "{\"%s\", %d},\n", $$1, $$2}' > $@

gcov:
	gcov -o .libs basic -t -k -b | grep -v -E '[[:digit:]]:[ /][*]'
	gcov -o .libs utility -t -k -b | grep -v -E '[[:digit:]]:[ /][*]'
```

Утилита **gcov** принимает на вход имя файла со статистикой покрытия (без расширения) которое соответствует **`.c`**-файлу из директории исходных текстов. Файлы статистики формируются в рабочей директории **`.libs`**, её надо указать с помощью ключа ***`-o`***. Эти файлы собирают информацию о количестве отработанных в процессе тестирования строк кода и другие статистические параметры. Из данной статистики нас будут интересовать выводы по количеству выполнения строк кода и, дополнительно (благодаря ключу ***`-b`***), вариантах обработки условий (в условных операторах / switch-case конструкции и т. д.):

@user

```
[user@VM syscall-master]$ make COND_GCOV=1 gcov
Making check in src
<...>
make -C src gcov
gcov -o .libs basic -t -k -b | grep -v -E '[[:digit:]]:[ /][*]'
       -:    0:Colorization: profile count: zero coverage (exceptional) zero
coverage (unexceptional) unexecuted block
       -:    0:Source:basic.c
       -:    0:Graph:.libs/basic.gcno
       -:    0:Data:.libs/basic.gcda
       -:    0:Runs:14
       -:    1:#include "syscall.h"
       -:    2:
function parse_arg called 6 returned 100% blocks executed 81%
       6:    3:unsigned long parse_arg(const char *syscall_name, char *arg)
       -:    4:{
       -:    5:  unsigned long num;
       6:    6:  char *endp = NULL;
       -:    7:
       -:    8:  /* #hello - length of a string */
       6:    9:  if (arg[0] == '#') {
branch  0 taken 33% (fallthrough)
branch  1 taken 67%
       2:   10:    return (unsigned long)strlen(arg + 1);
       -:   11:  }
       -:   12:
       -:   13:  /* $0 - return value of a previous syscall */
       4:   14:  if (arg[0] == '$') {
branch  0 taken 50% (fallthrough)
branch  1 taken 50%
       2:   15:    num = strtoul(arg + 1, &endp, 10);
call    0 returned 100%
       2:   16:    if (num < CMD_MAX - 1) {
branch  0 taken 100% (fallthrough)
branch  1 taken 0%
       2:   17:      return (unsigned long)ret_values[num];
       -:   18:    }
       -:   19:  }
       -:   20:
       -:   21:  /* Try to parse it as a number */
       2:   22:  endp = NULL;
```

```
       2:   23:  num = strtoul(arg, &endp, 0);
call    0 returned 100%
       2:   24:  if (errno == ERANGE) {
branch  0 taken 0% (fallthrough)
branch  1 taken 100%
       0:   25:    errx(1, "%s: argument '%s' is out of range",
call    0 never executed
       -:   26:      syscall_name, arg);
       -:   27:  }
       2:   28:  if (endp && *endp == '\0') {
branch  0 taken 100% (fallthrough)
branch  1 taken 0%
branch  2 taken 100% (fallthrough)
branch  3 taken 0%
       -:   29:    /* strtoul succeeded */
       2:   30:    return num;
       -:   31:  }
       -:   32:
       -:   33:  /* assume it is a string */
       -:   34:  /* unescape any \n at the end of the string */
       0:   35:  unescape_nl(arg);
call    0 never executed
       0:   36:  return (unsigned long)arg;
       -:   37:}
       -:   38:
gcov -o .libs utility -t -k -b | grep -v -E '[[:digit:]]:[ /][*]'
<...>
[user@VM syscall-master]$ ls -la src/.libs/ src | grep -E "\.gc??"
-rw-r--r-- 1 builder builder  1668 Aug 11 22:58 syscall.gcno
-rw-r--r-- 1 builder builder    228 Aug 11 22:58 basic.gcda
-rw-r--r-- 1 builder builder   5392 Aug 11 22:58 basic.gcno
-rw-r--r-- 1 builder builder     61 Aug 11 22:58 globals.gcno
-rw-r--r-- 1 builder builder    208 Aug 11 22:58 utility.gcda
-rw-r--r-- 1 builder builder   1836 Aug 11 22:58 utility.gcno
```

Для удобства можно добавить задание **COND_GCOV** параметром ***configure***:

@**user**: **syscall-master/configure.ac**

```
<...>

# Variable definitins
AC_SUBST(CK_VERBOSITY, verbose)
AC_ARG_VAR(CK_VERBOSITY, [Default: "verbose", can be "silent", "minimal" or
"normal")])

# Enabe/disable things
AC_ARG_ENABLE([gcov],
              [AS_HELP_STRING([--enable-gcov], [use gcov to collect test suite
coverage])],
              [], [enable_gcov=no])

AM_CONDITIONAL([COND_GCOV],[test '!' "$enable_gcov" = no])

# Checks for libraries.

# Checks for header files.
<...>
```

Поскольку пакет с исполняемыми файлами, скомпилированными для запуска **gcov**, был бы не пригоден для эксплуатации, необходимо проводить тестирование на независимой сборке. Она будет проводиться в отдельной вспомогательной директории, а запуск проверки будет контролироваться с помощью макросов ***%def_enable*** для отслеживания ключа запуска тестирования и ***%if_enabled*** / ***%endif*** для обработки случая тестирования. В сам пакет же будут добавлены скомпилированные обычным образом файлы:

@user: **syscall-master/.gear/syscall.spec**

```
%def_enable gcov

Name: syscall
Version: 1.2
Release: alt1

<...>

%build
%autoreconf

%if_enabled gcov
mkdir gcov_check && cd gcov_check
../configure --enable-gcov
%make_build
cd ..
%endif

%configure
%make_build

%install
%makeinstall_std

%check
%if_enabled gcov
  cd gcov_check
  make gcov
  cd ..
%else
  make check
%endif

%files
%_bindir/%name
%_libdir/*
%_man1dir/*

%changelog
* Mon Aug 11 2025 UsamG1t <usamg1t@altlinux.org> 1.2-alt1
- Add gcov

* Fri Aug 08 2025 UsamG1t <usamg1t@altlinux.org> 1.1-alt1
- Add xUnit check

* Fri Aug 08 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial Build
```

@user

```
[user@VM syscall-master]$ gear-hsh --lazy
<...>
Executing(%check): /bin/sh -e /usr/src/tmp/rpm-tmp.73347
+ umask 022
+ /bin/mkdir -p /usr/src/RPM/BUILD
+ cd /usr/src/RPM/BUILD
+ cd syscall-1.2
+ make check
<...>
checkmk include.ts > include.c
<...>
PASS: include
PASS: upstream
============================================================================
Testsuite summary for syscall 1.0
============================================================================
# TOTAL: 2
# PASS:  2
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
<...>
make -C src gcov
make[1]: Entering directory '/usr/src/RPM/BUILD/syscall-1.2/src'
gcov -o .libs basic -t -k -b | grep -v -E '[[:digit:]]:[ /][*]'
<...>
```

Глава 14. Документирование пакета

14.1. Doxygen

При разработке продукта одной из ключевых составляющих является поддержка **информационного пространства**, позволяющего сообществу разработчиков и пользователей ориентироваться в нём и разбираться с доступным функционалом. Информационное пространство состоит как из описания внутренних элементов проекта, так и из информации о возможностях самого продукта и о правилах взаимодействия с ним. При поддержке продукта основную часть информационного пространства заполняет описание и структуризация всех проводящихся изменений проекта.

Написание пользовательской документации — отдельный фронт работ, он более или менее независим от процесса разработки — главное, чтобы она дошла хоть до какой-то тестовой эксплуатации.

А вот *внутренняя* документация, предназначенная для сообщества разработчиков и пользователей разрабатываемого программного инструментария, — это дело самих разработчиков, и тут необходима некоторая дисциплина.

Основная идея: внутренняя документация должна появляться *одновременно* с написанием соответствующих функций, при этом вносить как можно меньше «шума» в исходный текст программы и не мешать работе программиста (который должен её оформлять):

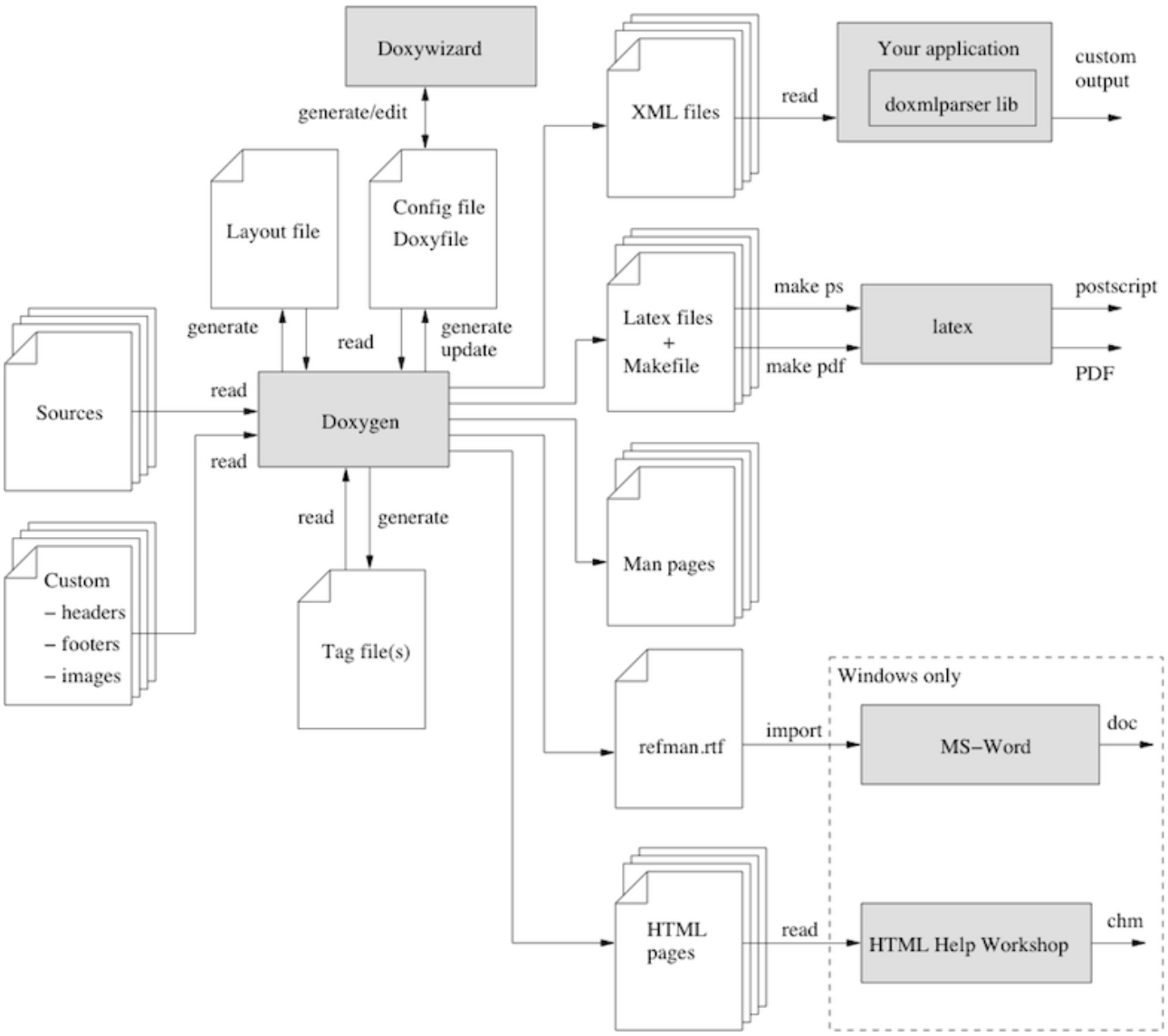
- радикальный вариант: документация пишется *до* реализации функций, которые она документирует. Одним из ярких примеров является методика Literate Programming, предложенная Дональдом Кнутом;
- если язык программирования поддерживает т. н. *самодокументирование* (как, например, docstrings в Python), большая часть технической документации уходит туда. В этом случае полезно разделять «документацию вообще», не привязанную к конкретному тексту функций, внутреннюю документацию в docstrings и «просто комментарии», которые теперь ориентированы на более узкую аудиторию — на разработчика, который собирается понять и изменить сложное место в программе;
- техническое документирование — например, документирование API — обычно оперирует довольно обширным набором понятий (классы, типы их методов и полей, параметры функций и методов, содержимое библиотек и т. п.). Для ведения внутренней документации с давних пор существует целое семейство подсистем со своим синтаксисом и правилами привязки к исходным текстам. Таким подсистемам не обязательна поддержка самодокументирования — достаточно комментариев особого вида.

За пределами внутреннего документирования остаётся всё остальное информационное пространство:

- offline-справочники, в первую очередь man и GNU Texinfo;
- сайты, предназначенные для совместного создания сообществом (wiki или простые CMS), — как встроенные в Git-хостинги, так и в виде отдельных приложений;
- т. н. «сайтогенераторы» — приложения, предназначенные для порождения статического HTML и последующей публикации. Это могут быть специализированные приложения, наподобие Pelican или Jekyll, но можно заставить работать в таком режиме и саму систему технического документирования;
- специализированные сервисы документирования (ReadTheDocs, Git Book).

14.1. Doxygen

Самым популярным на сегодня инструментом документирования является Sphinx; в нашем проекте косвенно задействован другой — Perl Podlators — с его помощью форматируется man-страница. Внутреннее документирование мы организуем с помощью ещё одной классической системы — Doxygen. Процесс создания документации с её помощью поддерживается Autotools, при этом доступно немалое количество выходных форматов документации — от HTML-страницы до печатного варианта:



Добавим в проект поддержку doxygen:

```
.
├── configure.ac
├── doc
│   ├── Makefile.am
│   └── syscall.pod
├── LICENSE
├── Makefile.am
├── src
│   ├── basic.c
│   ├── globals.c
│   ├── lssyscalls
│   ├── Makefile.am
│   ├── syscall.c
│   ├── syscall.h
│   └── utility.c
└── tests
    ├── include.ts
    ├── Makefile.am
    └── upstream.ts
```

Для описания параметров документирования используется базовый шаблонный файл **`Doxyfile.in`**, генерируемый командой:

@**user**

```
[user@VM syscall-master]$ doxygen -g Doxyfile.in
```

Файл представляет собой тщательно откомментированный перечень свойств будущей документации:

@**user**: **`syscall-master/Doxyfile.in`**

```
#---------------------------------------------------------------------------
# Project related configuration options
#---------------------------------------------------------------------------

# This tag specifies the encoding used for all characters in the configuration
# file that follow. The default is UTF-8 which is also the encoding used for all
# text before the first occurrence of this tag. Doxygen uses libiconv (or the
# iconv built into libc) for the transcoding. See
# https://www.gnu.org/software/libiconv/ for the list of possible encodings.
# The default value is: UTF-8.

DOXYFILE_ENCODING      = UTF-8

# The PROJECT_NAME tag is a single word (or a sequence of words surrounded by
# double-quotes, unless you are using Doxywizard) that should identify the
# project for which the documentation is generated. This name is used in the
# title of most generated pages and in a few other places.
# The default value is: My Project.

PROJECT_NAME           = "My Project"

# The PROJECT_NUMBER tag can be used to enter a project or revision number. This
# could be handy for archiving the generated documentation or if some version
# control system is used.

PROJECT_NUMBER         =
```

Поменяем основные параметры:

@**user**: **`syscall-master/Doxyfile.in`**

```
diff --git a/Doxyfile.in b/Doxyfile.in
index a822940..29071c4 100644
--- a/Doxyfile.in
+++ b/Doxyfile.in
@@ -42,13 +42,13 @@ DOXYFILE_ENCODING      = UTF-8
# title of most generated pages and in a few other places.
# The default value is: My Project.

-PROJECT_NAME           = "My Project"
+PROJECT_NAME           = "@PACKAGE_TARNAME@"

# The PROJECT_NUMBER tag can be used to enter a project or revision number. This
# could be handy for archiving the generated documentation or if some version
# control system is used.
```

```
-PROJECT_NUMBER         =
+PROJECT_NUMBER         = @PACKAGE_VERSION@

# Using the PROJECT_BRIEF tag one can provide an optional one line description
# for a project that appears at the top of each page and should give viewer a
@@ -54,7 +54,7 @@ PROJECT_NUMBER         = @PACKAGE_VERSION@
# for a project that appears at the top of each page and should give viewer a
# quick idea about the purpose of the project. Keep the description short.

-PROJECT_BRIEF          =
+PROJECT_BRIEF          = "@PACKAGE_NAME@"

# With the PROJECT_LOGO tag one can specify a logo or an icon that is included
# in the documentation. The maximum height of the logo should not exceed 55
@@ -74,7 +74,7 @@ PROJECT_ICON           =
# entered, it will be relative to the location where doxygen was started. If
# left blank the current directory will be used.

-OUTPUT_DIRECTORY       =
+OUTPUT_DIRECTORY       = @DX_DOCDIR@

# If the CREATE_SUBDIRS tag is set to YES then doxygen will create up to 4096
# sub-directories (in 2 levels) under the output directory of each output format
@@ -215,7 +215,7 @@ SHORT_NAMES            = NO
# description.)
# The default value is: NO.

-JAVADOC_AUTOBRIEF      = NO
+JAVADOC_AUTOBRIEF      = YES

# If the JAVADOC_BANNER tag is set to YES then doxygen will interpret a line
# such as
@@ -297,7 +297,7 @@ ALIASES                =
# members will be omitted, etc.
# The default value is: NO.

-OPTIMIZE_OUTPUT_FOR_C  = NO
+OPTIMIZE_OUTPUT_FOR_C  = C

# Set the OPTIMIZE_OUTPUT_JAVA tag to YES if your project consists of Java or
# Python sources only. Doxygen will then generate output that is more tailored
@@ -524,31 +524,31 @@ TIMESTAMP              = NO
# normally produced when WARNINGS is set to YES.
# The default value is: NO.

-EXTRACT_ALL            = NO
+EXTRACT_ALL            = YES

# If the EXTRACT_PRIVATE tag is set to YES, all private members of a class will
# be included in the documentation.
# The default value is: NO.
 -EXTRACT_PRIVATE        = NO
+EXTRACT_PRIVATE        = YES

# If the EXTRACT_PRIV_VIRTUAL tag is set to YES, documented private virtual
# methods of a class will be included in the documentation.
# The default value is: NO.

-EXTRACT_PRIV_VIRTUAL   = NO
```

```
+EXTRACT_PRIV_VIRTUAL   = YES

# If the EXTRACT_PACKAGE tag is set to YES, all members with package or internal
# scope will be included in the documentation.
# The default value is: NO.

-EXTRACT_PACKAGE        = NO
+EXTRACT_PACKAGE        = YES

# If the EXTRACT_STATIC tag is set to YES, all static members of a file will be
# included in the documentation.
# The default value is: NO.

-EXTRACT_STATIC         = NO
+EXTRACT_STATIC         = YES

# If the EXTRACT_LOCAL_CLASSES tag is set to YES, classes (and structs) defined
# locally in source files will be included in the documentation. If set to NO,
@@ -564,7 +564,7 @@ EXTRACT_LOCAL_CLASSES  = YES
# included.
# The default value is: NO.

-EXTRACT_LOCAL_METHODS  = NO
+EXTRACT_LOCAL_METHODS  = YES

# If this flag is set to YES, the members of anonymous namespaces will be
# extracted and appear in the documentation as a namespace called
@@ -949,7 +949,7 @@ WARN_LOGFILE           =
# spaces. See also FILE_PATTERNS and EXTENSION_MAPPING
# Note: If this tag is empty the current directory is searched.

-INPUT                  =
+INPUT                  = @top_srcdir@/src

# This tag can be used to specify the character encoding of the source files
# that doxygen parses. Internally doxygen uses the UTF-8 encoding. Doxygen uses
```

Самодокументирование представляет собой комментирование исходников, которое и уйдёт в документацию. Doxygen поддерживает как однострочные, так и многострочные комментарии, при чём вне зависимости от формата документирования в разных языках программирования. Ключевые слова, отмеченные символом ***@*** позволяют описывать в документации основные компоненты программы, а также определять разделы (как, например, ключевое слово ***@mainpage*** для размещения текста на главной странице документации):

@**user**: **syscall-master/src/syscall.c**

```
/** @mainpage Syscall
* syscall - send system calls from your shell
*
* Execute a list of raw system calls. All the system calls listed in your
system's unistd.h are
* supported, with up to 5 arguments. A maximum of 20 calls can be executed per
invocation, each
* separated by a comma.
*
* Arguments starting by a "#" symbol are used to give a string length. For
instance, "#hello"
* would be evaluated as 5.
```

```
*
* Arguments starting by a "$" followed by a number from 0 to 19 refer to a
previous system call
* return code. For instance, $0 refers to to the return code of the first system
call executed.
* To display those values, use the "echo" built-in command.
*
* The "echo" command can be used like any other system call to easily display "$"
or "#" values,
* or any string or number.
*
* @param -<n> execute the given commands n times, where n is an integer between 0
and "INT_MAX"
* @param -h/--help return program usage
* @param -v/--version return version of program
*
* @return 0 if all syscalls were successful, 1 on error. Note that if any system
returns -1,
* the program will exit immediately after printing the associated error message.
*/

#include "syscall.h"
/** Parse input shell-string and execute syscalls
*
* @return 0 if string parsing and syscall's execution were successfully
completed, 1 if weren't
*/
int main(int argc, char **argv)
{
 int repeat = 1, skip = 1;

 if (argc <= 1) {
   usage();
   return 0;
 }

 /* arguments */
 if (argv[1][0] == '-') {
   /* help */
   if (streq(argv[1], "-h") || streq(argv[1], "--help")) {
     usage();
     return 0;
   }

   /* version */
   if (streq(argv[1], "-v") || streq(argv[1], "--version")) {
     puts("syscall version "VERSION);
     return 0;
   }

   /* Handle -n option */
   repeat = atoi(argv[1] + 1);
   if (repeat < 1) {
     errx(1, "option -<n> must be between 1 and %d", INT_MAX);
   }
   if (argc <= 3) {
     usage();
     return 1;
   }
```

```
    skip++;
  }

  memset(ret_values, -1, sizeof ret_values);

  while (repeat--) {
    split_cmdline(argc - skip, argv + skip);
  }

  return 0;
}
```

@user: **syscall-master/src/utility.c**

```
/** @page utility
* There are special functions for internal workings of the program
*/

#include "syscall.h"
/** Help-string of program usage
*/
void usage()
{
 puts("usage: syscall [-<n>] name [args...] [, name [args...]]...");
}

/** Syscall`s names comparator
*
* @param m1 first comparing syscall
* @param m2 second comparing syscall
*
* @return comparing result
*/
int scomp(const void *m1, const void *m2)
{
 Syscall *sys1 = (Syscall *)m1;
 Syscall *sys2 = (Syscall *)m2;
 return strcmp(sys1->name, sys2->name);
}

/** Binary search of executing syscall @p name
*
* @param name executing syscall
* @return system code of syscall
*/
long lookup(const char *name)
{
 Syscall key, *res;
 key.name = name;

 res = bsearch(&key, systab, systab_size, sizeof key, scomp);
 if (res == NULL) {
   errx(1, "unknown system call: %s", name);
 }

 return res->code;
}

/** Quick and dirty way to unescape \n at the end of strings
```

```
*/
void unescape_nl(char *str) {
 size_t end = strlen(str) - 1;

 while (str[end] == 'n' && str[end-1] == '\\') {
   str[end-1] = '\n';
   str[end] = '\0';
   end -= 2;
 }
}

/** Special debugging function
*/
void dump_ret_values(void) {
 for (int i = 0; i < CMD_MAX; i++) {
   printf("%0d  %d\n", i, ret_values[i]);
 }
}
```

В **`configure.ac`** опишем использование Doxygen, для этого необходимо лишь инициализировать его (при этом указываются имя проекта, путь к файлу настройки и путь к директории, где будет храниться документация) и указать в списке генератов:

@**user**: **`syscall-master/configure.ac`**

```
#                                               -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.


AC_INIT([syscall in Shell], [1.0], [UsamG1t], [syscall])
AC_CONFIG_SRCDIR([src/syscall.c])

AM_INIT_AUTOMAKE([foreign subdir-objects])
LT_INIT([disable-static])
AC_CONFIG_HEADERS([config.h])

DX_INIT_DOXYGEN([syscall], [Doxyfile], [doxygen-doc])

# Checks for programs.
<...>
AC_CONFIG_FILES([Makefile src/Makefile doc/Makefile tests/Makefile Doxyfile])
AC_OUTPUT
```

В spec-файле все необходимые для документации пакеты должны быть занесены в BuildRequires:

- непосредственно, doxygen;
- graphviz для демонстрации графиков зависимостей файлов;
- autoconf-archive для загрузки файлов с doxygen-макросами для их работы.

Для установки документации при установке пакета в директиву **`%files`** необходимо указать зависимость на данные в директории документации:

@**user**: **`syscall-master/.gear/syscall.spec`**

```
<...>
# Automatically added by buildreq on Fri Aug 08 2025
# optimized out: glibc-kernheaders-generic glibc-kernheaders-x86 gnu-config
libgpg-error perl perl-Encode perl-Pod-Escapes perl-Pod-Simple perl-parent perl-
podlators sh5
BuildRequires: perl-Pod-Usage check libcheck-devel doxygen graphviz autoconf-
archive

<...>

%files
%_bindir/%name
%_libdir/*
%_docdir/*
%_man1dir/*

%changelog
* Tue Aug 12 2025 UsamG1t <usamg1t@altlinux.org> 1.3-alt1
- Add Doxygen

* Mon Aug 11 2025 UsamG1t <usamg1t@altlinux.org> 1.2-alt1
- Add gcov

* Fri Aug 08 2025 UsamG1t <usamg1t@altlinux.org> 1.1-alt1
- Add xUnit check

* Fri Aug 08 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial Build
```

Итоговый вид директорий проекта с документацией выглядит так:

```
.
├── configure.ac
├── doc
│   ├── Makefile.am
│   └── syscall.pod
├── Doxyfile.in
├── LICENSE
├── Makefile.am
├── src
│   ├── basic.c
│   ├── globals.c
│   ├── lssyscalls
│   ├── Makefile.am
│   ├── syscall.c
│   ├── syscall.h
│   └── utility.c
└── tests
    ├── include.ts
    ├── Makefile.am
    └── upstream.ts
```

@user

```
[user@VM syscall-master]$ gear-hsh --lazy
<...>
Wrote: /usr/src/in/srpm/syscall-1.3-alt1.src.rpm (w1.gzdio)
Installing syscall-1.3-alt1.src.rpm
<...>
```

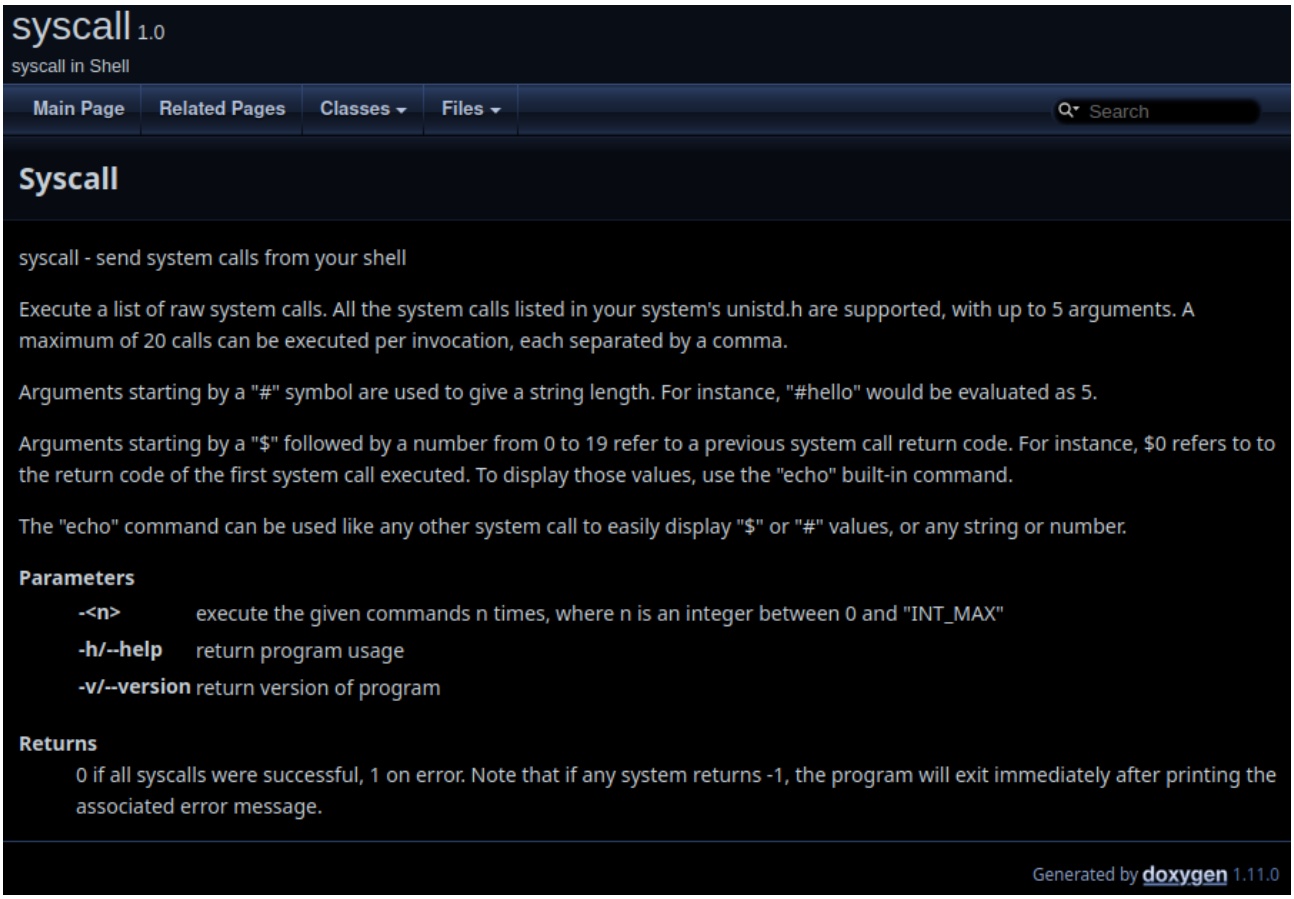
```
SRCDIR='.' PROJECT='syscall' VERSION='1.0' PERL_PATH='/usr/bin/perl'
HAVE_DOT='NO' GENERATE_MAN='NO' GENERATE_RTF='NO' GENERATE_XML='NO'
GENERATE_HTMLHELP='NO' GENERATE_CHI='NO' GENERATE_HTML='YES' GENERATE_LAT
EX='NO' DOCDIR=doxygen-doc /usr/bin/doxygen Doxyfile
<...>
finished...
echo Timestamp >doxygen-doc/syscall.tag
```

```
<...>
Wrote: /usr/src/RPM/SRPMS/syscall-1.3-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/syscall-1.3-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/syscall-debuginfo-1.3-alt1.x86_64.rpm (w2.lzdio)
```

@rooter

```
[user@VM syscall-master]$ hsh-shell --rooter
[root@localhost .in]# rpm -i syscall-1.3-alt1.x86_64.rpm
<13>Aug 12 11:39:55 rpm: syscall-1.3-alt1 1754998745 installed

[root@localhost .in]# ls /usr/share/doc/syscall/html/
annotated.html                          doc.svg
functions_vars.html   graph_legend.png  navtree.css
syscall_8c.html            systab_8h.html            tab_sd.png
basic_8c.html                           docd.svg
globals.html          index.html        open.png
syscall_8c__incl.map       systab_8h__dep__incl.map  tabs.css
basic_8c__incl.map                      doxygen.css
globals_8c.html       jquery.js         plus.svg
syscall_8c__incl.md5       systab_8h__dep__incl.md5  utility_8c.html
basic_8c__incl.md5                      doxygen.svg
globals_8c__incl.map  menu.js           plusd.svg
syscall_8c__incl.png       systab_8h__dep__incl.png  utility_8c__incl.map
basic_8c__incl.png                      doxygen_crawl.html
globals_8c__incl.md5  menudata.js       resize.js
syscall_8h.html            systab_8h_source.html     utility_8c__incl.md5
bc_s.png                                dynsections.js
globals_8c__incl.png  minus.svg         search
syscall_8h__dep__incl.map  tab_a.png                utility_8c__incl.png
bc_sd.png                               files.html
globals_defs.html     minusd.svg        splitbar.png
syscall_8h__dep__incl.md5  tab_ad.png
classes.html                            folderclosed.svg
globals_func.html     nav_f.png         splitbard.png
syscall_8h__dep__incl.png  tab_b.png
clipboard.js                            folderclosedd.svg
globals_type.html     nav_fd.png        structsyscall-members.html
syscall_8h__incl.map       tab_bd.png
closed.png                              folderopen.svg
globals_vars.html     nav_g.png         structsyscall.html
syscall_8h__incl.md5       tab_h.png
cookie.js                               folderopend.svg
```

```
graph_legend.html     nav_h.png         sync_off.png
syscall_8h__incl.png       tab_hd.png
dir_68267d1309a1af8e8297ef4c3efbcdba.html  functions.html
graph_legend.md5      nav_hd.png        sync_on.png
syscall_8h_source.html     tab_s.png

[root@localhost .in]#
```



С помощью Doxygen возможна генерация практически любого формата документации. Для создания дополнительных справочников, например, man, необходимо указать это в настройщике, а также добавить правила генерации в **`Makefile.am`**:

@**user**: **`syscall-master/Doxyfile.in`**

```
diff --git a/Doxyfile.in b/Doxyfile.in
index d097c79..aa9f764 100644
--- a/Doxyfile.in
+++ b/Doxyfile.in
@@ -2204,7 +2204,7 @@ RTF_EXTRA_FILES        =
# classes and files.
# The default value is: NO.

-GENERATE_MAN           = NO
+GENERATE_MAN           = YES

# The MAN_OUTPUT tag is used to specify where the man pages will be put. If a
# relative path is entered the value of OUTPUT_DIRECTORY will be put in front of
lines 1-13/13 (END)
```

Поскольку Autotools умеет генерировать рецепты для сборки man, необходимо лишь указать путь, где должна будет храниться страница с документацией:

@**user**: **syscall-master/Makefile.am**

```
SUBDIRS = src doc tests

@DX_RULES@

all-local:      doxygen-doc

doxygen-doc/man/man3/syscall.h.3: doxygen-doc

man3_MANS = doxygen-doc/man/man3/syscall.h.3

checklog:       check
       cat tests/*.log

gcov:   check
       $(MAKE) -C src gcov

http:   doxygen-doc
       python3 -m http.server --directory $</html
```

```
...<Building PKG>...
```

@**rooter**

```
[user@VM syscall-master]$ hsh-shell --rooter
[root@localhost .in]# rpm -i syscall-1.3-alt1.x86_64.rpm
<13>Aug 12 12:09:43 rpm: syscall-1.3-alt1 1754998745 installed

[root@localhost .in]# ls /usr/share/man/man1/syscall.1.xz
/usr/share/man/man1/syscall.1.xz
[root@localhost .in]# ls /usr/share/man/man3/syscall.h.3.xz
/usr/share/man/man3/syscall.h.3.xz
[root@localhost .in]#
```

/usr/share/man/man3/syscall.h.3.xz

```
src/syscall.h(3)                                          Library Functions
Manual                                          src/syscall.h(3)

NAME
      src/syscall.h

SYNOPSIS
      #include <err.h>
      #include <errno.h>
      #include <limits.h>
      #include <stdio.h>
      #include <stdlib.h>
      #include <string.h>
      #include <unistd.h>

  Classes
      struct syscall
```

```
  Macros
      #define _DEFAULT_SOURCE
      #define VERSION   'unknown'
      #define streq(a,  b)   (strcmp(a,b) == 0)
      #define ARG(n)   parse_arg(syscall_name, cmd[n])
      #define CMD_MAX   20  /* maximum number of commands per invocation */

  Typedefs
      typedef struct syscall Syscall

  Functions
      void usage ()
          Help-string of program usage.
      int scomp (const void *m1, const void *m2)
          Syscall`s names comparator.
      long lookup (const char *name)
          Binary search of executing syscall name.
      void unescape_nl (char *str)
          Quick and dirty way to unescape
           at the end of strings. "
      void dump_ret_values (void)
          Special debugging function.
      void echo (int argc, char **argv)
          Info syscall for input data presentation.
      unsigned long parse_arg (const char *syscall_name, char *arg)
          Argument Parser.
      void parse_syscall (int cmd_no, char **cmd, int cmd_len)
          Search and Execute syscall with fix number of args.
      void split_cmdline (int argc, char **argv)
          Parse input to groups <<syscall + fix number of args>> and execute
syscalls.

  Variables
      int ret_values [CMD_MAX]
      Syscall systab []
      int systab_size

Macro Definition Documentation
  #define _DEFAULT_SOURCE
  #define ARG( n)   parse_arg(syscall_name, cmd[n])
  #define CMD_MAX   20  /* maximum number of commands per invocation */
  #define streq( a,  b)   (strcmp(a,b) == 0)
  #define VERSION   'unknown'
Typedef Documentation
  typedef struct syscall Syscall
Function Documentation
  void dump_ret_values (void )
      Special debugging function.

  void echo (int argc, char ** argv)
      Info syscall for input data presentation.

      Returns
          input shell-string with parsed arguments

  long lookup (const char * name)
      Binary search of executing syscall name.

      Parameters
```

```
          name executing syscall

      Returns
          system code of syscall

  unsigned long parse_arg (const char * syscall_name, char * arg)
      Argument Parser.

      Parameters
          syscall_name syscall to which the argument belongs
          arg argument which need to be parsed:

          o #<word> for length of a string

          o $<number> for return values of a previous syscalls

          o <number> for a number

          o <word> for a string

      Returns
          parsing value depending on input

  void parse_syscall (int cmd_no, char ** cmd, int cmd_len)
      Search and Execute syscall with fix number of args.

      Parameters
          cmd_no number of syscall
          cmd syscall text pointer
          cmd_len length of arguments of syscall

  int scomp (const void * m1, const void * m2)
      Syscall`s names comparator.

      Parameters
          m1 first comparing syscall
          m2 second comparing syscall

      Returns
          comparing result

  void split_cmdline (int argc, char ** argv)
      Parse input to groups <<syscall + fix number of args>> and execute
syscalls.

      Parameters
          argc length of shell-string (words)
          argv input shell-string

  void unescape_nl (char * str)
      Quick and dirty way to unescape
       at the end of strings.

  void usage ()
      Help-string of program usage.

Variable Documentation
  int ret_values[CMD_MAX] [extern]
  Syscall systab[] [extern]
```

```
  int systab_size [extern]
Author
      Generated automatically by Doxygen for syscall from the source code.

syscall                                          Version
1.0                                          src/syscall.h(3)
```

Глава 15. Лицензирование и множественная сборка пакетов

15.1. Лицензирование

Лицензирование продукта — ещё один важный этап разработки. Каждый проект имеет некоторые цели в его использовании, распространении, доступности и т. д. И для реализации этих целей необходим некоторый инструмент, ограничивающий (или не ограничивающий) пользователей ПО. Лицензии глобально можно разделить на проприетарные и свободные.

Проприетарные лицензии эксплуатируют свойство безущербного *копирования* информации: для того, чтобы из одного *экземпляра* программного продукта сделать два, никаких дополнительных затрат не требуется, в отличие от материальных объектов, на копирование которых требуются расходные материалы и созидательный труд рабочего (особенно в отсутствие автоматизации). Как следствие, бизнес на основе проприетарных лицензий на ПО стремится максимально монетизировать именно факт *копирования экземпляра*, так так он в идеале приносит бесконечный процент прибыли. При этом, как правило:

- *распространение* ПО без санкций правообладателя запрещается полностью;
- количество одновременно работающих копий продукта фиксировано. Грубо говоря, каждая работающая копия продукта требует *покупки ещё одной лицензии*, и в эту новую покупку не входит вообще никакого ПО и другой информации, кроме увеличения числа лицензий у потребителя.

Бизнес на основе торговли копиями также влечёт за собой особенности *разработки* ПО — она становится *закрытой*. Дабы пресечь непрямое несанкционированное распространение, полностью запрещается доступ к исходным текстам программ, это, в свою очередь, накладывает ограничения на состав и мобильность команды разработчиков и т. д. Основные принципы такого бизнеса, а также некоторые (отчасти сомнительные) моральные его аспекты изложены Биллом Гейтсом в его знаменитом «Открытом письме любителям».

Свободные лицензии, напротив, используют факт безущербного копирования информации, чтобы как можно шире распространять программные продукты и привлекать к их разработке как можно больший круг участников. Это подразумевает *открытую* модель разработки ПО, в которой (в идеале) исходные тексты программ не только не закрываются, но и лежат на самом видном месте, да ещё и сопровождаются толковой внутренней документацией, дисциплиной оформления, принятой в сообществе, открытым диалогом в соответствующих информационных каналах и т. п.

Бизнес на основе свободного ПО технически отличается от проприетарного только отсутствием монетизации копирования. Однако это влечёт за собой изменение *целей*: такой бизнес в первую очередь старается спровоцировать пользователей *принять участие в разработке* ПО — протестировать, исправить недочёт, дописать недостающую функциональность, задокументировать, сделать перевод и т. п. Для этого пользователю нужно дать чёткие права по отношению к самому продукту и его исходным текстам, таким образом любой пользователь свободного ПО автоматически становится *партнёром* основного разработчика.

Одним из главных идеологов свободного ПО является Ричард Столлман. Ему принадлежит авторство одной из самых известных свободных лицензий — GPL, а также самое, пожалуй, прозрачное и непротиворечивое определение свободного ПО (в том или ином виде вошедшее в различные законодательные практики).

По Столлману свободная лицензия предоставляет пользователю всего четыре права:

1. **Право запускать программу для любых целей** — пользователи могут запускать программу на любом оборудовании и для любых задач;
2. **Право изучать и изменять исходный код** — пользователи имеют доступ к исходному коду программы и могут изменять его, чтобы адаптировать ПО под свои нужды;
3. **Право распространять копии программы** — пользователи могут свободно распространять копии программы как в исходном виде, так и в измененном; как бесплатно, так и за плату;
4. **Право распространять измененные версии программы** — пользователи могут публиковать измененные версии программы без каких-либо санкций от разработчика.

Данные свободы гарантируют полный контроль над ПО при его использовании и преобразовании. Различные аспекты местного законодательства (например, неотторжимое авторство или «право на имя») имеют более высокий приоритет, но не конфликтуют с этим определением.

Большой класс свободных лицензий — т. н. «разрешительные» (или *пермиссивные*) лицензии воспроизводят эти четыре права с различными модификациями. Наиболее популярные разрешительные лицензии — BSD, MIT и Creative Commons в варианте «Attribution» (CC-BY).

Бизнес-применение разрешительных лицензий имеет важную особенность: пользователь имеет право *изменять политику лицензирования*. Практически все производители проприетарного ПО используют свободные компоненты — у них есть право распространять производный продукт под закрытой лицензией. К сожалению, культура закрытой разработки по многим причинам не позволяет оставлять получившиеся компоненты (и, что более важно, их эффективные модификации) открытыми.

Стоит заметить, что в современном мире постоянно увеличивается количество ПО и его компонентов под разрешительными лицензиями, причём изрядную долю этого роста обеспечивают именно крупные корпорации с проприетарной моделью разработки — Google, Microsoft, Facebook и т. п. Разрешительные лицензии играют здесь роль катализатора для *законного обмена технологиями*. Скажем, в Facebook разработана высокоэффективная библиотека кеширования ссылок (чем бы она ни являлась ☺). Корпорация заинтересована в *распространении* этой библиотеки (выгода очевидна: стороннее тестирование, в том числе в принципиально других условиях, доработка, кроссплатформенная совместимость и т. д.), а бизнеса на её продаже не планирует. Если наложить на такой продукт проприетарную лицензию, часть предполагаемых пользователей от неё откажутся из-за высокой стоимости владения, другая часть — из-за лавинообразного роста обязательств в стиле «кто кому сколько за что должен». Выход — минималистичная свободная лицензия.

Запрет на изменение политики лицензирования никак не влияет на свойство самого исходного текста ПО, но ограничивает пользователя в плане выбора лицензии, под которой он будет распространять производный продукт. Такое добавление к свободной лицензии носит название «копилефт», и его автор — тот же Ричард Столлман.

«Копилефтные» лицензии содержат пятое правило: «В лицензии производного ПО в обязательном порядке требуется наличие, как минимум, описанных пяти пунктов текущей лицензии». При таких условиях пользователь встаёт перед выбором: либо не распространять обновлённое ПО, либо как минимум по запросу предоставлять исходники.

Наиболее популярно семейство копилефтных лицензий GPL с вариациями, а также Creative Commons в варианте «Sharing Requirements» (CC-BY-SA).

Стоит заметить, что «некоммерческие лицензии», как, например, Creative Commons с суффиксом NC, не являются свободными, так как противоречат двум последним пунктам.

Выбор лицензии зависит от задач и целей проекта. Прочитать о том, как правильно выбрать лицензию для своего продукта, можно по рекомендациям GitHub; можно ещё здесь и здесь. Изучать лицензии и их формальные свойства подробнее стоит на сайте «System Package Data Exchange». Также для выбора лицензии существуют специальные сервисы (choosealicense, Creative Commons).

15.2. Множественная сборка пакетов

При работе с проектами со свободными лицензиями нередко встречаются случаи, когда из одного и того же набора исходных текстов необходимо собирать несколько пакетов. Частым примером является сборка библиотек, а именно обычной их версии и devel-версии, и если это необходимо — статической версии. Ведение отдельных репозиториев для сборки разных пакетов в таком случае — нерациональное решение. Важным случаем является как раз проблема лицензирования: в проекте могут присутствовать материалы разных лицензий (например, для исходных текстов и для документации). По правилам ALT Linux Team допустима сборка пакета со множеством лицензий для исходников при условии упоминания всех лицензий в соответствующей директиве spec-файла. Однако лицензии могут быть даже *несовместимыми* — например, программная часть компьютерной игры может распространяться под GPL, а контент к ней (изображения, музыка) — под лицензией, *запрещающей модификацию*. Строго говоря, такая лицензия не свободна, но допускает свободное распространение. Кстати, в этом варианте есть и ещё один повод разделить пакеты — программная часть будет архитектурно зависимой (попадёт в раздел репозитория **x86_64**), а часть с мультимедиа-контентом — архитектурно независимой (попадёт в раздел **noarch**).

Рассмотрим небольшую программу, выводящую картинку на экран, в котором на основе одних исходников соберём два пакета: один с исполняемым файлом, второй — с картинкой.

Исходный текст программы задействует библиотеку GTK для представления картинки, задаваемой в специальном формате растрового изображения XPM:

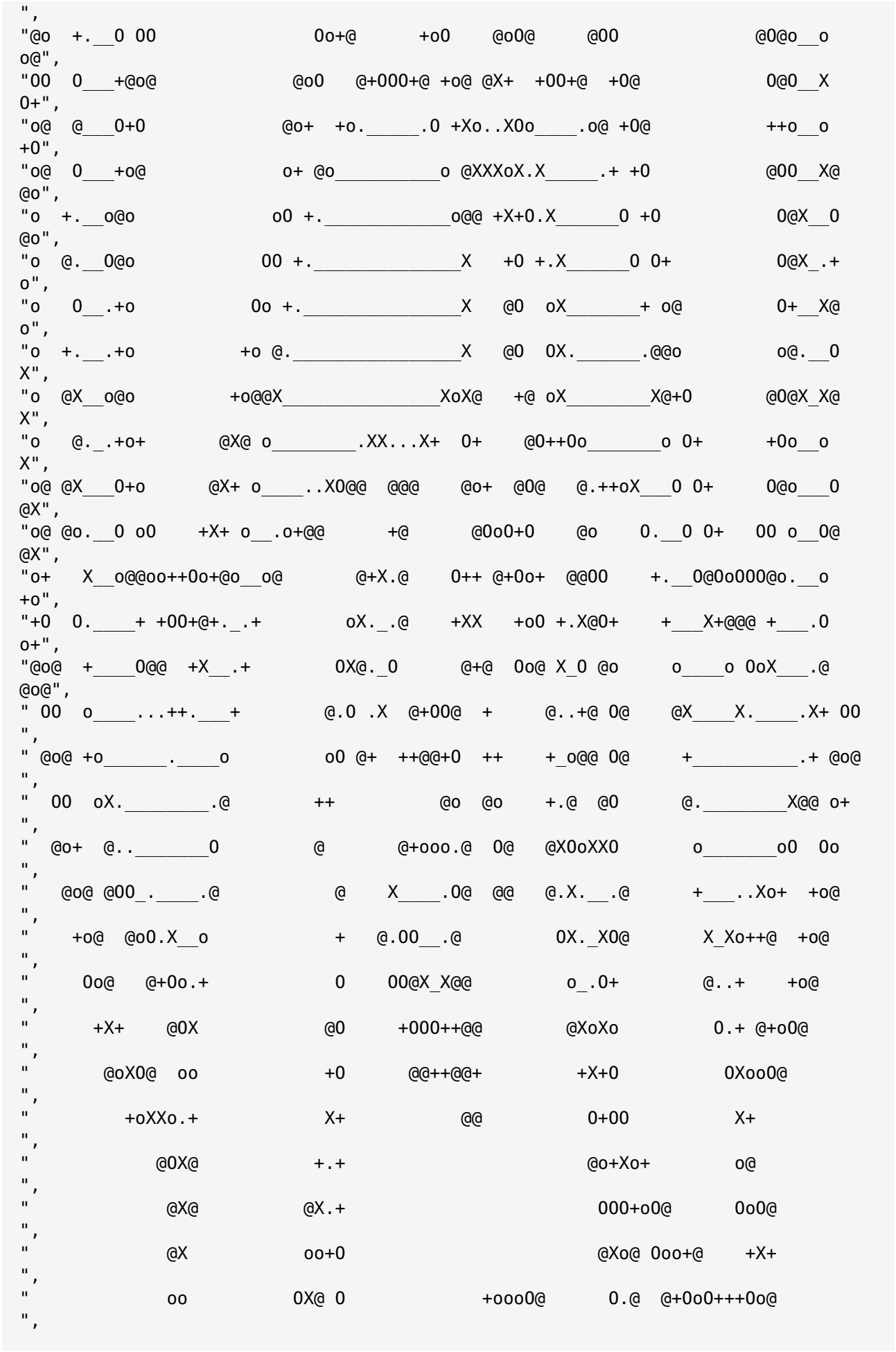
@user: **`GNU_picture/gtk.c`**

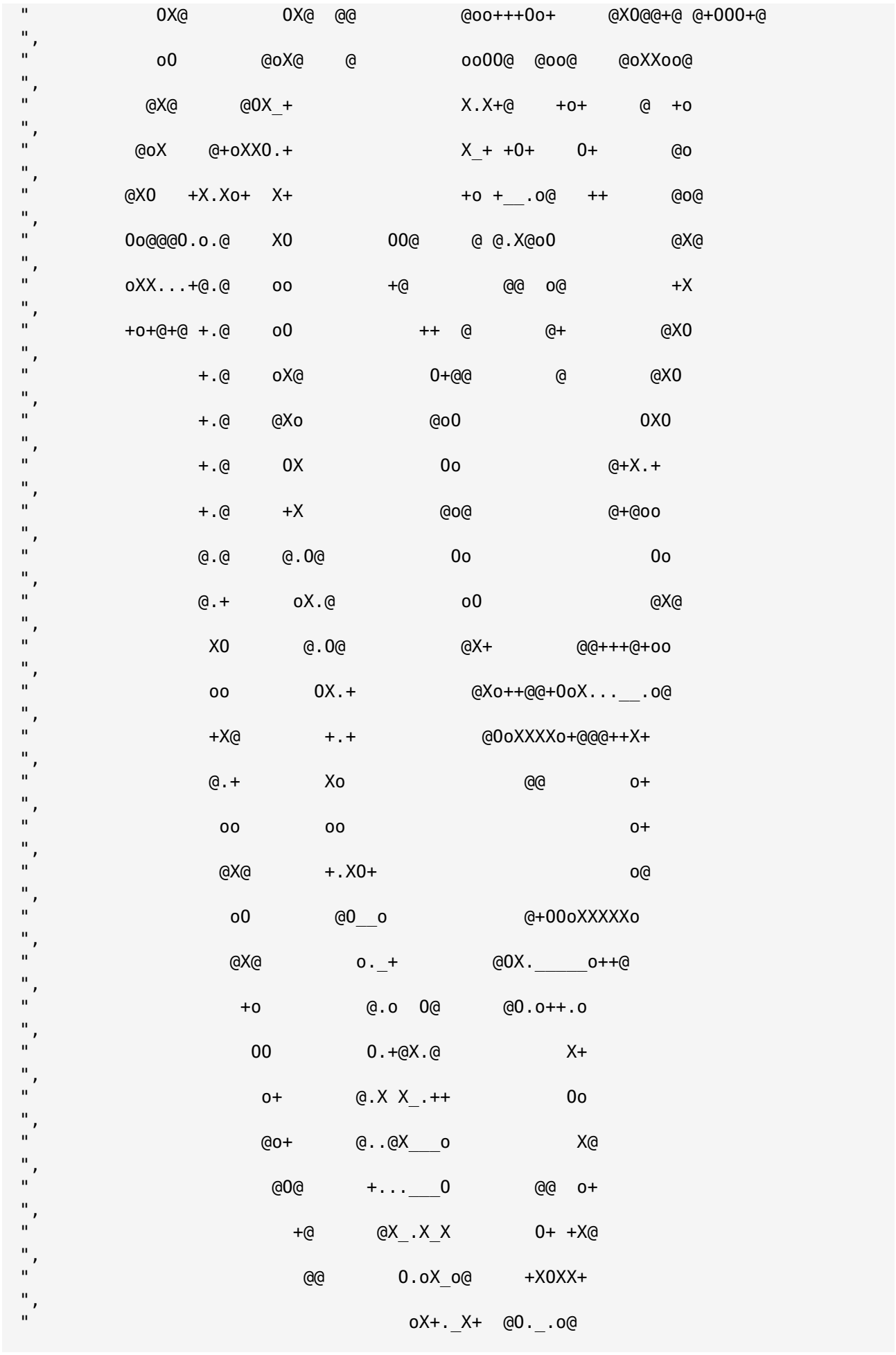
```
#include <gtk/gtk.h>

#ifndef FILEPATH
#define FILEPATH "./GNU.xpm"
#endif

int main(int argc, char *argv[]) {
```

```
    GtkWidget *window;
    GtkWidget *image;
    GError *error = NULL;

    gtk_init(&argc, &argv);

    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "GTK Logo");
    gtk_window_set_default_size(GTK_WINDOW(window), 72, 72);
    g_signal_connect(window, "destroy", G_CALLBACK(gtk_main_quit), NULL);

    image = gtk_image_new_from_file(FILEPATH);
    GdkPixbuf* pixbuf = gtk_image_get_pixbuf(GTK_IMAGE(image));
    pixbuf = gdk_pixbuf_scale_simple(pixbuf, 80*4, 78*4, GDK_INTERP_TILES);
    gtk_image_set_from_pixbuf(GTK_IMAGE(image), pixbuf);
    gtk_container_add(GTK_CONTAINER(window), image);

    gtk_widget_show_all(window);
    gtk_main();

    return 0;
}
```

@**user**: **GNU_picture/GNU.xpm**

```
/* XPM */
static char *GNU[] = {
/* columns rows colors chars-per-pixel */
"80 78 8 1 ",
"  c none",
"_ c black",
". c #242424",
"X c #494949",
"o c #6D6D6D",
"O c #929292",
"+ c #B6B6B6",
"@ c #DBDBDB",
/* pixels */
"            +OoXXXXXXXXX0@                                  @+OOOOOO+@
",
"         @Ooo+@@    @@@@OoO@                                @OoO++++++OoO+@
",
"        OoO@   +ooOOX.XoO++oO                               @Oo+@@+O@ @@  @+oO
",
"      +oo@ @ooX_____XX_XoXO@OoOOo                         @@+o+@oX._.....o++@ Oo@
",
"     OX+   o___.+++@ @@   @@ @+Oo                        @XoO@ O+@OO@OooX___o  +X+
",
"    OX@ +X._.oo+ @Ooooooo0+@@@+Oo@                        OO@    @@++++@@X.._oO@@X+
",
"   +X@ @.___O  +ooO+@@@+Ooooo0+@                          +OOOOooO+++OoO@@+___o @X@
",
"  @X@ @X__.X@@oo@                                         @@        +o+ X___+ +o@
",
"  oO @.__.@ +o+                                                      OO@+.__O OO
",
" @o  @___X @o@                  @@@@@            @@@@                 OO X__o  o@
",
" o+  O___O o@                @OooOOOooO@       +OOOOOO+@               o@O__X  +O
```





```
",
"                                    +.X O__X +.___.@
",
"                                   O.o@ @._X@._____O
",
"                                   oO   o__.+._____O@@
",
"                                   @+  @.X__X._______.o++
",
"                                    @  +o@._____________.o@
",
"                                        O@O_____._____o@
",
"                                         @ O_.X_.OX__X+@
",
"                                            +XOOoO++OO+
"
};
```

@**user**: **GNU_picture/.gear/rules**

```
spec: .gear/GNU_picture.spec
copy: .
```

Для сборки двух или более пакетов из одного комплекта исходников в spec-файле необходимо указать 3 дополнительных директивы:

- директива ***%package <suffix>*** описывает основные метаданные пакета, название которого соответствует формату ***%name-suffix***. В рамках данной директивы обязательно должны быть описаны директивы Summary, License и Group. Сюда же прописываются Requires и BuildArch при их необходимости;
- директива ***%description <suffix>*** даёт описание создаваемого дополнительного пакета;
- директива ***%files <suffix>*** описывает входящие в дополнительный пакет файлы. Если лицензии или архитектуры целевых пакетов различаются, следует использовать соответствующие директивы вдобавок к этим трём.

В нашем случае для дополнительного пакета с картинкой указываются, как другая лицензия, так и информация об архитектурной независимости компонента (***BuildArch: noarch***). Также для пакета с программой добавлена зависимость на пакет с данными (поскольку программа без картинки картинку не откроет☹):

@**user**: **GNU_picture/.gear/GNU_picture.spec**

```
Name: GNU_picture
Version: 1.0
Release: alt1

Summary: Print your special GNU picture
License: GPL-3.0-or-later
Group: Development/Other
Requires: GNU_picture-data

# Automatically added by buildreq on Thu Aug 18 2025
# optimized out: at-spi2-atk glib2-devel glibc-kernheaders-generic glibc-
kernheaders-x86 libat-spi2-core libatk-devel libcairo-devel
# libcairo-gobject libcairo-gobject-devel libgdk-pixbuf libgdk-pixbuf-devel
```

```
libgio-devel libgpg-error libharfbuzz-devel libpango-devel
# libwayland-client libwayland-cursor libwayland-egl pkg-config python3 python3-
base sh5 zlib-devel
BuildRequires: libgtk+3-devel

%package data
Summary: Your special GNU picture
License: GFDL-1.3-or-later
Group: Other
BuildArch: noarch

Source0: gtk.c
Source1: GNU.xpm

%description
This is GTK App using for show of a GNU picture

%description data
This is just a GNU picture

%build
gcc %optflags %SOURCE0 -o %name `pkg-config --cflags --libs gtk+-3.0`
-DFILEPATH=\"%_datadir/GNU_picture/GNU.xpm\"

%install
install -D %name %buildroot%_bindir/%name
install -D %SOURCE1 %buildroot%_datadir/%name/GNU.xpm

%files
%_bindir/%name

%files data
%_datadir/*

%changelog
* Mon Aug 18 2025 UsamG1t <usamg1t@altlinux.org> 1.0-alt1
- Initial Build
```

При сборке из одного spec-файла собираются один пакет с исходным кодом и несколько установочных пакетов согласно описанному именованию:

@user

```
[user@VM GNU_picture]$ gear-hsh --lazy
<...>
Wrote: /usr/src/RPM/SRPMS/GNU_picture-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/GNU_picture-1.0-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/GNU_picture-data-1.0-alt1.x86_64.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/GNU_picture-debuginfo-1.0-alt1.x86_64.rpm
(w2.lzdio)
2.46user 2.38system 0:10.46elapsed 46%CPU (0avgtext+0avgdata 63832maxresident)k
16840inputs+976outputs (48major+223513minor)pagefaults 0swaps
[user@VM GNU_picture]$ cp ~/hasher/repo/x86_64/RPMS.hasher/GNU_picture-data-1.0-
alt1.x86_64.rpm ~/hasher/chroot/.in/
[user@VM GNU_picture]$ cp ~/hasher/repo/x86_64/RPMS.hasher/GNU_picture-1.0-
alt1.x86_64.rpm ~/hasher/chroot/.in/
```

@rooter

```
[root@localhost .in]# rpm -i GNU_picture-1.0-alt1.x86_64.rpm
error: Failed dependencies:
       GNU_picture-data = 1.0-alt1 is needed by GNU_picture-1.0-alt1.x86_64

[root@localhost .in]# rpm -i GNU_picture-data-1.0-alt1.x86_64.rpm
<13>Aug 22 07:11:37 rpm: GNU_picture-data-1.0-alt1 1755846621 installed

[root@localhost .in]# rpm -i GNU_picture-1.0-alt1.x86_64.rpm
<13>Aug 22 07:11:43 rpm: GNU_picture-1.0-alt1 1755846621 installed

[root@localhost .in]# which GNU_picture
/usr/bin/GNU_picture
[root@localhost .in]# ls /usr/share/GNU_picture/
GNU.xpm
[root@localhost .in]# GNU_picture
```

Глава 16. Установка проекта

Итоговый продукт любой разработки почти всегда представляет из себя целую систему связанных элементов, и не всегда удобно или возможно создавать на его основе пакет. Для распространения и использования итогового продукта необходимо предусмотреть возможность «работы из коробки» исполняемых файлов и сопутствующих данных (в частности, доступ к документации). Несомненно, существует вариант распространения «чистых» исходников (через публичные репозитории, например), однако такой вариант требует от пользователя знаний и умений для донастройки (если не перенастройки) всех связей при ручной сборке и установке. Кроме этого могут возникнуть проблемы, связанные с недостатком библиотек или необходимых для сборки утилит.

16.1. Варианты установки

Прежде всего заметим, что наиболее надёжный вариант установки ПО в систему — это *пакет*, собранный в эту систему сообразно дисциплине сообщества. Использование Hasher и Gear делает результат RPM-пакета ещё более совместимым. Главный недостаток пакета, созданного по всем правилам — его необходимо *сопровождать*, то есть пересматривать работоспособность с каждым обновлением, затрагивающим его зависимости, пересобирать и исправлять в случае чего и т. д.

Какими неприятностями грозят «классические» варианты установки свежесобранного ПО по алгоритму «крибле! крабле! бумс!», то есть **`./configure`**, **`make`**, **`make install`**?

Установка в соответствии со *стандартом иерархии файлов* в файловой системе (Filesystem_Hierarchy_Standard), которую мы использовали для сборки пакета, предполагает, что исполняемые файлы устанавливаются в системный каталог **`/usr/bin`**, библиотеки — в **`/usr/lib64`**, документация — в **`/usr/share/doc`** и т. д. Установка в системные директории согласно FHS гарантирует работу устанавливаемого ПО, поскольку работа со всеми системными утилитами настроена под такое распределение файлов. С точки зрения разработчика у такого способа есть важный недостаток: размещение файлов в системных каталогах — это вмешательство в архитектуру ОС. Оно требует прав суперпользователя (даже если задача — просто потестировать очередной вариант сборки) и возлагает на программиста несвойственную ему ответственность за возможные файловые конфликты. Например, назвал программист свою программу «kitten» (котёнок) — что может пойти не так? А нет, в системе есть пакет с файлом **`/usr/bin/kitten`**, который заменился при установке.

Для избежания конфликтов с системой появился метод установки в директории **`/usr/local/`**. В традиционных Linux-дистрибутивах дерево каталогов в **`/usr/local`** не используется, но при этом каталоги **`/usr/local/bin`**, **`/usr/local/lib64`**, **`/usr/local/share/man`** и т. п. считаются стандартными местами для размещения файлов соответствующего типа. Устанавливаемое туда ПО не конфликтует с базовой системой, и именно этот способ включится по умолчанию в GNU Autotools, если отдельно не задать префикс установки. В FHS каталог **`/usr/local`** предназначен для т. н. «локальной установки» ПО, что частично совпадает с нашей целью, а частично — нет. Для установки в **`/usr/local`** нужно иметь права суперпользователя, и продолжать следить за файловыми конфликтами устанавливаемых проектов *между собой*.

Гарантированно не вызывающий конфликтов способ установки — использование для каждого приложения уникального расположения в файловой системе. Такой способ возможен при установке в уникальную директорию **`/opt/<AppName>/`**. Для установки всё ещё требуются права суперпользователя, хотя это уже не обязательно, достаточно программисту выдать отдельный доступный подкаталог. Данный способ позволяет избежать файловых конфликтов. Основной недостаток такого способа — необходимость явного указания в ***`$PATH`*** **`/bin`**-директории с устанавливаемым приложением для его запуска. Что ещё важнее — подгружать динамические библиотеки из **`/opt/<AppName>/lib`** необходимо вручную — как минимум, дополнять путь загрузки с помощью ***`LD_LIBRARY_PATH`***. Следовательно, исполняемый файл в такой установке должен сопровождаться shell-сценарием с соответствующей настойкой. Кроме того, согласно дисциплине ALT, динамическое изменение параметров запуска приложений, включая ***`LD_LIBRARY_PATH`***, ***`D_PRELOAD`***, не заработает для SUID/SGID программ из соображений безопасности. Такая схема установки часто используется в сложных проектах, в состав которых входит большое количество собственных библиотек (в том числе совпадающих по именам с системными, но модифицированных).

Время от времени разработчикам таких проектов приходит в голову, например, добавить **`/opt/<AppName>/lib`** в настройки динамического компоновщика — практика показывает, что это опасная идея: не ровен час, сторонняя библиотека заместит системную…

Для установки без прав суперпользователя располагать файлы можно в подкаталогах **`$HOME/.local/`**. Такие установки являются локальными для конкретного пользователя, что может приводить к установке в одну систему нескольких копий одних и тех же приложений. Также установка в локальное пространство пользователя требует полной настройки зависимостей на другие локальные установки. Такой способ полезен для систем, ориентированных на разработчика, поскольку при разработке часто требуется отдельное (а иногда и не одно) независимое пространство установки, и для него требуется ещё более замысловатая настройка путей для загрузки динамических библиотек (в которых появляется непостоянный компонент ***`$HOME`***), наподобие той, что реализована в GNU Libtool.

Все схемы установки в специализированные директории файловой системы связаны с необходимостью дополнительно настраивать динамическую компоновку с библиотеками, входящими в состав проекта. Проблему можно решить *полностью статической* сборкой и компоновкой. В результате получается статический же исполняемый файл. Статические прогарммы не требуют для запуска никаких библиотек, даже GLibC — только поддержку ядром загружаемого формата ELF. Такие программы можно просто *скопировать* в произвольную Linux-систему, и они там заработают. Например, тремя статически собранными пакетами — *ash-static*, *cpio-static* и *find-static* — пользуется hasher для того, чтобы развернуть изолированное сборочное окружение в chroot-каталоге, в котором изначально нет вообще ничего.

@user

```
[user@VM ~] rpmquery -f /usr/bin/cpio /usr/bin/cpio.static
cpio-2.15-alt1.x86_64
cpio-static-2.15-alt1.x86_64
[user@VM ~] file /usr/bin/cpio /usr/bin/cpio.static
/usr/bin/cpio:        ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=d6dff4c6e0de24cd3c1334b152efcc5ebf89694a, for GNU/Linux 3.2.0,
stripped
/usr/bin/cpio.static: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux),
statically linked, BuildID[sha1]=e047490a73892cb9e762b0e409c73ea774c51ce9, for
GNU/Linux 3.2.0, stripped
[user@VM ~] ldd /usr/bin/cpio /usr/bin/cpio.static
/usr/bin/cpio:
        linux-vdso.so.1 (0x00007f8786626000)
        libc.so.6 => /lib64/libc.so.6 (0x00007f87863de000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f8786628000)
/usr/bin/cpio.static:
        not a dynamic executable
```

Если программа разрабатывается для запуска в каком-то предсказуемом окружении, можно понадеяться на то, что в системе будут доступны стандартные библиотеки с определённым диапазоном версий. Это позволит не «таскать с собой» как минимум тот же **libc**, а при удачном стечении обстоятельств — отказаться и от базовых прикладных инструментариев, таких как Qt или GTK. При этом часть библиотек всё-таки собирается, для удобства — статически. Как правило это библиотеки собственной разработки и сторонние библиотеки, требующие модификации в составе проекта, и какая-нибудь сугубая редкость, надеяться на наличие которой в системе не имеет смысла. Напомним, что разговор идёт возможности установки «по месту», а не о *пакете*, в котором такая «редкость» просто ставится в зависимость. В последнее время стало популярно скачивать собирать из исходных текстов в составе проекта практически все сторонние компоненты. Такой подход называется «вендоринг», и он принят как основной в экосистемах языков Go и Rust.

Программа с завендоренными сторонними компонентами получается довольно большой, версии этих компонент отличаются от сборки к сборке, а получившийся исполняемый файл компонуется динамически, но зависит от минимума системных библиотек (часто — от одной только **libc**).

16.2. Управление установкой с помощью Autotools

Autotools непосредственно поддерживает возможность установки проектов в систему. По умолчанию за установочный путь принят **`/usr/local`**. Автоматически сгенерированные рецепты **`make install`** самостоятельно распределяют все компоненты по поддиректориям. Однако, как было сказано ранее, установка в **`/usr/local`** требует прав суперпользователя:

@user

```
[user@VM syscall-master]$ autoreconf -fisv
<...>
[user@VM syscall-master]$ ./configure
<...>
[user@VM syscall-master]$ make install
Making install in src
<...>
/usr/bin/mkdir -p '/usr/local/lib'
/bin/sh ../libtool   --mode=install /usr/bin/ginstall -c   libsyscall.la '/usr/
local/lib'
libtool: install: /usr/bin/ginstall -c .libs/libsyscall.so.0.0.0 /usr/local/lib/
libsyscall.so.0.0.0
/usr/bin/ginstall: cannot create regular file '/usr/local/lib/
libsyscall.so.0.0.0': Permission denied
<...>
[user@VM syscall-master]$ tree /usr/local
/usr/local
├── bin
│   └── hypersh
├── doc
├── etc
├── games
├── include
├── lib
├── lib64
├── libexec
├── libx32
├── man
├── sbin
└── share
    ├── info
    └── man

15 directories, 1 file
```

Важно

Ещё раз напоминаем, что при работе с правами суперпользователя нужно быть особенно бдительным!

```
[root@VM ~]# cd /home/user/syscall-master/
[root@VM syscall-master]# make install
Making install in src
<...>
/usr/bin/mkdir -p '/usr/local/lib'
/bin/sh ../libtool   --mode=install /usr/bin/ginstall -c   libsyscall.la '/usr/
local/lib'
libtool: install: /usr/bin/ginstall -c .libs/libsyscall.so.0.0.0 /usr/local/lib/
libsyscall.so.0.0.0
libtool: install: (cd /usr/local/lib && { ln -s -f libsyscall.so.0.0.0
libsyscall.so.0 || { rm -f libsyscall.so.0 && ln -s libsyscall.so.0.0.0
libsyscall.so.0; }; })
libtool: install: (cd /usr/local/lib && { ln -s -f libsyscall.so.0.0.0
libsyscall.so || { rm -f libsyscall.so && ln -s libsyscall.so.0.0.0
libsyscall.so; }; })
libtool: install: /usr/bin/ginstall -c .libs/libsyscall.lai /usr/local/lib/
libsyscall.la
/usr/bin/mkdir -p '/usr/local/bin'
 /bin/sh ../libtool   --mode=install /usr/bin/ginstall -c syscall '/usr/local/
bin'
libtool: install: /usr/bin/ginstall -c .libs/syscall /usr/local/bin/syscall
/usr/bin/mkdir -p '/usr/local/share/man/man1'
/usr/bin/ginstall -c -m 644 syscall.1 '/usr/local/share/man/man1'

[root@VM syscall-master]# tree /usr/local
/usr/local
├── bin
│   ├── hypersh
│   └── syscall
├── doc
├── etc
├── games
├── include
├── lib
│   ├── libsyscall.la
│   ├── libsyscall.so -> libsyscall.so.0.0.0
│   ├── libsyscall.so.0 -> libsyscall.so.0.0.0
│   └── libsyscall.so.0.0.0
├── lib64
├── libexec
├── libx32
├── man
├── sbin
└── share
    ├── info
    └── man
        └── man1
            └── syscall.1

16 directories, 8 files
[root@VM syscall-master]#
```

Также **`/usr/local`** не является единственным местом установки даже среди общепринятых. Более того, никто не ограничивает пользователя, разбирающегося в собственных действиях, производить установку в *любые* места системы с оговоркой готовности пользователя самостоятельно настраивать загрузку динамических библиотек.

Для явного указания места установки проекта среди параметров ***configure*** существует ключ ***--prefix=***, использующийся для указания места установки программы:

```
[user@VM syscall-master]$ tree /tmp/
/tmp/
├── systemd-private-cf21b62e498d45469bbfe8c619515c2b-chronyd.service-1UNCOt
[error opening dir]
└── systemd-private-cf21b62e498d45469bbfe8c619515c2b-systemd-logind.service-
pFMnpU  [error opening dir]

3 directories, 0 files
[user@VM syscall-master]$ ./configure --prefix=/tmp/qq
<...>
[user@VM syscall-master]$ make install
Making install in src
<...>
/bin/sh ../libtool   --mode=install /usr/bin/ginstall -c   libsyscall.la '/tmp/
qq/lib'
libtool: install: /usr/bin/ginstall -c .libs/libsyscall.so.0.0.0 /tmp/qq/lib/
libsyscall.so.0.0.0
libtool: install: (cd /tmp/qq/lib && { ln -s -f libsyscall.so.0.0.0
libsyscall.so.0 || { rm -f libsyscall.so.0 && ln -s libsyscall.so.0.0.0
libsyscall.so.0; }; })
libtool: install: (cd /tmp/qq/lib && { ln -s -f libsyscall.so.0.0.0 libsyscall.so
|| { rm -f libsyscall.so && ln -s libsyscall.so.0.0.0 libsyscall.so; }; })
libtool: install: /usr/bin/ginstall -c .libs/libsyscall.lai /tmp/qq/lib/
libsyscall.la
libtool: finish: PATH="/home/user/.gemie/bin:/home/user/bin:/usr/bin:/bin:/usr/
local/bin:/usr/games:/sbin" ldconfig -n /tmp/qq/lib
----------------------------------------------------------------------
Libraries have been installed in:
  /tmp/qq/lib

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the '-LLIBDIR'
flag during linking and do at least one of the following:
  - add LIBDIR to the 'LD_LIBRARY_PATH' environment variable
    during execution
  - add LIBDIR to the 'LD_RUN_PATH' environment variable
    during linking
  - use the '-Wl,-rpath -Wl,LIBDIR' linker flag
  - have your system administrator add LIBDIR to '/etc/ld.so.conf'

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
----------------------------------------------------------------------
/usr/bin/mkdir -p '/tmp/qq/bin'
 /bin/sh ../libtool   --mode=install /usr/bin/ginstall -c syscall '/tmp/qq/bin'
libtool: install: /usr/bin/ginstall -c .libs/syscall /tmp/qq/bin/syscall
/usr/bin/mkdir -p '/tmp/qq/share/man/man1'
/usr/bin/ginstall -c -m 644 syscall.1 '/tmp/qq/share/man/man1'

[user@VM syscall-master]$ tree /tmp/qq
/tmp/qq
├── bin
│   └── syscall
├── lib
│   ├── libsyscall.la
```

```
│   ├── libsyscall.so -> libsyscall.so.0.0.0
│   ├── libsyscall.so.0 -> libsyscall.so.0.0.0
│   └── libsyscall.so.0.0.0
└── share
    └── man
        └── man1
            └── syscall.1

6 directories, 6 files
[user@VM syscall-master]$
```

Место установки можно указать и на более раннем этапе — при оформлении **`configure.ac`**. Для этого в Autotools предусмотрены специальные макросы:

- ***`AC_PREFIX_DEFAULT(<каталог>)`*** используется для явного указания пути установки по умолчанию взамен **`/usr/local`**;
- ***`AC_PREFIX_PROGRAM(<путь>)`*** позволяет указать не явный путь установки, а уже установленную программу для использования её места установки (если он описан в ***`PATH`***). Например, если программой выбрана **gcc**, и ***`PATH`*** содержит путь к **`/usr/local/gnu/bin/gcc`**, путём установки будет выбран **`/usr/local/gnu`**.

@user: **`syscall-master/configure.ac`**

```
<...>
AM_INIT_AUTOMAKE([foreign subdir-objects])
LT_INIT([disable-static])
AC_CONFIG_HEADERS([config.h])

AC_PREFIX_DEFAULT([/tmp/QKRQ])

DX_INIT_DOXYGEN([syscall], [Doxyfile], [doxygen-doc])
<...>
```

@user

```
[user@VM syscall-master]$ autoreconf -fisv
<...>
[user@VM syscall-master]$ ./configure
<...>
[user@VM syscall-master]$ make install
Making install in src
<...>
/usr/bin/mkdir -p '/tmp/QKRQ/lib'
/bin/sh ../libtool   --mode=install /usr/bin/ginstall -c   libsyscall.la '/tmp/
QKRQ/lib'
libtool: install: /usr/bin/ginstall -c .libs/libsyscall.so.0.0.0 /tmp/QKRQ/lib/
libsyscall.so.0.0.0
libtool: install: (cd /tmp/QKRQ/lib && { ln -s -f libsyscall.so.0.0.0
libsyscall.so.0 || { rm -f libsyscall.so.0 && ln -s libsyscall.so.0.0.0
libsyscall.so.0; }; })
libtool: install: (cd /tmp/QKRQ/lib && { ln -s -f libsyscall.so.0.0.0
libsyscall.so || { rm -f libsyscall.so && ln -s libsyscall.so.0.0.0
libsyscall.so; }; })
libtool: install: /usr/bin/ginstall -c .libs/libsyscall.lai /tmp/QKRQ/lib/
libsyscall.la
libtool: finish: PATH="/home/user/.gemie/bin:/home/user/bin:/usr/bin:/bin:/usr/
local/bin:/usr/games:/sbin" ldconfig -n /tmp/QKRQ/lib
```

```
----------------------------------------------------------------------
Libraries have been installed in:
  /tmp/QKRQ/lib

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the '-LLIBDIR'
flag during linking and do at least one of the following:
  - add LIBDIR to the 'LD_LIBRARY_PATH' environment variable
    during execution
  - add LIBDIR to the 'LD_RUN_PATH' environment variable
    during linking
  - use the '-Wl,-rpath -Wl,LIBDIR' linker flag
  - have your system administrator add LIBDIR to '/etc/ld.so.conf'

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
----------------------------------------------------------------------
/usr/bin/mkdir -p '/tmp/QKRQ/bin'
 /bin/sh ../libtool   --mode=install /usr/bin/ginstall -c syscall '/tmp/QKRQ/bin'
libtool: install: /usr/bin/ginstall -c .libs/syscall /tmp/QKRQ/bin/syscall
/usr/bin/mkdir -p '/tmp/QKRQ/share/man/man1'
/usr/bin/ginstall -c -m 644 syscall.1 '/tmp/QKRQ/share/man/man1'

[user@VM syscall-master]$ tree tmp
tmp  [error opening dir]

0 directories, 0 files
[user@VM syscall-master]$ tree /tmp
/tmp
├── QKRQ
│   ├── bin
│   │   └── syscall
│   ├── lib
│   │   ├── libsyscall.la
│   │   ├── libsyscall.so -> libsyscall.so.0.0.0
│   │   ├── libsyscall.so.0 -> libsyscall.so.0.0.0
│   │   └── libsyscall.so.0.0.0
│   └── share
│       └── man
│           └── man1
│               └── syscall.1
├── qq
│   ├── bin
│   │   └── syscall
│   ├── lib
│   │   ├── libsyscall.la
│   │   ├── libsyscall.so -> libsyscall.so.0.0.0
│   │   ├── libsyscall.so.0 -> libsyscall.so.0.0.0
│   │   └── libsyscall.so.0.0.0
│   └── share
│       └── man
│           └── man1
│               └── syscall.1
├── systemd-private-cf21b62e498d45469bbfe8c619515c2b-chronyd.service-1UNCOt
[error opening dir]
└── systemd-private-cf21b62e498d45469bbfe8c619515c2b-systemd-logind.service-
```

```
pFMnpU  [error opening dir]

15 directories, 12 files
[user@VM syscall-master]$
```

16.3. Собственный репозиторий

Одним из преимуществ работы работы с Gear и Hasher является автоматическое ведение собственного репозитория пакетов, собираемых в системе. Каталог с установочными пакетами и с пакетами с исходниками находится в специальной директории **~/hasher/repo/**:

@user

```
[user@VM ~]$ tree ~/hasher/repo/
/home/user/hasher/repo/
├── SRPMS.hasher
│   ├── autoenv-pkg-1.0-alt1.src.rpm
│   ├── autoenv-pkg-1.1-alt1.src.rpm
│   ├── double-1.0-alt1.src.rpm
│   ├── gdb-check-pkg-1.0-alt1.src.rpm
│   ├── GNU_picture-1.0-alt1.src.rpm
│   ├── hello-upgrade-1.0-alt1.src.rpm
│   ├── inc-1.0-alt1.src.rpm
│   ├── Multilab-1.0-alt1.src.rpm
│   ├── not-null-pkg-1.0-alt1.src.rpm
│   ├── null-pkg-1.0-alt1.src.rpm
│   ├── pkg-ncurses-1.0-alt1.src.rpm
│   ├── regex-pkg-1.0-alt1.src.rpm
│   ├── sheepcounter-0.0-alt1.src.rpm
│   ├── sheepcounter-1.0-alt1.src.rpm
│   ├── sheepcounter-pkg-1.0-alt1.src.rpm
│   ├── strace-pkg-1.0-alt1.src.rpm
│   ├── syscall-1.0-alt1.src.rpm
│   ├── syscall-1.1-alt1.src.rpm
│   ├── syscall-1.2-alt1.src.rpm
│   ├── syscall-1.3-alt1.src.rpm
│   ├── todo-pkg-1.0-alt1.src.rpm
│   └── Weatherminal-1.0-alt1.src.rpm
└── x86_64
    └── RPMS.hasher
        ├── autoenv-pkg-1.0-alt1.x86_64.rpm
        ├── autoenv-pkg-1.1-alt1.x86_64.rpm
        ├── autoenv-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── autoenv-pkg-debuginfo-1.1-alt1.x86_64.rpm
        ├── double-1.0-alt1.x86_64.rpm
        ├── double-second-1.0-alt1.x86_64.rpm
        ├── gdb-check-pkg-1.0-alt1.x86_64.rpm
        ├── gdb-check-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── GNU_picture-1.0-alt1.x86_64.rpm
        ├── GNU_picture-data-1.0-alt1.noarch.rpm
        ├── GNU_picture-data-1.0-alt1.x86_64.rpm
        ├── GNU_picture-debuginfo-1.0-alt1.x86_64.rpm
        ├── hello-upgrade-1.0-alt1.x86_64.rpm
        ├── hello-upgrade-debuginfo-1.0-alt1.x86_64.rpm
        ├── inc-1.0-alt1.x86_64.rpm
        ├── inc-debuginfo-1.0-alt1.x86_64.rpm
        ├── Multilab-1.0-alt1.x86_64.rpm
```

```
        ├── Multilab-debuginfo-1.0-alt1.x86_64.rpm
        ├── not-null-pkg-1.0-alt1.x86_64.rpm
        ├── null-pkg-1.0-alt1.x86_64.rpm
        ├── pkg-ncurses-1.0-alt1.x86_64.rpm
        ├── pkg-ncurses-debuginfo-1.0-alt1.x86_64.rpm
        ├── regex-pkg-1.0-alt1.x86_64.rpm
        ├── regex-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── sheepcounter-0.0-alt1.x86_64.rpm
        ├── sheepcounter-1.0-alt1.x86_64.rpm
        ├── sheepcounter-debuginfo-0.0-alt1.x86_64.rpm
        ├── sheepcounter-debuginfo-1.0-alt1.x86_64.rpm
        ├── sheepcounter-pkg-1.0-alt1.x86_64.rpm
        ├── sheepcounter-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── strace-pkg-1.0-alt1.x86_64.rpm
        ├── strace-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── syscall-1.0-alt1.x86_64.rpm
        ├── syscall-1.1-alt1.x86_64.rpm
        ├── syscall-1.2-alt1.x86_64.rpm
        ├── syscall-1.3-alt1.x86_64.rpm
        ├── syscall-debuginfo-1.0-alt1.x86_64.rpm
        ├── syscall-debuginfo-1.1-alt1.x86_64.rpm
        ├── syscall-debuginfo-1.2-alt1.x86_64.rpm
        ├── syscall-debuginfo-1.3-alt1.x86_64.rpm
        ├── todo-pkg-1.0-alt1.x86_64.rpm
        └── Weatherminal-1.0-alt1.x86_64.rpm

4 directories, 64 files
[user@VM ~]$
```

При успешной пересборке пакета с помощью hasher в *чистом окружении* (не изнутри hasher с помощью ***rpmbuild***, а через Gear (***gear-hsh***) или с помощью hsh <name.src.rpm>) содержимое этих каталогов автоматически обновляется:

@user

```
[user@VM ~]$ ls -la ~/hasher/repo/x86_64/RPMS.hasher/null-pkg-1.0-alt1.x86_64.rpm
-rw-r--r-- 1 user_b user_b 1377 июл  9 21:13 /home/user/hasher/repo/x86_64/
RPMS.hasher/null-pkg-1.0-alt1.x86_64.rpm

[user@VM ~]$ hsh ~/hasher/repo/SRPMS.hasher/null-pkg-1.0-alt1.src.rpm
<86>Aug 23 08:29:47 userdel[1115812]: delete user 'rooter'
<86>Aug 23 08:29:47 userdel[1115812]: removed group 'rooter' owned by 'rooter'
<86>Aug 23 08:29:48 groupadd[1115819]: group added to /etc/group: name=rooter,
GID=1003
<86>Aug 23 08:29:48 groupadd[1115819]: group added to /etc/gshadow: name=rooter
<86>Aug 23 08:29:48 groupadd[1115819]: new group: name=rooter, GID=1003
<86>Aug 23 08:29:48 useradd[1115825]: new user: name=rooter, UID=1003, GID=1003,
home=/root, shell=/bin/bash, from=none
<86>Aug 23 08:29:48 userdel[1115835]: delete user 'builder'
<86>Aug 23 08:29:48 userdel[1115835]: removed group 'builder' owned by 'builder'
<86>Aug 23 08:29:48 userdel[1115835]: removed shadow group 'builder' owned by
'builder'
<86>Aug 23 08:29:49 groupadd[1115842]: group added to /etc/group: name=builder,
GID=1004
<86>Aug 23 08:29:49 groupadd[1115842]: group added to /etc/gshadow: name=builder
<86>Aug 23 08:29:49 groupadd[1115842]: new group: name=builder, GID=1004
<86>Aug 23 08:29:49 useradd[1115848]: new user: name=builder, UID=1004, GID=1004,
home=/usr/src, shell=/bin/bash, from=none
Building target platforms: x86_64
```

```
Building for target x86_64
Wrote: /usr/src/in/nosrpm/null-pkg-1.0-alt1.src.rpm (w1.gzdio)
Installing null-pkg-1.0-alt1.src.rpm
Building target platforms: x86_64
Building for target x86_64
Processing files: null-pkg-1.0-alt1
Wrote: /usr/src/RPM/SRPMS/null-pkg-1.0-alt1.src.rpm (w2.lzdio)
Wrote: /usr/src/RPM/RPMS/x86_64/null-pkg-1.0-alt1.x86_64.rpm (w2.lzdio)
0.01user 0.01system 0:00.02elapsed 96%CPU (0avgtext+0avgdata 5900maxresident)k
0inputs+40outputs (0major+1010minor)pagefaults 0swaps

[user@VM ~]$ ls -la ~/hasher/repo/x86_64/RPMS.hasher/null-pkg-1.0-alt1.x86_64.rpm
-rw-r--r-- 1 user_d user_d 1377 авг 23 11:29 /home/user/hasher/repo/x86_64/
RPMS.hasher/null-pkg-1.0-alt1.x86_64.rpm
[user@VM ~]$
```

Каталог **`/home/user/hasher/repo`** может вполне выступать в качестве дополнительного *репозитория* пакетов системы. Обновление из такого репозитория будет происходить даже при совпадении версий: в пакете зафиксировано время сборки, и более новым считается собранный позже.

Все пути к преднастроенным репозиториям, используемым в системе, хранятся в файлах **`/etc/apt/sources.list`** и **`/etc/apt/sources.list/*.list`**. Формат описания репозитория в файле следующий:

- способ организации (тип) репозитория;
- (*необязательный*) ключ, которым подписаны индексы;
- URL хранилища;
- архитектура;
- раздел.

@user

```
[user@VM ~]$ cat /etc/apt/sources.list
# Local package resource list for APT goes here.
# To inspect package defined part, see /etc/apt/sources.list.d/*.list

[user@VM ~]$ ls /etc/apt/sources.list.d/
alt.list  heanet.list  ipsl.list  yandex.list

[user@VM ~]$ cat /etc/apt/sources.list.d/alt.list
# ftp.altlinux.org (ALT Linux, Moscow)

# ALT Platform 11
#rpm [p11] ftp://ftp.altlinux.org/pub/distributions/ALTLinux p11/branch/x86_64
classic
#rpm [p11] ftp://ftp.altlinux.org/pub/distributions/ALTLinux p11/branch/x86_64-
i586 classic
#rpm [p11] ftp://ftp.altlinux.org/pub/distributions/ALTLinux p11/branch/noarch
classic

rpm [p11] http://ftp.altlinux.org/pub/distributions/ALTLinux p11/branch/x86_64
classic
rpm [p11] http://ftp.altlinux.org/pub/distributions/ALTLinux p11/branch/x86_64-
```

```
i586 classic
rpm [p11] http://ftp.altlinux.org/pub/distributions/ALTLinux p11/branch/noarch
classic

#rpm [p11] rsync://ftp.altlinux.org/ALTLinux p11/branch/x86_64 classic
#rpm [p11] rsync://ftp.altlinux.org/ALTLinux p11/branch/x86_64-i586 classic
#rpm [p11] rsync://ftp.altlinux.org/ALTLinux p11/branch/noarch classic

[user@VM ~]$
```

Локальный репозиторий — например, для разработчика, который что-то правит в пакете и тут же проверяет результат — можно оформить прямо в каталоге **/home/user/hasher/repo**: добавить его в список доступных репозиториев напрямую, с типом ***rpm-dir*** (репозиторий без индекса):

@user

```
[user@VM ~]$ su -
Password:
[root@VM ~]# vim /etc/apt/sources.list
[root@VM ~]#
выход
[user@VM ~]$ cat /etc/apt/sources.list
# Local package resource list for APT goes here.
# To inspect package defined part, see /etc/apt/sources.list.d/*.list

rpm-dir file:///home/user/hasher repo/x86_64 hasher
[user@VM ~]$
```

Теперь можно пользоваться репозиторием, как и другими — устанавливать пакеты с помощью пакетных менеджеров:

@root

```
[root@VM ~]# apt-repo
rpm-dir file:///home/user/hasher repo/x86_64 hasher
rpm [p11] http://ftp.altlinux.org/pub/distributions/ALTLinux p11/branch/x86_64
classic
rpm [p11] http://ftp.altlinux.org/pub/distributions/ALTLinux p11/branch/x86_64-
i586 classic
rpm [p11] http://ftp.altlinux.org/pub/distributions/ALTLinux p11/branch/noarch
classic

[root@VM ~]# apt-get install not-null-pkg
Чтение списков пакетов... Завершено
Построение дерева зависимостей... Завершено
Следующие НОВЫЕ пакеты будут установлены:
 not-null-pkg
0 будет обновлено, 1 новых установлено, 0 пакетов будет удалено и 60 не будет
обновлено.
Необходимо получить 0B/1845B архивов.
После распаковки потребуется дополнительно 29B дискового пространства.
Совершаем изменения...
Подготовка...
########################################################################## [100%]
Обновление / установка...
1: not-null-pkg-1.0-alt1
########################################################################## [100%]
```

```
Завершено.
[root@VM ~]# not-null-pkg
This is not null pkg
[root@VM ~]#
```

Этот репозиторий по умолчанию уже включён в сам hasher, именно поэтому только что собранные пакеты можно немедленно устанавливать с помощью hsh-install.

16.4. Репозиторий и индексами

Просматривая репозиторий без индексов (в которых хранится информация о зависимостях, конфликтах, времени сборки и т. д.), APT будет регулярно строить их динамически. Если пакетов в нём достаточно, это может занимать довольно много времени. Кроме того, без индексов не будет работать доступ по сети.

Создать стандартный репозиторий типа ***rpm*** довольно просто.

Репозиторий APT типа ***rpm*** состоит из двух компонентов: *индексов*, где описаны все пакеты, ссылки на них и их версии, и самих *.rpm*-пакетов.

Каталог для репозитория создаётся по особым правилам:

- в каталоге должно быть хранилище пакетов в формате **…/Раздел/RPMS.компонент**:
  - **раздел** — описание архитектуры хранимых пакетов. Ограничений на его именование нет, однако обычно оно соответствует аппаратной архитектуре, под которую собран пакет — **x86_64**, **aarch64** и т. д. для архитектурно зависимых пакетов, **noarch** — для архитектурно независимых;
  - **компонент** используется для классификации и организации пакетов в репозитории по функциональности или назначению. Ограничений на его именование нет;
  - **RPMS.компонент** — непосредственно каталог с пакетами.

@user

```
[user@VM ~]$ mkdir NewRepo
[user@VM ~]$ cd NewRepo/
[user@VM NewRepo]$ mkdir -p x86_64/RPMS.classic
[user@VM NewRepo]$ cp ~/hasher/repo/x86_64/RPMS.hasher/* x86_64/RPMS.classic/
[user@VM NewRepo]$ tree
.
└── x86_64
    └── RPMS.classic
        ├── autoenv-pkg-1.0-alt1.x86_64.rpm
        ├── autoenv-pkg-1.1-alt1.x86_64.rpm
        ├── autoenv-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── autoenv-pkg-debuginfo-1.1-alt1.x86_64.rpm
        ├── double-1.0-alt1.x86_64.rpm
        ├── double-second-1.0-alt1.x86_64.rpm
        ├── gdb-check-pkg-1.0-alt1.x86_64.rpm
        ├── gdb-check-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── GNU_picture-1.0-alt1.x86_64.rpm
        ├── GNU_picture-data-1.0-alt1.noarch.rpm
        ├── GNU_picture-data-1.0-alt1.x86_64.rpm
        ├── GNU_picture-debuginfo-1.0-alt1.x86_64.rpm
        ├── hello-upgrade-1.0-alt1.x86_64.rpm
```

```
        ├── hello-upgrade-debuginfo-1.0-alt1.x86_64.rpm
        ├── inc-1.0-alt1.x86_64.rpm
        ├── inc-debuginfo-1.0-alt1.x86_64.rpm
        ├── Multilab-1.0-alt1.x86_64.rpm
        ├── Multilab-debuginfo-1.0-alt1.x86_64.rpm
        ├── not-null-pkg-1.0-alt1.x86_64.rpm
        ├── null-pkg-1.0-alt1.x86_64.rpm
        ├── pkg-ncurses-1.0-alt1.x86_64.rpm
        ├── pkg-ncurses-debuginfo-1.0-alt1.x86_64.rpm
        ├── regex-pkg-1.0-alt1.x86_64.rpm
        ├── regex-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── sheepcounter-0.0-alt1.x86_64.rpm
        ├── sheepcounter-1.0-alt1.x86_64.rpm
        ├── sheepcounter-debuginfo-0.0-alt1.x86_64.rpm
        ├── sheepcounter-debuginfo-1.0-alt1.x86_64.rpm
        ├── sheepcounter-pkg-1.0-alt1.x86_64.rpm
        ├── sheepcounter-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── strace-pkg-1.0-alt1.x86_64.rpm
        ├── strace-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── syscall-1.0-alt1.x86_64.rpm
        ├── syscall-1.1-alt1.x86_64.rpm
        ├── syscall-1.2-alt1.x86_64.rpm
        ├── syscall-1.3-alt1.x86_64.rpm
        ├── syscall-debuginfo-1.0-alt1.x86_64.rpm
        ├── syscall-debuginfo-1.1-alt1.x86_64.rpm
        ├── syscall-debuginfo-1.2-alt1.x86_64.rpm
        ├── syscall-debuginfo-1.3-alt1.x86_64.rpm
        ├── todo-pkg-1.0-alt1.x86_64.rpm
        └── Weatherminal-1.0-alt1.x86_64.rpm

3 directories, 42 files
[user@VM NewRepo]$
```

Для создания индексов для репозитория используется пакет *apt-repo-tools* и утилита **genbasedir**, создающий индексы для пакетов одного раздела репозитория. Обратите внимание на то, что **hasher** не определяет архитектурную зависимость пакета: для него они все **x86_64**. Также стоит заметить, что репозиторий формируется «в рабочем порядке», его непротиворечивость ничем не гарантирована (например, у нас образовалось два одноимённых пакета *GNU_picture-data* разной архитектуры):

@user

```
[user@VM NewRepo]$ genbasedir --create --progress --topdir=. x86_64 classic
Creating base directory... done
Components: classic
Processing packages... RPMS.classic 42/42 42/42 done
Waiting for bzip2 and xz to finish... done
Creating component releases... classic done
Updating global release file... done
Appending MD5Sum... classic done
Appending BLAKE2b... classic done
All your base are belong to us!!!
[user@VM NewRepo]$ tree
.
└── x86_64
    ├── base
    │   ├── pkglist.classic
    │   ├── pkglist.classic.bz2
```

```
    │   ├── pkglist.classic.xz
    │   ├── release
    │   └── release.classic
    └── RPMS.classic
        ├── autoenv-pkg-1.0-alt1.x86_64.rpm
        ├── autoenv-pkg-1.1-alt1.x86_64.rpm
        ├── autoenv-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── autoenv-pkg-debuginfo-1.1-alt1.x86_64.rpm
        ├── double-1.0-alt1.x86_64.rpm
        ├── double-second-1.0-alt1.x86_64.rpm
        ├── gdb-check-pkg-1.0-alt1.x86_64.rpm
        ├── gdb-check-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── GNU_picture-1.0-alt1.x86_64.rpm
        ├── GNU_picture-data-1.0-alt1.noarch.rpm
        ├── GNU_picture-data-1.0-alt1.x86_64.rpm
        ├── GNU_picture-debuginfo-1.0-alt1.x86_64.rpm
        ├── hello-upgrade-1.0-alt1.x86_64.rpm
        ├── hello-upgrade-debuginfo-1.0-alt1.x86_64.rpm
        ├── inc-1.0-alt1.x86_64.rpm
        ├── inc-debuginfo-1.0-alt1.x86_64.rpm
        ├── Multilab-1.0-alt1.x86_64.rpm
        ├── Multilab-debuginfo-1.0-alt1.x86_64.rpm
        ├── not-null-pkg-1.0-alt1.x86_64.rpm
        ├── null-pkg-1.0-alt1.x86_64.rpm
        ├── pkg-ncurses-1.0-alt1.x86_64.rpm
        ├── pkg-ncurses-debuginfo-1.0-alt1.x86_64.rpm
        ├── regex-pkg-1.0-alt1.x86_64.rpm
        ├── regex-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── sheepcounter-0.0-alt1.x86_64.rpm
        ├── sheepcounter-1.0-alt1.x86_64.rpm
        ├── sheepcounter-debuginfo-0.0-alt1.x86_64.rpm
        ├── sheepcounter-debuginfo-1.0-alt1.x86_64.rpm
        ├── sheepcounter-pkg-1.0-alt1.x86_64.rpm
        ├── sheepcounter-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── strace-pkg-1.0-alt1.x86_64.rpm
        ├── strace-pkg-debuginfo-1.0-alt1.x86_64.rpm
        ├── syscall-1.0-alt1.x86_64.rpm
        ├── syscall-1.1-alt1.x86_64.rpm
        ├── syscall-1.2-alt1.x86_64.rpm
        ├── syscall-1.3-alt1.x86_64.rpm
        ├── syscall-debuginfo-1.0-alt1.x86_64.rpm
        ├── syscall-debuginfo-1.1-alt1.x86_64.rpm
        ├── syscall-debuginfo-1.2-alt1.x86_64.rpm
        ├── syscall-debuginfo-1.3-alt1.x86_64.rpm
        ├── todo-pkg-1.0-alt1.x86_64.rpm
        └── Weatherminal-1.0-alt1.x86_64.rpm

4 directories, 47 files
[user@VM NewRepo]$
```

Добавим репозиторий к другим доступным и проверим работу индексов:

@root

```
[root@VM ~]# cat /etc/apt/sources.list
# Local package resource list for APT goes here.
# To inspect package defined part, see /etc/apt/sources.list.d/*.list

rpm-dir file:///home/user/NewRepo x86_64 classic
```

```
[root@VM ~]# apt-repo
rpm-dir file:///home/user/NewRepo x86_64 classic
rpm [p11] http://ftp.altlinux.org/pub/distributions/ALTLinux p11/branch/x86_64
classic
rpm [p11] http://ftp.altlinux.org/pub/distributions/ALTLinux p11/branch/x86_64-
i586 classic
rpm [p11] http://ftp.altlinux.org/pub/distributions/ALTLinux p11/branch/noarch
classic

[root@VM ~]# apt-cache search GNU_picture
GNU_picture-debuginfo - Print your special GNU picture (debug files)
GNU_picture-data - Your special GNU picture
GNU_picture - Print your special GNU picture
[root@VM ~]#
```

Соблюдая надлежащую дисциплину (например, вовремя удаляя устаревшие и неактуальные пакеты), репозиторий можно опубликовать и использовать для обновления нескольких систем.

Запись в **`/etc/sources.list`** примет вид:

```
rpm http://<адрес_сервера>/<путь_до_NewRepo> x86_64 classic
```

Протестируем репозиторий, запустив примитивный HTTP-сервер:

@**user**

```
[user@VM NewRepo]$ python3 -m http.server -d
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Добавим информацию о репозитории в **`sources.list`**. Заметим, что URL не содержит дополнительных компонентов пути, потому что базовым каталогом HTTP-сервера является каталог с репозиторием:

@**root2**

```
[root@VM ~]# cat /etc/apt/sources.list
# Local package resource list for APT goes here.
# To inspect package defined part, see /etc/apt/sources.list.d/*.list

rpm http://192.168.0.1:8000 x86_64 classic
[root@VM ~]#
```

Обновим индексы с помощью **`apt-get update`**:

@**root2**

```
[root@VM ~]# apt-get update
Получено: 1 http://192.168.0.1:8000 x86_64 release [1068B]
Получено: 2 http://ftp.altlinux.org p11/branch/x86_64 release [4210B]
Получено: 3 http://ftp.altlinux.org p11/branch/x86_64-i586 release [1665B]
Получено: 4 http://ftp.altlinux.org p11/branch/noarch release [2831B]
Получено 9774B за 0s (119kB/s).
Получено: 1 http://192.168.0.1:8000 x86_64/classic pkglist [8276B]
Получено: 2 http://192.168.0.1:8000 x86_64/classic release [126B]
Найдено http://ftp.altlinux.org p11/branch/x86_64/classic pkglist
```

```
Найдено http://ftp.altlinux.org p11/branch/x86_64/classic release
Найдено http://ftp.altlinux.org p11/branch/x86_64-i586/classic pkglist
Найдено http://ftp.altlinux.org p11/branch/x86_64-i586/classic release
Найдено http://ftp.altlinux.org p11/branch/noarch/classic pkglist
Найдено http://ftp.altlinux.org p11/branch/noarch/classic release
Получено 8402B за 0s (363kB/s).
Чтение списков пакетов... Завершено
Построение дерева зависимостей... Завершено
[root@VM ~]#
```

Посмотрим информацию о локально собранных пакетах:

@**root2**

```
[root@VM ~]# apt-cache search autoenv
autoenv-pkg - Test pkg with autotool
autoenv-pkg-debuginfo - Test pkg with autotool (debug files)
[root@VM ~]#
```