

Linux для себя

None

Linux4Yourself community

None

Table of contents

1. Linux для себя	7
2. Журнал изменений	8
2.1 Благодарности	8
2.2 Изменения	8
2.3 Обновления пакетов	8
2.4 20.07.2021	9
3. Предисловие	10
3.1 Предисловие	10
3.2 От авторов	11
3.3 Преимущества	12
3.4 Прежде чем начать	13
3.5 Принятые обозначения	14
3.6 Информация об используемых пакетах	15
3.7 Опечатки и неточности	16
4. Подготовительные работы	17
4.1 Целевые архитектуры	17
4.2 Требования к рабочей станции и операционной системе	18
4.3 О времени сборки пакетов	20
4.4 Установка переменной \$LIN	21
4.5 Создание базовых директорий и символических ссылок	22
4.6 Требуемые пакеты и патчи	23
4.7 Создание пользователя LIN	24
4.8 Настройка окружения	25
4.9 О тестах	27
4.10 О приоритете	28
4.11 Общая инструкция по сборке пакетов	29
4.12 О потоках	30
5. Сборка кросс-компилятора	31
5.1 Сборка кросс компилятора	31
5.2 Binutils	32
5.3 GCC	33
6. Заголовочные файлы ядра Linux	35
6.1 Сборка	35
6.2 Установка	35

7. Сборка временной системы	36
7.1 Сборка временной системы	36
7.2 glibc	37
7.3 LibstdC++ Проход 1	40
7.4 m4	41
7.5 ncurses	42
7.6 bash	44
7.7 coreutils	45
7.8 diffutils	46
7.9 file	47
7.10 findutils	48
7.11 gawk	49
7.12 grep	50
7.13 gzip	51
7.14 make	52
7.15 patch	53
7.16 sed	54
7.17 tar	55
7.18 xz	56
7.19 binutils проход 2	57
7.20 gcc проход 2	58
7.21 Смена владельца для каталога \$LIN	60
7.22 Вход в окружение chroot	61
7.23 Создание основных файлов и символических ссылок	62
7.24 LibstdC++ проход 2	63
7.25 gettext	64
7.26 bison	65
7.27 perl	66
7.28 python	67
7.29 texinfo	68
7.30 util-linux	69
7.31 Очистка и сохранение временной системы	71
8. Сборка Linux системы	72
8.1 Сборка конечной Linux системы	72
8.2 Настройка окружения bash	73
8.3 iana-etc	75
8.4 glibc	76
8.5 zlib-ng	82

8.6 bzip2	84
8.7 xz	86
8.8 zstd	88
8.9 file	89
8.10 readline	90
8.11 m4	91
8.12 bc	92
8.13 flex	93
8.14 tcl	94
8.15 expect	95
8.16 dejagnu	96
8.17 binutils	97
8.18 gmp	99
8.19 mpfr	100
8.20 mpc	101
8.21 isl	102
8.22 attr	103
8.23 acl	104
8.24 libcap	105
8.25 shadow	106
8.26 gcc	107
8.27 pkg-config	110
8.28 ncurses	111
8.29 sed	113
8.30 psmisc	114
8.31 gettext	115
8.32 bison	116
8.33 grep	117
8.34 bash	118
8.35 libtool	119
8.36 gdbm	120
8.37 gperf	121
8.38 expat	122
8.39 inetutils	123
8.40 perl	124
8.41 XML::Parser	125
8.42 intltool	126
8.43 autoconf	127

8.44	automake	128
8.45	kmod	129
8.46	libelf	131
8.47	libffi	132
8.48	openssl	133
8.49	python	134
8.50	ninja	135
8.51	meson	136
8.52	coreutils	137
8.53	check	139
8.54	diffutils	140
8.55	gawk	141
8.56	findutils	142
8.57	groff	143
8.58	less	144
8.59	gzip	145
8.60	iproute2	146
8.61	kbd	147
8.62	libpipeline	148
8.63	make	149
8.64	patch	150
8.65	tar	151
8.66	man-db	152
8.67	texinfo	156
8.68	popt	157
8.69	freetype	158
8.70	dosfstools	159
8.71	wget	160
8.72	libtasn1	161
8.73	p11-kit	162
8.74	make-ca	163
8.75	MarkupSafe	164
8.76	Jinja2	165
8.77	Выбор текстового редактора	166
8.78	Выбор системы инициализации	170
8.79	e2fsprogs	184
8.80	GRUB	185
8.81	Очистка системы	192

9. Настройка системы	193
9.1 Настройка системы	193
9.2 Создание файла fstab	194
9.3 Создание файла /etc/shells	196
9.4 systemd	197
9.5 SysVInit	201
10. Настройка и установка ядра	203
10.1 Настройка и установка ядра	203
10.2 Настройка	204
10.3 О прошивках	209
11. Делаем систему загрузочной	210
11.1 Делаем систему загрузочной	210
11.2 Создание загрузочной системы EFI	211
11.3 Создание файла конфигурации GRUB	212
11.4 Создание загрузочной системы Legacy Boot MBR	214
12. Заключительная часть	216
12.1 Заключительная часть	216
13. Вспомогательные материалы	220
13.1 Вспомогательные материалы	220
13.2 Строение GNU/Linux	221
13.3 Установка программ из исходного кода в Linux	258
13.4 Решение ошибок сборки	261
13.5 Кросс-компилятор	263
13.6 Выбор размера файла подкачки	265
13.7 Настройка zram	267
13.8 О шебангах в скриптах Linux	269

1. Linux для себя



Linux для Себя

Подробное руководство по созданию собственной
Linux системы из исходного кода

Lx4U или "Linux для себя" - русскоязычное подробное руководство по созданию Linux-системы, используя лишь исходные тексты необходимого программного обеспечения. Это руководство - самостоятельное ответвление от проекта [Linux From Scratch](#).

На ваш выбор предлагается использование multilib системы, поддержка EFI и набор дополнительного программного обеспечения для организации комфортной работы. Кроме того, существует расширенное руководство, в котором содержится информация о настройке окружения рабочего стола и программного обеспечения, охватывающего различные области применения.

Но главная идея остаётся - вы в центре всего, и вы решаете, какой должна быть Ваша собственная система. А мы просто протягиваем Вам руку помощи.

Начните собственное путешествие в мир Linux прямо сейчас!

2. Журнал изменений

2.1 Благодарности

- [Михаил](#) за реализацию разделов по сборке пакета, используя раздельную структуру каталогов

2.2 Изменения

- Миграция на mkdocs
- [Исправлено #870](#)
- [Исправлено #891](#)
- [Исправлено #877](#)
- [Исправлено #863](#)
- [Исправлено #849](#)
- [Исправлено #816](#)
- [Выполнено #756](#)
- [Выполнено #767](#)
- [Выполнено #779](#)
- [Исправлено #794](#)
- [Исправлено #776](#)
- [Исправлено #777](#)
- [Исправлено #797](#)
- [Исправлено #804](#)
- [Исправлено #815](#)
- [Исправлено #821](#)
- [Исправлено #829](#)

2.3 Обновления пакетов

- tzdata-2021b
- meson-0.59.2
- linux-5.14.9
- bison-3.8.2
- coreutils-9.0
- automake-1.16.4
- bc-5.0.1
- dbus-1.13.18
- diffutils-3.8
- e2fsprogs-1.46.4
- gcc-11.2.0
- gdbm-1.21
- glibc-2.34
- grep-3.7

- gzip-1.11
- inetutils-2.2
- iproute2-5.14.0
- libcap-2.53
- openssl-3.0.0
- shadow-4.9
- systemd-249.4
- util-linux-2.37.2
- vim-8.2.3455
- wget-1.21.2
- zlib-ng-2.0.5

2.4 20.07.2021

- Выпущен релиз 1.3

3. Предисловие

3.1 Предисловие

Существует огромное количество операционных систем семейства Linux, каждая из которых предлагает то или иное преимущество. Различные варианты могут затруднить выбор и забрать много времени для поиска необходимого именно под Ваши задачи.

Linux распахивает дверь в гигантский мир открытых систем, в котором существует огромное количество средств для решения самых разнообразных задач.

Нужна ОС попроще? Для игр? А, возможно, нужен мультимедийный дистрибутив для комфортного просмотра и прослушивания медиа контента?

Используя это руководство, вы получите работающую систему на ваш вкус. вы всё контролируете, а мы просто протягиваем Вам руку помощи.

Начните собственное путешествие в мир Linux прямо сейчас!

3.2 От авторов

Уважаемый читатель!

Мы ценим, что вы читаете это руководство. Мы искренне надеемся, что оно принесёт Вам как образовательную пользу, так и практическую.

Если у вас возникли вопросы, проблемы, или вы хотите внести свой вклад в развитие этого проекта - пожалуйста, оставьте запрос в нашем официальном репозитории по адресу [Linux4Yourself/Linux4Yourself.Book](https://github.com/Linux4Yourself/Linux4Yourself.Book).

С уважением, команда проекта «Linux для себя».

3.3 Преимущества

У проекта несколько целей. Одна из них - образовательная. Создание системы по этой книге поможет узнать, для чего используется каждый пакет и каждая программа, как компоненты системы работают вместе и взаимодействуют друг с другом. Так же вы получите много опыта в компиляции ПО из исходного кода и решения проблем при сборке.

Другая цель - сборка системы, отвечающей потребностям конечного пользователя. Большинство дистрибутивов включают в себя большое число программного обеспечения. Не всё можно удалить "безболезненно" для системы. В LX4 вы можете диктовать каждый аспект своей системы, создать миниатюрную систему, способную стабильно и быстро работать на слабом оборудовании, а можете собрать довольно тяжёлый дистрибутив для мощного ПК.

Преимуществом собственной сборки Linux является безопасность. При компиляции каждого компонента системы из исходных кодов вы можете всё проверить и применить необходимые патчи. Теперь не надо ждать, когда кто-нибудь скомпилирует пакет с требуемыми исправлениями.

3.4 Прежде чем начать

Сборка своей системы - не самая простая задача. Потребуются знания в администрировании систем семейства Unix, чтобы вы смогли устранять проблемы в процессе сборки и правильно выполнять ввод требуемых команд.

Как минимум, вы должны уметь пользоваться командной оболочкой, копировать и выполнять перемещение файлов и каталогов, просматривать списки каталогов и содержимое файлов и изменять текущий каталог. Также ожидается, что у вас есть знания о процессе установки программного обеспечения в системах Linux.

Рекомендуем чаще обращаться к разделу [вспомогательные материалы](#), в котором находятся разъяснения по многим вопросам. Этот раздел часто дополняется и актуализируется.

Советуем вам не пропускать тестирование скомпилированных программ, если тесты указаны в книге. Тестирование поможет вам сделать ваш дистрибутив максимально стабильным и надёжным. В случае ошибок тестирования вы можете пересобрать нужный пакет, чтобы он работал корректно.

3.5 Принятые обозначения

В руководстве используются следующие обозначения:

```
1 ./configure --prefix=/usr
```

Этот текст необходимо набрать в командной строке в точности так, как показано, если иное не сказано в тексте рядом. Это оформление также используется в объяснениях, когда указываются команды.

В некоторых случаях строка разделяется до двух или более линий с использованием символа обратного слэша в конце строки.

```
1 CC="gcc -B/usr/bin/" ./binutils-2.18/configure \
2 --prefix=/tools --disable-nls --disable-werror
```

Обратите внимание, что после обратного слэша должен быть перевод строки. Другие символы приведут к некорректному результату.

```
1 install-info: unknown option '--dir-file=/mnt/lin/usr/info/dir'
```

Этот текст показывает вывод результатов на экран.

Используется чтобы подчеркнуть важную информацию, на которую следует обратить внимание.

Используется для ссылок на страницы проекта, а также на внешние источники. Может включать справочную информацию, ссылки на загрузки и различные сайты.

Warning

Используется для указания на критически важную информацию, на которую следует обратить особое внимание.

Info

Используется для указания на информацию рекомендательного характера. Рекомендуется не пропускать данные указания и внимательно с ними ознакомиться.

Советуем Вам следовать рекомендациям, приведённым в книге для достижения наилучшего результата сборки системы.

3.6 Информация об используемых пакетах

Как говорилось ранее, одной из целей проекта является сборка системы базового уровня. Она будет включать в себя пакеты, необходимые для репликации и распространения, а также относительно небольшой набор программ, с помощью которых можно расширять систему в любом направлении на ваше усмотрение. Это не значит, что она будет максимально компактной.

Есть пакеты, которые включены, но строго не требуются. В списке, который расположен ниже, имеются описания для каждого пакета.

3.6.1 {{ pkg.name }}

Версия: {{ pkg.version }}

{{ pkg.description }}

[{{ pkg.url }}](#)

3.7 Опечатки и неточности

Проблему ошибок и опечаток в публикациях мы стараемся рассматривать и исправлять как можно оперативнее. Мы открыты к диалогу, и Вы, как читатель, всегда можете предложить свои замечания, улучшения и пожелания.

Для этого достаточно создать запрос в нашем официальном репозитории на GitHub по адресу: <https://github.com/Linux4Yourself/book/issues/new>.

Tip

Обязательно прикрепляйте к тексту сообщения место, в котором находится опечатка, и ссылку на нужную страницу с опечаткой или ошибкой.

4. Подготовительные работы

4.1 Целевые архитектуры

Основной целевой архитектурой является `x86_64` (64-разрядная).

С другой стороны, инструкции в этом руководстве также работают, с некоторыми модификациями, с PowerPC, ARM и AMD / Intel x86 (32-bit) процессорами.

Чтобы собрать систему, которая должна работать на одном из вышеуказанных процессоров, главным условием является существующая операционная система Linux, например, уже ранее собранная система по инструкциям этого руководства, Ubuntu, Red Hat/Fedora, SuSE, или другой дистрибутив, который нацелен на требуемую архитектуру.

Также обратите внимание, что 32-разрядный дистрибутив может быть установлен и использоваться как хост-система на 64-разрядном AMD / Intel компьютере.

4.1.1 О поддержке multilib

В этом руководстве присутствует частичная поддержка multilib.

Что такое multilib

Процессоры `x86_64` могут выполнять как скомпилированный для них код, так и скомпилированный для архитектуры `i386`. Но 32-битные исполняемые файлы работают только с 32-битными библиотеками (а 64-битные - только с 64-битными библиотеками), поэтому для запуска 32-битного исполняемого файла требуются 32-битные версии библиотек, которые он использует. Если в ОС присутствуют библиотеки для нескольких архитектур, её называют multilib системой.

Зачем это нужно

Некоторые программы с закрытым исходным кодом до сих пор имеют только 32-битные версии. Для Linux таких программ не много, а вот для Windows их существует огромное количество. А для того чтобы запустить их, необходим Wine с поддержкой multilib.

Как это реализовано в руководстве

В руководстве в конце многих страниц присутствуют инструкции для multilib систем. Поддержка multilib является опциональной. Если Вам она не нужна, не выполняйте эти инструкции. Поддержка multilib является частичной - инструкции для сборки 32-битных версий библиотек предоставляются только тогда, когда они необходимы для сборки пакета.

[Подробнее про архитектуры процессора.](#)

4.2 Требования к рабочей станции и операционной системе

4.2.1 Требования к аппаратному обеспечению

- Раздел на жёстком диске, рекомендуемый размер - более 20 Гб, так как для сборки пакетов необходимо много свободного места.
- Если оперативной памяти ПК мало (3 Гб и меньше), то рекомендуется создать раздел/файл подкачки. В крайнем случае используйте [zram](#).

4.2.2 Требования к программному обеспечению

Более ранние версии перечисленных программных пакетов могут работать, но корректность работы не проверялась.

- Bash-3.2 (`/bin/sh` - жесткая или символьическая ссылка на `bash`)
- Binutils-2.25
- Bison-2.7 (`/usr/bin/yacc` - символьическая ссылка на `bison` или на файл сценария, который его запускает)
- Bzip2-1.0.4
- Coreutils-6.9
- Diffutils-2.8.1
- Findutils-4.2.31
- Gawk-4.0.1 (`/usr/bin/awk` - символьическая ссылка на `gawk`)
- GCC-6.2 с компилятором C++, g++
- Glibc-2.11
- Grep-2.5.1a
- Gzip-1.3.12
- Linux Kernel-3.2
- M4-1.4.10
- Make-4.0
- Patch-2.5.4
- Perl-5.8.8
- Python-3.4
- Sed-4.1.5
- Tar-1.22
- Texinfo-4.7
- Xz-5.0.0

В зависимости от семейства ОС Linux, выполните следующий команды, чтобы обеспечить совместимость и установить необходимые пакеты:

Для Debian, Ubuntu:

```
1 apt-get install build-essential bison gawk texinfo
2 ln -sf bash /bin/sh
```

Для ArchLinux

```
1 pacman -S base-devel
```

Для Rosa

```
1  urpmi bison gawk texinfo make gcc-c++
```

4.2.3 Проверка соответствия программного обеспечения

Чтобы узнать, что ваша хост-система полностью соответствует всем необходимым для дальнейшей работы требованиям, выполните следующий набор команд:

filename

!> Внимательно изучите результат выполнения. В нём не должно встречаться строк, содержащих `ERROR, command not found, failed.`

Ошибочный результат

```
1  bash, version 5.0.17(1)-release
2  /bin/sh -> /usr/bin/dash
3  ERROR: /bin/sh does not point to bash
4  Binutils: version-check.sh: line 12: ld: command not found
5  version-check.sh: line 13: bison: command not found
6  yacc not found
7  bzip2, Version 1.0.8, 13-Jul-2019.
8  Coreutils: 8.32
9  diff (GNU diffutils) 3.7
10 find (GNU findutils) 4.7.0
11 version-check.sh: line 27: gawk: command not found
12 /usr/bin/awk -> /usr/bin/mawk
13 version-check.sh: line 37: gcc: command not found
14 version-check.sh: line 38: g++: command not found
15 (Ubuntu GLIBC 2.32-0ubuntu3) 2.32
16 grep (GNU grep) 3.4
17 gzip 1.10
18 Linux version 5.8.0-25-generic (buildd@lcy01-amd64-022) (gcc (Ubuntu 10.2.0-13ubuntu1) 10.2.0, GNU ld (GNU Binutils for Ubuntu) 2.35.1) #26-Ubuntu SMP Thu Oct 15 10:30:38 UTC 2020
19 version-check.sh: line 43: m4: command not found
20 version-check.sh: line 44: make: command not found
21 GNU patch 2.7.6
22 Perl version='5.30.3';
23 Python 3.8.6
24 sed (GNU sed) 4.7
25 tar (GNU tar) 1.30
26 version-check.sh: line 50: makeinfo: command not found
27 xz (XZ Utils) 5.2.4
28 version-check.sh: line 53: g++: command not found
29 g++ compilation failed
```

Успешный результат

```
1  bash, version 5.0.0(1)-release
2  /bin/sh -> /bin/bash
3  Binutils: (GNU Binutils) 2.32
4  bison (GNU Bison) 3.4.1
5  yacc is bison (GNU Bison) 3.4.1
6  bzip2, Version 1.0.8, 13-Jul-2019.
7  Coreutils: 8.31
8  diff (GNU diffutils) 3.7
9  find (GNU findutils) 4.6.0
10 GNU Awk 5.0.1, API: 2.0 (GNU MPFR 4.0.2, GNU MP 6.1.2)
11 /usr/bin/awk -> /usr/bin/gawk
12 gcc (GCC) 9.2.0
13 g++ (GCC) 9.2.0
14 (GNU Libc) 2.30
15 grep (GNU grep) 3.3
16 gzip 1.10
17 m4 (GNU M4) 1.4.18
18 GNU Make 4.2.1
19 GNU patch 2.7.6
20 Perl version='5.30.0';
21 Python 3.7.4
22 sed (GNU sed) 4.7
23 tar (GNU tar) 1.32
24 texi2any (GNU texinfo) 6.6
25 xz (XZ Utils) 5.2.4
26 g++ compilation OK
```

4.3 О времени сборки пакетов

Время сборки пакетов во многом зависит от мощности компьютера.

В этом руководстве используется единица времени, аналогичная такой же как в Linux from scratch - SBU (Standard Build Unit).

Она равна времени сборки первого пакета. Первым пакетом является `binutils`, время его компиляции в один поток на компьютере с процессором i3-2370M составляет примерно 3 минуты. 3 минуты = 1 SBU. На Вашем ПК значение будет немного другим. Также, если какой-то пакет компилируется 10 SBU, то, переводя SBU в более привычную единицу, это будет равняться 30 минутам (учитывая то, что 1 SBU = 3 минутам).

Время на сборку 32-битных версий библиотек не учитывается.

4.3.1 Самостоятельное измерение SBU

Для того чтобы вычислить время сборки определённого пакета, выполните:

```

1  TIMEFORMAT='%1R Elapsed Time - $PROGRAM'
2  tar -zvxf $PROGRAM.tar.gz
3  pushd $PROGRAM
4  { time \
5  {
6      ./configure &&
7      make &&
8      make install
9  }
10 } 2>&1 | tee ../build.log
11 popd

```

Замените `$PROGRAM.tar.gz` на имя нужного пакета, который надо распаковать.

4.3.2 Значения новых команд

Когда мы измеряем время, необходимое для сборки пакета, мы используем функцию `time`. Время, затраченное на распаковку архива, не учитывается. Фигурные скобки группируют команды как для функции определения затраченного времени, так и для перенаправления всего вывода в файл журнала (`../build.log`).

Note

SBU не даёт совсем точных значений, потому что эти значения зависят от многих факторов, включая версию GCC хост-системы. Они приведены здесь, чтобы *примерно* оценить, сколько времени может потребоваться для установки пакета.

4.4 Установка переменной \$LIN

В этом руководстве для обозначения пути к собираемой системе используется переменная \$LIN. Эта переменная должна указывать на каталог, в который будет смонтирован корень собираемой ОС. В принципе, он может быть любым, однако ниже, для примера, будет использован /mnt/lin.

Установим переменную:

```
1  export LIN=/mnt/lin
```

Создадим этот каталог:

```
1  mkdir -pv $LIN
```

Аргумент `-p` указывает создавать родительские директории в случае их отсутствия, а аргумент `-v` выводить сведения о выполнении.

⚠ Warning

Если вы вышли из командной оболочки или сменили пользователя, необходимо повторно задать эту переменную. Для удобства добавьте строку `export LIN=...` в `~/.bashrc` (пользователя, из-под которого будет производиться дальнейшая сборка кросс-компилятора и некоторых дополнительных пакетов временной системы). Это снимет с вас обязанность каждый раз объявлять переменную `LIN`.

⚡ Danger

Если данная переменная по какой-либо причине не была задана при выполнении дальнейших инструкций, вы можете повредить хост систему. Чтобы проверить, выполните `echo $LIN`. Если вывод будет таким: `/mnt/lin` - значит, всё в порядке.

4.5 Создание базовых директорий и символьических ссылок

4.5.1 Создание каталога \$LIN/tools

Все программы, которые будут скомпилированы в следующей части, будут установлены в каталог `$LIN/tools`, чтобы можно было оставить их отдельно от сборки конечной системы. Программы, которые будут скомпилированы - временные инструменты и не будут входить в конечную сборку системы.

После использования кросс-компилятора, от него можно избавиться. Использование каталога `$LIN/tools` необходимо для того, чтобы не засорять рабочие каталоги хост-системы.

Кросс-компилятор будет установлен в директорию `/tools`, создайте её:

```
1 mkdir -pv $LIN/tools
```

4.5.2 Создание иерархии файловой системы

Теперь создайте базовую иерархию файловой системы. Мы предлагаем вам использовать упрощённую структуру каталогов, когда директории `/bin`, `/sbin`, `/usr/sbin` являются ссылками на `/usr/bin`, а `/lib` - на `/usr/lib`.

Подобную структуру использует всё больше дистрибутивов: Fedora, Arch, Ubuntu и др. Если вы хотите создать упрощённую иерархию файловой системы, выполните следующие команды:

filename

Если вы хотите использовать "классическую" иерархию, в которой `/bin`, `/sbin`, `/usr/bin`, `/usr/sbin`, `/lib` и `/usr/lib` - разные каталоги, выполните следующие команды:

filename

Подобное поведение сделает возможным выносить `/usr` в отдельный раздел, либо загружать его по сети. Так же может повыситься безопасность и надёжность ОС. Однако потребуется выполнить множество дополнительных действий при установке пакетов. Если вы не знаете что выбрать - используйте вариант с упрощённой структурой - он лучше протестирован и требует меньше действий при сборке пакетов.

Подробнее об иерархии каталогов можно узнать в спецификации [FHS](#).

4.5.3 Для multilib

Создайте директорию для 32-битных библиотек:

```
1 mkdir -pv $LIN/usr/lib32
```

А также символьическую ссылку на неё:

```
1 ln -sv usr/lib32 $LIN/lib32
```



Tip

В случае если вы используете классическую иерархию каталогов, разделение `/lib32` и `/usr/lib32` не имеет смысла так как 32-битные библиотеки не требуются для запуска системы.



Warning

Обратите внимание, что указанная символьическая ссылка правильная. Если указать `$LIN/usr/lib32`, то при входе в среду `chroot`, ссылка будет некорректная.

4.6 Требуемые пакеты и патчи

На данном этапе всё готово для загрузки необходимых пакетов и патчей.

Самый простой способ загрузки - воспользоваться файлом [wget-list](#) с копиями файлов, расположенных на нашем сервере.

Можно также воспользоваться файлом [wget-list.orig](#), который содержит аналогичные ссылки, но с ресурсов репозиториев и хранилищ разработчиков этих пакетов.

Note

Во всех случаях используются обычные версии пакетов, идентичные находящимся на официальных сайтах их разработчиков, однако они были загружены на наш сервер во избежание проблем с загрузкой.

Загрузите файл `wget-list` или `wget-list.orig` и передайте в параметр `--input-file` путь до файла программе `wget`.

```
1 wget --input-file=wget-list --continue --directory-prefix=$LIN/usr/src
```

Если вы загружали файл `wget-list.orig`, то замените ключ `--input-file=wget-list` на `--input-file=wget-list.orig`.

При желании вы можете выполнить проверку пакетов на соответствие контрольным суммам.

Загрузите файл [md5Sums](#).

Поместите его в каталог `$LIN/usr/src` и выполните команду:

```
1 pushd $LIN/usr/src
2 md5sum -c md5sums
3 popd
```

4.7 Создание пользователя LIN

Когда мы находимся в системе под пользователем `root`, одна единственная ошибка может привести к повреждению или поломке всей хост-системы.

Следовательно, рекомендуется выполнять сборку пакетов для временного набора инструментов от обычного пользователя, без привилегий.

Вы можете использовать произвольного пользователя, но, для упрощения настройки чистого рабочего окружения, создайте нового пользователя с именем `lin` как члена группы `lin` и используйте этого пользователя на время всего процесса установки временного набора инструментов.

```
1 groupadd lin
2 useradd -s /bin/bash -g lin -m -k /dev/null lin
```

Значение параметров командной строки:

`-s /bin/bash` Устанавливает `bash` оболочкой по умолчанию для пользователя `lin`.

`-g lin` Опция добавляет пользователя `lin` в созданную группу `lin`.

`-m` Создает домашний каталог для пользователя `lin`.

`-k /dev/null` Этот параметр предотвращает возможное копирование файлов из предустановленного набора каталогов (по умолчанию `/etc/skel`), изменив местоположение ввода на специальное `null` устройство. `/dev/null` — специальный файл в системах класса UNIX, представляющий собой так называемое «пустое устройство». Запись в него происходит успешно, независимо от объема «записанной» информации. Чтение из `/dev/null` эквивалентно считыванию конца файла (`EOF`).

`lin` Это имя созданного пользователя.

При желании можно создать пароль для этой учётной записи:

```
1 passwd lin
```

Предоставим пользователю `lin` полный доступ к каталогам будущей ОС:

```
1 chown -vR lin $LIN
```

Выполните вход как пользователь `lin`. Это действие можно выполнить в графической оболочке, используя виртуальный терминал, или в обычной пользовательской среде:

```
1 su - lin
```

4.8 Настройка окружения

Необходимо настроить окружение для недавно созданного пользователя. Создадим `.bash_profile`:

```
1 cat > ~/.bash_profile << "EOF"
2 exec env -i HOME=$HOME TERM=$TERM PS1='\u:\w\$ ' /bin/bash
3 EOF
```

Создадим базовый `.bashrc`:

```
1 cat > ~/.bashrc << "EOF"
2 set +h
3 umask 022
4 LIN=/mnt/lin
5 LC_ALL=C
6 LIN_TGT=$(uname -m)-lin-linux-gnu
7 PATH=/usr/bin
8 if [ ! -L /bin ]; then PATH=/bin:$PATH; fi
9 PATH=$LIN/tools/bin:$PATH
10 export LIN LC_ALL LIN_TGT PATH CONFIG_SITE
11 EOF
```

4.8.1 CFLAGS и CXXFLAGS

Использовать оптимизацию на данном этапе не стоит, однако вы можете добавить флаг `-s`, чтобы сразу после сборки автоматически удалялись ненужные и отладочные символы, а также `-O2` во избежание ошибки сборки glibc:

```
1 echo "export CFLAGS=\"-s -O2\" " >> ~/.bashrc
2 echo "export CXXFLAGS=\"-s -O2\" " >> ~/.bashrc
```

Это может сэкономить несколько гигабайт места на диске.

4.8.2 Bash-completion

Если вы используете программу `bash-completion`, то можете добавить её поддержку для пользователя `lin`:

```
1 echo " /etc/bash_completion" >> ~/.bashrc
```

4.8.3 MAKEFLAGS

Для экономии времени на многоядерном процессоре используйте параллельную сборку. Чтобы её включить, надо добавить для `make` переменную `-jN`, где `N` - число потоков вашего процессора. Это можно сделать двумя способами:

- Указывать при каждом вызове `make` аргумент `-jN`
- Добавить переменную окружения `MAKEFLAGS`

Замените `N` на число потоков вашего процессора.

О дополнительной информации о потоках процессора читайте [здесь](#).

4.8.4 Для multilib

Для `multilib` выполните:

```
1 echo "export LIN_TGT32=i686-lin-linux-gnu" >> ~/.bashrc
```

Эта переменная используется для сборки i386 библиотек.

4.8.5 Применение изменений

Для того чтобы применить изменения, выполните:

```
1  source ~/bash_profile
```

4.8.6 Значения параметров базового bashrc

`set +h` - Данный параметр отключает сохранение путей к исполняемым файлам в памяти bash. Это необходимо для того чтобы новые исполняемые файлы становились доступны немедленно.

`umask 022` - Гарантирует, что для новых файлов будут установлены права 644.

`LIN=/mnt/lin` - Задает путь к корню собираемой системы. `mnt/lin` взят в качестве образца.

`LC_ALL=C` - Исключает связанные с локализацией ошибки.

`PATH=/usr/bin if [! -L /bin]; then PATH=/bin:$PATH; fi` - Задаёт пути поиска исполняемых файлов в хост-системе.

`PATH=$LIN/tools/bin:$PATH` - Необходимо для обнаружения исполняемых файлов кросс-компилятора.

4.9 О тестах

Большинство пакетов предоставляет наборы тестов для проверки их работоспособности после сборки.

Выполнить тесты может быть хорошей идеей, поскольку это позволит проверить, что все компоненты были скомпилированы корректно. Успешное прохождение пакетом всех тестов обычно гарантирует, что пакет будет работать именно так, как задумано разработчиком. Если тесты провалены, значит в пакете может содержаться ошибка.

Запуск тестов необязателен, но рекомендуется, если вам важна надежность системы. Некоторые тесты занимают очень много времени, например, тесты для GCC продолжаются около сотни SBU.

SBU для пакетов будет указан без учета тестов. При кросс-компиляции тесты могут не работать, и выполнять их для пакетов временной системы не рекомендуется.

4.10 О приоритете

В этом руководстве будет определён приоритет пакетов. Он может быть следующим:

4.10.1 Необходимый

Невозможно обойтись без данного пакета, хотя в некоторых случаях можно заменить альтернативой. Внимательно отнеситесь к сборке таких пакетов.

4.10.2 Важный

Теоретически, вы можете обойтись без этого пакета, но вам придется сильно отклоняться от инструкций, и у вас могут возникнуть проблемы.

4.10.3 Необязательный

Сборку данного пакета можно пропустить, и это не повлияет на работоспособность других пакетов, однако может повлиять на пакеты за пределами Core и, может быть, Extra книги.

4.11 Общая инструкция по сборке пакетов

Сборка всех пакетов в этом руководстве выполняется по следующему алгоритму:

1. Загрузка исходного кода - можно сделать непосредственно перед сборкой пакета, а можно воспользоваться `wget-list` и сразу загрузить все файлы.
2. Распаковка исходных кодов. Используйте команду `tar -xf архив.tar.xz`.
3. Переход в директорию с недавно распакованным исходным кодом. Используйте `cd`. Чаще всего, название директории соответствует названию архива с пакетом без расширения `tar.{gz,xz,bzip2}`.
4. Выполнение инструкции.
5. Выход из директории с исходным кодом.
6. Удаление директории с исходным кодом (данный шаг обязателен при сборке временной системы).

Больше информации о сборке пакетов вы можете найти [здесь](#).

4.12 О потоках

Сборочная система `make` может разбивать сборку на несколько потоков, число которых регулируется флагом `-j`. Рекомендуется использовать число потоков равное количеству потоков процессора или чуть больше. Чтобы параллельную сборку, нужно добавить для `make` переменную `-jN`, где `N` - число потоков вашего процессора. Это можно сделать двумя способами:

- Указывать при каждом вызове `make` аргумент `-jN`
- Добавить переменную окружения `MAKEFLAGS`

Для того, чтобы узнать число потоков вашего ЦП, выполните:

```
1 lscpu | grep "CPU(s):"
```

Результат выполнения будет примерно следующий. Значения будут отличаться в зависимости от характеристик вашего процессора:

```
1 CPU(s):          4
2 NUMA node0 CPU(s): 0-3
```

`CPU(s)` - кол-во потоков.

При желании добавьте соответствующую переменную окружения, заменив `N` на нужное значение:

```
1 echo "MAKEFLAGS=\"-jN\" " >> ~/.bashrc
```

5. Сборка кросс-компилятора

5.1 Сборка кросс-компилятора

Эта глава описывает процесс создания кросс-компилятора и минимально необходимого набора инструментов, который потребуется для сборки конечной системы.

Все пакеты, которые будут скомпилированы в этой главе, будут установлены в каталог `$LIN/tools`, чтобы хранить их отдельно от хост-системы и конечной системы. Как обсуждалось ранее, этот набор инструментов - временный, и нам не требуется его наличие в конечном счёте.

!> При работе следует строго выполнять последовательность сборки пакетов.

Подробнее про [кросс-компилятор](#).

5.2 Binutils

5.2.1 Настройка

!> Пакет binutils должен быть установлен раньше GCC и libc.

Пакет Binutils требует использовать отдельную директорию для сборки. Создайте её:

```
1  mkdir -v build
2  cd      build
```

?> 1 SBU равен времени сборки данного пакета.

Запустим скрипт `configure`:

```
1  ./configure --prefix=$LIN/tools      \
2    --with-sysroot=$LIN                \
3    --target=$LIN_TGT                 \
4    --disable-nls                     \
5    --disable-werror
```

Для multilib

Добавьте параметр `--enable-multilib`

Значения параметров

`--with-sysroot=$LIN --target=$LIN_TGT` - необходимо для кросс-компиляции.

`--disable-nls` - Для кросс-компилятора не требуется локализация.

`--disable-werror` - отключает остановку сборки при предупреждениях.

5.2.2 Сборка

```
1  make
```

5.2.3 Установка

```
1  make install -j1
```

`-j1` предотвращает возможную ошибку установки.

Дополнительную информацию о компиляции пакетов смотрите [здесь](#).

5.3 GCC

5.3.1 Подготовка

Дополнительные необходимые файлы

```
{{ mpc.url}}
{{ gmp.url}}
{{ mpfr.url}}
{{ isl.url}}
```

Распакуйте дополнительные пакеты:

```
tar -xf ..{{ mpfr.fileName }}
mv -v {{ mpfr.name }}-{{ mpfr.version }} {{ mpfr.name }}
tar -xf ..{{ gmp.fileName }}
mv -v {{ gmp.name }}-{{ gmp.version }} {{ gmp.name }}
tar -xf ..{{ mpc.fileName }}
mv -v {{ mpc.name }}-{{ mpc.version }} {{ mpc.name }}
tar -xf ..{{ isl.fileName }}
mv -v {{ isl.name }}-{{ isl.version }} {{ isl.name }}
```

!> Обратите внимание, что распаковка указанных пакетов должна производиться из каталога пакета GCC.

Смените пути установки библиотек:

```
1 sed -e '/m64=/s/Lib64/Lib/' \
2   -e '/m32=/s/m32=.*/Lib32$(call if_multiarch,:i386-Linux-gnu)/* \
3     -i.orig gcc/config/i386/t-linux64
```

Пакет GCC требует использовать отдельную директорию для сборки. Создайте её:

```
1 mkdir -v build
2 cd      build
```

5.3.2 Настройка

```
1 ./configure \
2   --target=$LIN_TGT \
3   --prefix=$LIN/tools \
4   --with-glibc-version=2.11 \
5   --with-sysroot=$LIN \
6   --with-newlib \
7   --without-headers \
8   --enable-initfini-array \
9   --disable-nls \
10  --disable-shared \
11  --disable-decimal-float \
12  --disable-threads \
13  --disable-libatomic \
14  --disable-libomp \
15  --disable-libquadmath \
16  --disable-libssp \
17  --disable-libvtv \
18  --disable-libstdcxx \
19  --enable-languages=c,c++ \
20  --disable-multilib
```

Для multilib

замените параметр `--disable-multilib` на `--enable-multilib --with-multilib-list=m64,m32`

Значения параметров

--with-glibc-version = 2.11 Эта опция гарантирует, что пакет будет совместим с версией glibc на хосте. Для него установлено минимальное требование glibc, указанное в Требованиях к хост-системе.

--with-newlib Поскольку рабочая библиотека C еще недоступна, это гарантирует, что константа `__init__libc` определена при сборке libgcc. Это предотвращает компиляцию любого кода, требующего поддержки libc.

--without-headers При создании полного кросс-компилятора GCC требует стандартных заголовков, совместимых с целевой системой. Для наших целей эти заголовки не понадобятся. Этот переключатель предотвращает их поиск GCC.

--enable-initfini-array Этот переключатель заставляет использовать некоторые внутренние структуры данных, которые необходимы, но не могут быть обнаружены при построении кросс-компилятора.

--disable-shared Этот переключатель заставляет GCC связывать свои внутренние библиотеки статически. Нам это нужно, потому что общие библиотеки требуют glibc, которая ещё не установлена

--disable-decimal-float, --disable-threads, --disable-libatomic, --disable-libgomp, --disable-libquadmath, --disable-libssp, --disable-libvtv, --disable-libstdc++ Эти переключатели отключают поддержку десятичных расширений с плавающей запятой, потоковой передачи, libatomic, libgomp, libquadmath, libssp, libvtv и стандартной библиотеки C++ соответственно. Эти функции не будут скомпилированы при сборке кросс-компилятора и не являются необходимыми для кросс-компиляции временной libc.

--enable-languages Эта опция гарантирует, что будут построены только компиляторы C и C++. Это единственные языки, которые нужны сейчас.

5.3.3 Сборка

```
1 make
```

5.3.4 Установка

```
1 make install
```

Создадим полную версию `limits.h` - заголовочного файла, в котором записаны лимиты:

```
1 cd ..
2 cat gcc/limitx.h gcc/glimits.h gcc/limity.h > \
3   `dirname ${LIN_TGT-gcc -print-libgcc-file-name}`/install-tools/include/limits.h
```

6. Заголовочные файлы ядра Linux

Ссылка для скачивания: [{{ package.url }}](#)

Текущая версия: [{{ package.version }}](#)

Домашняя страница: [{{ package.homeUrl }}](#)

Важность: **Необходимый**

Размер архива: [{{ package.size }}](#) Mb

Файлы заголовков определяют способ определения функций в исходном файле. Они используются таким образом, чтобы компилятор мог проверить правильность использования функции в качестве сигнатуры функции (возвращаемое значение и параметры) в файле заголовка. Для этой задачи фактическая реализация функции не требуется.

Когда вы компилируете драйвер устройства как модуль ядра, вам необходимы установленные заголовочные файлы ядра. Также они требуются, если вы собираете пользовательское приложение, которое взаимодействует напрямую с ядром.

!> Версия заголовочных файлов должна соответствовать версии устанавливаемого ядра.

6.1 Сборка

?> **Данный пакет является частью архива с ядром Linux.**

Убедитесь, что в архив не включены файлы которые могут помешать сборке.

```
1 make mrproper
```

Эта команда выполнит очистку дерева исходных текстов. Разработчики ядра рекомендуют, чтобы эта команда выполнялась перед каждым процессом компиляции.

6.1.1 Подготовьте заголовки для использования:

```
1 make headers
```

6.2 Установка

```
1 find usr/include -name '*' -delete
2 rm usr/include/Makefile
3 cp -rv usr/include $LIN/usr
```

!> Заголовочные файлы, расположенные в системном каталоге `/usr/include`, должны всегда быть те, которые использовались при компиляции [Glibc](#). Их никогда не следует заменять на чистые заголовочные файлы ядра или любые другие подготовленные заголовочные файлы.

6.2.1 Установленные файлы

Данный пакет устанавливает множество заголовочных файлов, в частности `/usr/include/asm/*.h`, `/usr/include/asm-generic/*.h`, `/usr/include/drm/*.h`, `/usr/include/linux/*.h`, `/usr/include/misc/*.h`, `/usr/include/mtd/*.h`, `/usr/include/rdma/*.h`, `/usr/include/scsi/*.h`, `/usr/include/sound/*.h`, `/usr/include/video/*.h`, and `/usr/include/xen/*.h`

7. Сборка временной системы

7.1 Сборка временной системы

В этой главе будет выполняться сборка пакетов, используя ранее созданный **кросс-компилятор**. Хотя пакеты и будут установлены на своё окончательное место, но их пока нельзя будет использовать, и основные инструменты будут задействованы из хост-системы. Тем не менее, установленные библиотеки используются при компоновке.

Использование установленных пакетов станет возможным в следующей главе после входа в среду `chroot`. Поэтому их необходимо собрать прежде, чем мы это сделаем.

Главная задача - отделить этот инструментарий от хост-системы, чтобы он работал независимо, и свести влияние хост-системы на процесс сборки к минимуму.

!> Следует обратить особое внимание, что все инструкции в разделе должны выполняться от имени пользователя `lin`, который был создан ранее. Внимательно проверьте настройку окружения данного пользователя, убедитесь, что выполнены инструкции, указанные в главе [Настройка окружения](#).

?> Все установленные здесь пакеты будут переустановлены впоследствии.

7.2 glibc

7.2.1 Дополнительные необходимые файлы

`{{ patch.url }}`

7.2.2 Настройка

Необходимо создать две символические ссылки:

```
1  ln -sfv ..;/lib/ld-linux-x86-64.so.2 $LIN/Lib64
2  ln -sfv ..;/lib/ld-linux-x86-64.so.2 $LIN/lib64/ld-lsb-x86-64.so.3
```

Первая ссылка используется GCC, вторую требует LSB.

В пакете Glibc по умолчанию используется несоответствующая стандарту FHS директория `/var/db`. Для исправления этого примените патч:

```
1  patch -Np1 -i ..;/glibc-2.33-fhs-1.patch
```

Пакет Glibc требует использовать отдельную директорию для сборки. Создайте её:

```
1  mkdir build
2  cd build
```

Запустите скрипт `configure`:

```
1  ./configure
2      --prefix=/usr
3      --host=$LIN_TGT
4      --build=$(..;/scripts/config.guess)
5      --enable-kernel=3.2
6      --with-headers=$LIN/usr/include
7      Libc_cv_libdir=/lib
8      Libc_cv_include_x86_isa_level=no
9      --disable-nscd
10     --disable-timezone-tools
```

Для multilib

Добавьте параметр `--enable-multi-arch`

Значения параметров

`--host=$LIN_TGT, --build=$(..;/scripts/config.guess)` - необходимо для кросс-компиляции.

`--enable-kernel=3.2` - оптимизирует glibc для использования с ядрами новее 3.2.

`--with-headers=$LIN/usr/include` - задает путь к заголовкам ядра.

`Libc_cv_include_x86_isa_level=no` - исключает возможную ошибку.

`--disable-nscd, --disable-timezone-tools` - демон nscd и инструменты для управления часовыми поясами не нужны для временной glibc.

7.2.3 Сборка

```
1  make
```

7.2.4 Установка

```
1 make DESTDIR=$LIN install
```

Завершите установку файла `limits.h`, запустив скрипт из состава GCC:

```
1 $LIN/tools/libexec/gcc/$LIN_TGT/11.2.0/install-tools/mkheaders
```

7.2.5 Тестирование

!> На данном этапе необходимо убедиться, что установленные ранее пакеты работают правильно. Внимательно изучите результаты вывода команд, и проверьте, что они строго соответствуют результатам вывода, приведенным ниже. Если есть несоответствия, значит инструкции на предыдущих этапах были выполнены некорректно.

Чтобы проверить правильность работы кросс-компилятора и libc, выполните:

```
1 echo 'int main(){}' > dummy.c
2 $LIN_TGT-gcc dummy.c
3 readelf -l a.out | grep '/ld-Linux'
```

ВЫВОД ДОЛЖЕН БЫТЬ ТАКИМ:

```
1 [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
```

ЕСЛИ ВСЕ ХОРОШО, УДАЛИТЕ НЕИЗУМНЫЕ ФАЙЛЫ:

```
1 rm -v dummy.c a.out
```

7.2.6 Для multilib

Для MultiLib требуется установить 32-битную версию glibc. Для этого, удалите оставшиеся файлы от 64-битной сборки glibc:

Настройка

```
1 rm -rf ./*
2 find .. -name "*_a" -delete
```

Запустите скрипт `configure`:

```
1 CC="$LIN_TGT-gcc -m32"
2 CXX="$LIN_TGT-g++ -m32"
3 ./configure
4   --prefix=/usr
5   --host=$LIN_TGT32
6   --build=$(./scripts/config.guess)
7   --enable-kernel=3.2
8   --with-headers=$LIN/usr/include
9   --enable-multi-arch
10  --libdir=/usr/lib32
11  --libexecdir=/usr/lib32
12  libc_cv_slibdir=/lib32
13  libc_cv_include_x86_isa_level=no
14  --disable-nscd
15  --disable-timezone-tools
```

Сборка

```
1 make
```

Установка

Установите 32-битные библиотеки из этого пакета:

```
1 make DESTDIR=$PWD/DESTDIR install
2 cp -a DESTDIR/lib32/* $LIN/lib32/
3 cp -a DESTDIR/usr/lib32 $LIN/usr/
4 install -vm64 DESTDIR/usr/include/gnu/{lib-names,stubs}-32.h \
```

```
5      $LIN/usr/include/gnu/
6  ln -svf ../Lib32/ld-Linux.so.2 $LIN/Lib/ld-Linux.so.2
```

Проверка работоспособности

Чтобы проверить работоспособность 32-битной glibc, выполните:

```
1  echo 'int main(){}' > dummy.c
2  $LIN_TGT-gcc -m32 dummy.c
3  readelf -l a.out | grep '/ld-linux'
```

Вывод должен быть таким:

```
1  [Requesting program interpreter: /lib/ld-linux.so.2]
```

Если всё хорошо, удалите ненужные файлы:

```
1  rm -v dummy.c a.out
```

7.3 LibstdC++ Проход 1

Пакет содержит библиотеку времени исполнения, необходимую программам, написанным на языке C++ и собранным при помощи компилятора GNU.

Версия `v{{ package.version }}`

Ссылка для скачивания: [{{ package.url }}](#)

Текущая версия: `{{ package.version }}`

Домашняя страница: [{{ package.homeUrl }}](#)

Важность: **Необходимый**

Размер архива: `{{ package.size }} Mb`

SBU: **1**

7.3.1 Настройка

!> Данный пакет входит в архив с исходным кодом пакета GCC

Создайте отдельную директорию для сборки:

```
1  mkdir -v build
2  cd      build
```

Запустите скрипт `configure`

```
1  ./libstdc++-v3/configure
2  --host=$LIN_TGT
3  --build=$(..../config.guess)
4  --prefix=/usr
5  --disable-multilib
6  --disable-nls
7  --disable-libstdcxx-pch
8  --with-gxx-include-dir=/tools/$LIN_TGT/include/c++/11.2.0
```

Значения параметров

`--host=$LIN_TGT --build=$(..../config.guess)` - необходимо для кросс-компиляции

`--disable-multilib` - 32-битная версия libstdc не нужна на данном этапе

`--disable-libstdcxx-pch` - отключает установку предварительно скомпилированных заголовков, ненужных на данном этапе

`--with-gxx-include-dir=/tools/$LIN_TGT/include/c++/11.2.0` - путь поиска заголовков C++

7.3.2 Сборка

```
1  make
```

7.3.3 Установка

```
1  make DESTDIR=$LIN install
```

7.4 m4

7.4.1 Настройка

```
1 ./configure --prefix=/usr \
2   --host=$LIN_TGT \
3   --build=$(build-aux/config.guess)
```

7.4.2 Сборка

```
1 make
```

7.4.3 Установка

```
1 make DESTDIR=$LIN install
```

7.5 ncurses

7.5.1 Настройка

Убедитесь, что `gawk` будет найден первым:

```
1 sed -i s/mawk// configure
```

Для установки `ncurses` требуется программа `tic`. Соберите её:

```
1 mkdir tic-build
2 cd tic-build
3 ./configure
4 make -C include
5 make -C progs tic
6 cd ..
```

Запустите скрипт `configure`:

```
1 ./configure --prefix=/usr
2           --host=$LIN_TGT
3           --build=$./config.guess
4           --without-manpages
5           --without-tests
6           --without-cxx
7           --with-shared
8           --without-debug
9           --without-ada
10          --without-normal
11          --enable-widec
```

Значения параметров

`--without-manpages`, `--without-tests`, `--without-cxx` - Man-страницы, тесты и библиотека C++ не нужны для временной системы.

`--without-ada` - отключает сборку компонентов на языке `ada`, так как в собираемой системе отсутствуют необходимые для их запуска компоненты.

`--without-normal` - отключает установку большинства статических библиотек.

`--enable-widec` - включает установку библиотек с поддержкой многобайтовых символов.

7.5.2 Сборка

```
1 make
```

7.5.3 Установка

```
1 make DESTDIR=$LIN TIC_PATH=$(pwd)/tic-build/progs/tic install
2 echo "INPUT(-lncursesw)" > $LIN/usr/lib/libncurses.so
```

7.5.4 При раздельной структуре каталогов

Переместите разделяемые библиотеки в `$LIN/lib`:

```
1 mv -v $LIN/usr/lib/libncursesw.so.* $LIN/lib
```

Поскольку библиотеки были перемещены, одна символьическая ссылка указывает на несуществующий файл. Исправьте это:

```
1 ln -svf ../../lib/$(readlink $LIN/usr/lib/libncursesw.so) $LIN/usr/lib/libncursesw.so
```

7.5.5 Для multilib

Настройка

Соберите 32-битную версию ncurses: Выполните:

```
1  make distclean
```

Чтобы очистить директорию от файлов предыдущей сборки.

Запустите скрипт `configure`:

```
1  CC="$LIN_TGT-gcc -m32"
2  CXX="$LIN_TGT-g++ -m32"
3  ./configure --prefix=/usr
4    --host=$LIN_TGT32
5    --build=$LIN_TGT32
6    --libdir=/usr/lib32
7    --without-manpages
8    --without-tests
9    --without-cxx
10   --without-progs
11   --with-shared
12   --without-debug
13   --without-ada
14   --without-normal
15   --enable-pc-files
16   --enable-widec
17   --with-pkg-config-libdir=/usr/lib32/pkgconfig
```

Сборка

```
1  make
```

Установка

```
1  make DESTDIR=$PWD/DESTDIR TIC_PATH=$(pwd)/tic-build/progs/tic install
2  ln -s libcursesw.so DESTDIR/usr/lib32/libcursesw.so
3  cp -Rv DESTDIR/usr/lib32/* $LIN/usr/lib32
4  rm -rf DESTDIR
```

7.6 bash

Настройка

Запустите скрипт `configure`:

```
1 ./configure --prefix=/usr
2     --build=$(support/config.guess) \
3     --host=$LIN_TGT
4     --without-bash-malloc
```

ЗНАЧЕНИЯ ПАРАМЕТРОВ

`--without-bash-malloc` - этот параметр отключает использование функции выделения памяти (malloc) Bash, которая вызывает ошибки сегментации. Отключив эту опцию, Bash будет использовать функции malloc из libc, которые более стабильны.

Сборка

```
1 make
```

7.6.1 Установка

```
1 make DESTDIR=$LIN install
```

Сделайте символьическую ссылку для программ, которые используют `sh` в качестве интерпретатора:

```
1 ln -sv bash $LIN/bin/sh
```

7.6.2 При раздельной структуре каталогов

Переместите `bash` в нужную директорию:

```
1 mv $LIN/usr/bin/bash $LIN/bin/bash
```

7.7 coreutils

7.7.1 Настройка

Запустите скрипт `configure`:

```
1 ./configure --prefix=/usr
2   --host=$LIN_TGT
3   --build=$(build-aux/config.guess)
4   --enable-install-program=hostname
5   --enable-no-install-program=kill,uptime
6   --disable-nts
```

Значения параметров

`--enable-install-program=hostname` - включает установку программы `hostname`, нужной некоторым пакетам.

`--enable-no-install-program=kill,uptime` - программы `kill` и `uptime` предоставляются другими пакетами.

7.7.2 Сборка

```
1 make
```

7.7.3 Установка

```
1 make DESTDIR=$LIN install
```

7.7.4 При раздельной структуре каталогов

Переместите программы в их окончательные ожидаемые места. Хотя в этой временной среде в этом нет необходимости, но вам нужно это сделать, потому что некоторые программы используют фиксированное расположение исполняемых файлов:

```
1 mv -v $LIN/usr/bin/{cat,chggrp,chmod,chown,cp,date,dd,df,echo} $LIN/bin
2 mv -v $LIN/usr/bin/{false,ln,ls,mkdir,mknod,mv,pwd,rm} $LIN/bin
3 mv -v $LIN/usr/bin/{rmdir,stty,sync,true,uname} $LIN/bin
4 mv -v $LIN/usr/bin/{head,nice,sleep,touch} $LIN/bin
5 mv -v $LIN/usr/bin/chroot $LIN/usr/sbin
6 mkdir -pv $LIN/usr/share/man/man8
7 mv -v $LIN/usr/share/man/man1/chroot.1 $LIN/usr/share/man/man8/chroot.8
8 sed -i 's/"1"/"8"/' $LIN/usr/share/man/man8/chroot.8
```

7.8 diffutils

7.8.1 Настройка

Запустите скрипт `configure`:

```
1 ./configure --prefix=/usr --host=$LIN_TGT --disable-nls
```

7.8.2 Сборка

```
1 make
```

7.8.3 Установка

```
1 make DESTDIR=$LIN install
```

7.9 file

7.9.1 Подготовка

На хост-системе должна присутствовать утилита File. Вы можете также собрать её следующим образом:

```
1  mkdir build
2  pushd build
3  ./configure --disable-bzlib \
4    --disable-libseccomp \
5    --disable-xzlib \
6    --disable-zlib
7  make
8  popd
```

7.9.2 Настройка

```
1  ./configure --prefix=/usr --host=$LIN_TGT --build=$(./config.guess)
```

7.9.3 Сборка

```
1  make FILE_COMPILE=$(pwd)/build/src/file
```

7.9.4 Установка

```
1  make DESTDIR=$LIN install
```

7.10 findutils

7.10.1 Настройка

```
1 ./configure --prefix=/usr --host=$LIN_TGT --disable-nls
```

7.10.2 Сборка

```
1 make
```

7.10.3 Установка

```
1 make DESTDIR=$LIN install
```

7.10.4 При раздельной структуре каталогов

Переместите исполняемый файл в нужную директорию:

```
1 mv -v $LIN/usr/bin/find $LIN/bin
2 sed -i 's|find:=${BINDIR}|find:=/bin|' $LIN/usr/bin/updatedb
```

7.11 gawk

7.11.1 Настройка

Убедимся, что лишние файлы не будут установлены:

```
1 sed -i 's/extras//' Makefile.in
```

Выполните скрипт `configure`:

```
1 ./configure --prefix=/usr \
2     --host=$LIN_TGT \
3     --build=$(./config.guess) --disable-nls
```

7.11.2 Сборка

```
1 make
```

7.11.3 Установка

```
1 make DESTDIR=$LIN install
```

7.12 grep

7.12.1 Настройка

```
1 ./configure --prefix=/usr \
2           --host=$LIN_TGT \
3           --bindir=/bin \
4           --disable-nls
```

7.12.2 Сборка

```
1 make
```

7.12.3 Установка

```
1 make DESTDIR=$LIN install
```

7.13 gzip

7.13.1 Настройка

```
1 ./configure --prefix=/usr --host=$LIN_TGT
```

7.13.2 Сборка

```
1 make
```

7.13.3 Установка

```
1 make DESTDIR=$LIN install
```

7.13.4 При раздельной структуре каталогов

Переместите исполняемый файл в нужную директорию:

```
1 mv -v $LIN/usr/bin/gzip $LIN/bin
```

7.14 make

7.14.1 Настройка

```
1 ./configure --prefix=/usr \
2           --without-guile \
3           --host=$LIN_TGT \
4           --build=$(build-aux/config.guess) --disable-nls
```

Значения параметров

--without-guile - если не указать этот параметр в процессе настройки, `guile` будет задействован с хост-системы. Так как мы выполняем кросс-компиляцию, следует выставить этот параметр.

7.14.2 Сборка

```
1 make
```

7.14.3 Установка

```
1 make DESTDIR=$LIN install
```

7.15 patch

7.15.1 Настройка

```
1 ./configure --prefix=/usr \
2           --host=$LIN_TGT \
3           --build=$(build-aux/config.guess)
```

7.15.2 Сборка

```
1 make
```

7.15.3 Установка

```
1 make DESTDIR=$LIN install
```

7.16 sed

7.16.1 Настройка

```
1 ./configure --prefix=/usr \
2           --host=$LIN_TGT \
3           --disable-nls \
4           --bindir=/bin
```

7.16.2 Сборка

```
1 make
```

7.16.3 Установка

```
1 make DESTDIR=$LIN install
```

7.17 tar

7.17.1 Настройка

```
1 ./configure --prefix=/usr
2           --host=$LIN_TGT
3           --build=$(build-aux/config.guess) \
4           --disable-nls --disable-acl \
5           --bindir=/bin
```

7.17.2 Сборка

```
1 make
```

7.17.3 Установка

```
1 make DESTDIR=$LIN install
```

7.18 xz

7.18.1 Настройка

```
1 ./configure --prefix=/usr --host=$LIN_TGT \
2     --build=$build \
3     --disable-static \
4     --disable-nls \
5     --disable-doc
```

7.18.2 Сборка

```
1 make
```

7.18.3 Установка

```
1 make DESTDIR=$LIN install
```

7.18.4 При раздельной структуре каталогов

Переместите часть файлов в нужные директории:

```
1 mv -v $LIN/usr/bin/{lzma,unlzma,lzcat,xz,unxz,xzcat} $LIN/bin
2 mv -v $LIN/usr/lib/liblzma.so.* $LIN/lib
3 ln -svf ../../lib/$(readlink $LIN/usr/lib/liblzma.so) $LIN/usr/lib/liblzma.so
```

7.19 binutils проход 2

7.19.1 Подготовка

Пакет Binutils требует использовать отдельную директорию для сборки. Создайте её:

```
1  mkdir -v build
2  cd      build
```

7.19.2 Настройка

```
1  ./configure
2  --prefix=/usr
3  --build=$(./config.guess) \
4  --host=$LIN_TGT
5  --disable-nls
6  --enable-shared
7  --disable-werror
8  --enable-64-bit-bfd
```

Для multilib

Добавьте параметр `--enable-multilib`

7.19.3 Сборка

```
1  make
```

7.19.4 Установка

```
1  make DESTDIR=$LIN install -j1
2  install -vm755 libctf/.libs/libctf.so.0.0.0 $LIN/usr/lib
```

7.20 gcc проход 2

7.20.1 Подготовка

Дополнительные необходимые файлы

```
{{ mpc.url}}
{{ gmp.url}}
{{ mpfr.url}}
{{ isl.url}}
{{ patch.url }}
```

Распакуйте дополнительные пакеты:

!> Обратите внимание, что распаковка указанных пакетов должна производится из каталога пакета GCC.

```
tar -xf ../{{ mpfr.fileName }}
mv -v {{ mpfr.name }}-{{ mpfr.version }} {{ mpfr.name }}
tar -xf ../{{ gmp.fileName }}
mv -v {{ gmp.name }}-{{ gmp.version }} {{ gmp.name }}
tar -xf ../{{ mpc.fileName }}
mv -v {{ mpc.name }}-{{ mpc.version }} {{ mpc.name }}
tar -xf ../{{ isl.fileName }}
mv -v {{ isl.name }}-{{ isl.version }} {{ isl.name }}
```

Смените пути установки библиотек:

```
1 sed -e '/m64=/s@/lib64@/lib@' \
2   -e '/m32=/s@/m32=.*@/lib32$(call if_multiarch,:i386-linux-gnu)@' \
3   -i.orig gcc/config/i386/t-linux64
```

Пакет GCC требует использовать отдельную директорию для сборки. Создайте её:

```
1 mkdir -v build
2 cd      build
```

Разрешим сборку libgcc с поддержкой многопоточности:

```
1 mkdir -pv $LIN_TGT/libgcc
2 ln -s ../../libgcc/gthr-posix.h $LIN_TGT/libgcc/gthr-default.h
```

Настройка:

```
1 ./configure \
2   --build=$(..//config.guess) \
3   --host=$LIN_TGT \
4   --prefix=/usr \
5   CC_FOR_TARGET=$LIN_TGT-gcc \
6   --with-build-sysroot=$LIN \
7   --enable-initfini-array \
8   --disable-nls \
9   --disable-decimal-float \
10  --disable-libatomic \
11  --disable-Libomp \
12  --disable-Libquadmath \
13  --disable-Libssp \
14  --disable-Libvtv \
15  --disable-libstdcxx \
16  --enable-languages=c,c++ \
17  --disable-multilib
```

Для multilib

замените параметр `--disable-multilib` на `--enable-multilib --with-multilib-list=m64,m32`

Значения параметров

--enable-initfini-array - параметр заставляет использовать некоторые внутренние структуры данных, которые необходимы, но не могут быть обнаружены при построении кросс-компилиатора.

--disable-decimal-float, --disable-threads, --disable-libatomic, --disable-libgomp, --disable-libquadmath, --disable-libssp, --disable-libvtv, --disable-libstdcxx - параметр отключают поддержку десятичных расширений с плавающей запятой, потоковой передачи, libatomic, libgomp, libquadmath, libssp, libvtv и стандартной библиотеки C++ соответственно. Эти функции не будут скомпилированы при сборке кросс-компилиатора и не являются необходимыми для кросс-компиляции временной libc.

--enable-languages [] - опция включает поддержку компиляторов C и C++. Это единственные языки, которые нужны сейчас.

7.20.2 Сборка

```
1 make
```

7.20.3 Установка

```
1 make DESTDIR=$LIN install
```

Некоторые программы используют команду `cc`, а не `gcc`. Создайте символьическую ссылку на `gcc`:

```
1 ln -sv gcc $LIN/usr/bin/cc
```

7.21 Смена владельца для каталога \$LIN

Теперь, когда все циклические зависимости были разрешены, мы сможем использовать среду `chroot`, чтобы максимально изолироваться от хост-системы, которая сейчас предоставляет инструменты для сборки. В конечном счёте, мы будем использовать только работающее ядро хост-системы. Остальные компоненты будут задействованы из среды `chroot`.

!> Все дальнейшие инструкции должны выполняться от пользователя `root`.

!> Проверьте, что переменная окружения `$LIN` установлена:

```
1 echo $LIN
```

В настоящее время вся иерархия каталогов в `$LIN` принадлежит пользователю `lin`, который существует только в хост-системе.

Если владельца оставить как есть, то в новой системе права будут принадлежать идентификатору пользователя без соответствующей учетной записи. Это опасно, поскольку какая-либо новая созданная учетная запись может получить этот идентификатор, что приведёт к полному доступу данного пользователя ко всей системе, что может быть нежелательно и небезопасно.

Сменим владельца на пользователя `root`:

```
1 chown -R root:root $LIN
```

После этих действий можно производить дальнейшие действия.

7.22 Вход в окружение chroot

Для того чтобы изолироваться от хост-системы, необходимо войти в среду `chroot`.

Создайте два файла в директории `/dev`:

```
1  mknod -m 600 $LIN/dev/console c 5 1
2  mknod -m 666 $LIN/dev/null c 1 3
```

!> После перезагрузки следует выполнить заново действия, идущие далее.

Смонтируйте виртуальные файловые системы ядра:

```
1  mount -v --bind /dev $LIN/dev
2  mount -v --bind /dev/pts $LIN/dev/pts
3  mount -vt proc proc $LIN/proc
4  mount -vt sysfs sysfs $LIN/sys
5  mount -vt tmpfs tmpfs $LIN/run
```

В некоторых хост-системах `/dev/shm` - ссылка на `/run/shm`. Если это так в вашей системе, выполните:

```
1  if [ -h $LIN/dev/shm ]; then
2    mkdir -pv $LIN/$(readlink $LIN/dev/shm)
3  fi
```

Войдите в `chroot`:

`filename`

?> После входа в `chroot`, в приглашении `bash` будет написано `I have no name!`. Это нормально, и ничего плохого в этом нет. Дело в том, что файл `/etc/passwd` ещё не создан.

`filename`

?> Сейчас выполнять действия по выходу из `chroot` не нужно, это понадобится после окончания сборки системы.

7.23 Создание основных файлов и символьических ссылок

Исторически, в файле `/etc/mtab` записывалась информация о смонтированных разделах. Создайте символьическую ссылку для совместимости:

```
1 ln -sv /proc/self/mounts /etc/mtab
```

Некоторым пакетам может понадобиться файл `/etc/hosts`:

```
1 cat > /etc/hosts << EOF
2 "127.0.0.1 localhost $(hostname)"
3 ::1 localhost
4 EOF
```

Создайте файл списка пользовательских учётных записей `/etc/passwd`:

```
1 cat > /etc/passwd << "EOF"
2 root:x:0:root:/root:/bin/bash
3 bin:x:1:bin:/dev/null:/bin/false
4 daemon:x:6:Daemon User:/dev/null:/bin/false
5 messagebus:x:18:18:D-Bus Message Daemon User:/run/dbus:/bin/false
6 uid:x:80:80:UUID Generation Daemon User:/dev/null:/bin/false
7 nobody:x:99:99:Unprivileged User:/dev/null:/bin/false
8 EOF
```

Создайте файл имён групп и членов каждой группы:

```
1 cat > /etc/group << "EOF"
2 root:x:0:
3 bin:x:1:daemon
4 sys:x:2:
5 kmem:x:3:
6 tape:x:4:
7 tty:x:5:
8 daemon:x:6:
9 floppy:x:7:
10 disk:x:8:
11 lp:x:9:
12 dialout:x:10:
13 audio:x:11:
14 video:x:12:
15 utmp:x:13:
16 usb:x:14:
17 cdrom:x:15:
18 adm:x:16:
19 messagebus:x:18:
20 input:x:24:
21 mail:x:34:
22 kvm:x:61:
23 uidd:x:80:
24 wheel:x:97:
25 nogroup:x:99:
26 users:x:999:
27 EOF
```

Для проведения тестов некоторых пакетов вы можете добавить пользователя `tester`:

```
1 echo "tester:x:101:101::/home/tester:/bin/bash" >> /etc/passwd
2 echo "tester:x:101:" >> /etc/group
3 install -o tester -d /home/tester
```

Чтобы убрать надпись `I have no name!` в приглашении `bash` выполните:

```
1 bash --login +h
```

Некоторые программы могут записывать информацию о входах в систему в журнал, но для этого требуется создать специальные файлы:

```
1 touch /var/log/{btmp,lastlog,faillog,wtmp}
2 chgrp -v utmp /var/log/lastlog
3 chmod -v 664 /var/log/lastlog
4 chmod -v 600 /var/log/btmp
```

7.24 LibstdC++ проход 2

Пакет содержит библиотеку времени исполнения, необходимую программам, написанным на языке C++ и собранным при помощи компилятора GNU.

Ссылка для скачивания: [{{ package.url }}](#)

Текущая версия: [{{ package.version }}](#)

Домашняя страница: [{{ package.homeUrl }}](#)

Важность: **Необходимый**

Размер архива: [{{ package.size }}](#) Mb

SBU: 1

7.24.1 Настройка

!> **Данный пакет входит в архив с исходниками GCC.**

Создайте необходимую символьическую ссылку:

```
1 ln -s gthr-posix.h libgcc/gthr-default.h
```

Создайте отдельную директорию для сборки:

```
1 mkdir -v build
2 cd      build
```

Запустите скрипт `configure`

```
1 ./libstdc++-v3/configure
2   CXXFLAGS="-O2 -s -D_GNU_SOURCE" \
3   --prefix=/usr \
4   --disable-nls \
5   --host=$(uname -m)-lin-linux-gnu \
6   --disable-libstdcxx-pch --disable-multilib
```

Для multilib

Замените `--disable-multilib` на `--enable-multilib`.

Значения параметров

`--disable-libstdcxx-pch` - отключает установку предварительно скомпилированных заголовков, ненужных на данном этапе

`--host=$(uname -m)-lin-linux-gnu` - Libstdc++ должна быть собрана с такими же параметрами, что и GCC

7.24.2 Сборка

```
1 make
```

7.24.3 Установка

```
1 make install
```

7.25 gettext

7.25.1 Настройка

```
1 ./configure --disable-static
```

Значения параметров

`--disable-static` - так как это временный инструмент, то не требуется наличие общих библиотек, поэтому и нет необходимости их создавать.

7.25.2 Сборка

```
1 make
```

7.25.3 Установка

```
1 make DESTDIR=/usr install
```

7.26 bison

7.26.1 Настройка

```
1 ./configure --prefix=/usr
```

7.26.2 Сборка

```
1 make
```

7.26.3 Установка

```
1 make install
```

7.27 perl

7.27.1 Настройка

```
1 sh Configure -des
2           -Dprefix=/usr
3           -Dvendorprefix=/usr
4           -Dprivlib=/usr/lib/perl5/5.34/core_perl
5           -Darchlib=/usr/lib/perl5/5.34/core_perl
6           -Dsitelib=/usr/lib/perl5/5.34/site_perl
7           -Dsitearch=/usr/lib/perl5/5.34/site_perl
8           -Dvendorlib=/usr/lib/perl5/5.34/vendor_perl
9           -Dvendorarch=/usr/lib/perl5/5.34/vendor_perl
```

Значения параметров

-des - это комбинация трёх опций:

-d - использует значения по умолчанию для всех элементов,

-e - обеспечивает выполнение всех задач,

-s - отключает вывод лишней информации.

7.27.2 Сборка

```
1 make
```

7.27.3 Установка

```
1 make install
```

7.28 python

7.28.1 Настройка

```
1 ./configure --prefix=/usr --enable-shared \
2      --without-ensurepip
```

Значения параметров

--enable-shared - включает установку динамических библиотек;
--without-ensurepip - менеджер пакетов `pip` не нужен на данном этапе.

7.28.2 Сборка

```
1 make
```

7.28.3 Установка

```
1 make install
```

7.29 texinfo

7.29.1 Настройка

```
1 ./configure --prefix=/usr
```

?> Во время процесса настройки выполняется тест, который указывает на ошибку TestXS_la-TestXS.lo. Её можно игнорировать.

7.29.2 Сборка

```
1 make
```

7.29.3 Установка

```
1 make install
```

7.30 util-linux

7.30.1 Настройка

В FHS рекомендуется использовать директорию `/var/lib/hwclock` для файла `adjtime`. Создайте её:

```
1 mkdir -pv /var/lib/hwclock
```

Запустите скрипт `configure`:

```
1 ./configure ADJTIME_PATH=/var/lib/hwclock/adjtime \
2   --disable-chfn-chsh \
3   --disable-login \
4   --disable-nologin \
5   --disable-su \
6   --disable-setpriv \
7   --disable-runuser \
8   --disable-pylibmount \
9   --disable-static \
10  --without-python \
11  runstatedir=/run
```

Значения параметров

`--disable-*` - отключает программы, которые предоставляются другими пакетами

`--without-python` - отключает сборку ненужных привязок python.

7.30.2 Сборка

```
1 make
```

7.30.3 Установка

```
1 make install
```

7.30.4 Для multilib

Необходимо собрать 32-битные библиотеки из состава этого пакета:

Очистка

```
1 make distclean
```

Настройка

Запустите скрипт `configure`:

```
1 CC="gcc -m32" \
2 ./configure ADJTIME_PATH=/var/lib/hwclock/adjtime \
3   --enable-usrdir-path \
4   --host=i686-pc-linux-gnu \
5   --libdir=/usr/lib32 \
6   --disable-bash-completion \
7   --disable-chfn-chsh \
8   --disable-fdisks \
9   --disable-fsck \
10  --disable-login \
11  --disable-mount \
12  --disable-nologin \
13  --disable-pylibmount \
14  --disable-runuser \
15  --disable-schedutils \
16  --disable-setpriv \
17  --disable-static
```

```
18      --disable-su
19      --without-python  \
```

Значение параметров:

--disable-* - позволяет сэкономить время, отключив сборку ненужных компонентов.

Сборка

```
1  make
```

Установка

```
1  make DESTDIR=$PWD/DESTDIR install
2  cp -Rv DESTDIR/usr/lib32/* /usr/lib32
3  rm -rf DESTDIR
```

7.31 Очистка и сохранение временной системы

Файлы `libtool` с расширением `.la` могут мешать линковке с динамическими библиотеками. Удалите их:

```
1  find /usr/{lib{,32},libexec} -name \*.la -delete
```

Удалите документацию временных инструментов:

```
1  rm -rf /usr/share/{info,man,doc}/*
```

7.31.1 Выход из среды chroot

[filename](#)

7.31.2 Удаление отладочных символов

Созданные исполняемые файлы и библиотеки содержат немногим более 90 МБ ненужных отладочных символов.

Удалите символы отладки из двоичных файлов:

```
1  strip --strip-debug $LIN/usr/lib/*
2  strip --strip-unneeded $LIN/usr/{,s}bin/*
3  strip --strip-unneeded $LIN/tools/bin/*
```

7.31.3 Сохранение

При желании сохраните временную систему в архив:

```
1  cd $LIN &&
2  tar -cJpf $HOME/lin-temp-tools.tar.xz .
```

Это может понадобиться в том случае, если нужно собрать новую систему, либо восстановить уже собранную в случае поломки. Сократит вам время, поскольку потребуется только распаковать временный инструментарий.

7.31.4 Восстановление

Выполняется из-под хост-системы:

```
1  cd $LIN &&
2  rm -rf ./* &&
3  tar -xpf $HOME/lin-temp-tools.tar.xz
```

8. Сборка Linux системы

8.1 Сборка конечной Linux системы

В этой главе мы приступаем к сборке конечной Linux системы.

 **Warning**

Прежде чем приступить к работе, проверьте, что выполнен [вход в окружение chroot](#).

8.2 Настройка окружения bash

Оболочка командной строки `bash` использует множество стартовых скриптов. С их помощью можно задать различные переменные, функции, алиасы, настройки `bash`. Ниже будут предложены команды по созданию минимальных версий стартовых скриптов `bash`:

8.2.1 /etc/profile

вы могли заметить файлы `~/.profile`, `~/.bash_profile` и ряд других. Они используются для задания элементов окружения для оболочки пользователя. Например, `umask`, а также переменных `PS1` или `PATH`. То же самое и с файлом `/etc/profile`, только он используется для задания общесистемных параметров, а файлы в каталоге пользователя - для задания параметров конкретного пользователя системы.

[filename](#)

O CFLAGS и CXXFLAGS

С помощью данных переменных можно задать флаги компилятора, влияющие в том числе на оптимизацию. Есть следующие уровни оптимизации (и соответствующие флаги)

- `-00` - без оптимизации. Не рекомендуется.
- `-01` - простейшие минимальные оптимизации. Не рекомендуется.
- `-02` - стандартная оптимизация. По умолчанию в GCC.
- `-03` - агрессивная оптимизация, почти все пакеты собираются и работают, рекомендуется нами.
- `-0fast` - чрезвычайно агрессивная оптимизация, некоторые пакеты не собираются.
- `-0s` - оптимизация по размеру.
- `-0g` - оптимизация для отладки.

вы можете сообщить компилятору оптимизировать код для вашего процессора добавив опцию `-march=native`, однако возможность запуска такого кода на других процессорах будет потеряна.

8.2.2 Bash-completion

`bash-completion` расширяет существующие возможности дополнения в `bash`, позволяя вводить сложные командные строки нажатием буквально нескольких клавиш. Целью является создание программируемых процедур дополнения для большинства распространённых команд Linux/UNIX, которые позволили бы системным администраторам и программистам сократить количество нажатий клавиш при выполнении ежедневных задач.

[filename](#)

8.2.3 Dircolors

Этот файл нужен для цветного вывода таких утилит, как `ls`. Пример: `ls --color=auto`.

[filename](#)

8.2.4 Readline

Этот файл нужен для настройки `inputrc`. Если пользователь не имеет индивидуальных установок, он использует глобальный файл.

[filename](#)

8.2.5 Umask

Установка значения `umask` важна для безопасности. Здесь права доступа для записи группы по умолчанию выключены для пользователей системы и когда имена пользователя и группы не совпадают.

`filename`

8.2.6 Другие стартовые скрипты

`filename`

8.2.7 Применить изменения

```
1  bash --login +h
```

8.3 iana-etc

8.3.1 Установка

Установленные файлы

`/etc/protocols` и `/etc/services`

Краткое описание

`/etc/protocols` - определяет, каким транспортным протоколом пользуется сервис. Имеется возможность различия протоколов udp или tcp. Сервис может работать с разными протоколами, а может быть два сервиса работают на одном порте, но с разными протоколами.

и `/etc/services` - файл определений протоколов. Данный файл является простым ASCII файлом, описывающим различные DARPA internet протоколы, которые доступны через подсистему TCP/IP. Не изменяйте этот файл, так как изменения могут привести к некорректному формированию IP пакетов. Номера протоколов и их имена определяются Центром Сетевой Информации (DDN Network Information Center).

8.4 glibc

8.4.1 Дополнительные необходимые файлы

```
{{ glibcPatch.url}}
```

```
{{ tzdata.url}}
```

8.4.2 Подготовка

Исправьте обнаруженную проблему безопасности:

```
1 sed -e '/NOTIFY_REMOVED/s// \&& data.attr != NULL/' \
2     -i sysdeps/unix/sysv/linux/mq_notify.c
```

Glibc по умолчанию использует несоответствующую стандарту FHS директорию `/var/db`. Для соответствия с FHS примените патч:

```
1 patch -Np1 -i ../glibc-2.33-fhs-1.patch
```

Исправьте ошибку:

```
1 sed -e '402a\\      *result = local->data.services[database_index];' \
2     -i nss/nss_database.c
```

Пакет Glibc требует использовать отдельную директорию для сборки. Создайте её:

```
1 mkdir -v build
2 cd      build
```

8.4.3 Настройка

```
1 ./configure
2     --prefix=/usr
3     --disable-werror
4     --enable-kernel=3.2
5     --with-headers=/usr/include
6     --libexecdir=/usr/lib
7     libc_cv_slibdir=/usr/lib
8     libc_cv_include_x86_isa_level=no
```

Если вы используете раздельную структуру каталогов, то измените значение параметра `libc_cv_slibdir` на `/lib`: `libc_cv_slibdir=/lib`, и удалите параметр `--libexecdir=/usr/lib`.

Для multilib

Добавьте параметр `--enable-multi-arch`

Значения параметров

`--enable-kernel=3.2` - оптимизирует glibc для использования с ядрами новее 3.2.

`--with-headers=/usr/include` - задаёт путь к заголовкам ядра.

`libc_cv_include_x86_isa_level=no` - исключает возможную ошибку.

8.4.4 Сборка

```
1 make
```

8.4.5 Тестирование

Создайте необходимую для тестирования ссылку (**только если вы собираете систему с раздельными каталогами!**):

```
1 ln -sfnv $PWD/elf/ld-linux-x86-64.so.2 /lib
```

Вы можете запустить тесты, выполнив:

```
1 make check
```

Известно, что следующие тесты могут дать сбой:

- `io/tst_lchmod`, `misc/tst_ttynname` выдают ошибку в среде LX4;
- `elf/tst-cpu-features-cpuinfo` может дать сбой на некоторых архитектурах;
- `nss/tst-nss-files-hosts-multi` может потерпеть неудачу по пока неизвестным причинам;
- `rt/tst-cputimer{1,2,3}` сильно зависит от версии ядра. На старых ядрах (4.14.91-4.14.96, 4.19.13-4.19.18, и 4.20.0-4.20.5) известно, что тесты выдают ошибку;
- Математические тесты иногда терпят неудачу при работе на системах, где ЦП не является относительно новым процессором Intel или AMD.

8.4.6 Установка

Удалите предупреждение при установке и отключите запуск ненужных проверок:

```
1 touch /etc/ld.so.conf
2 sed '/test-installation/s@$(PERL)@echo not running@' -i ..\Makefile
```

Установите пакет и файлы конфигурации для демона `nscd`

```
1 make install
2 cp -v ..\nscd\nscd.conf /etc/nscd.conf
3 mkdir -pv /var/cache/nscd
```

Если вы собираетесь использовать `systemd`, установите соответствующие файлы для демона `nscd`:

```
1 install -v -Dm644 ..\nscd\nscd.tmpfiles /usr/lib/tmpfiles.d/nscd.conf
2 install -v -Dm644 ..\nscd\nscd.service /lib/systemd/system/nscd.service
```

Установка локали

Разные страны и культуры имеют различные соглашения для коммуникаций. Эти соглашения состоят как из очень простых, таких как форматы даты и времени, так и из более сложных, таких как разговорный язык. "Интернационализация" программ GNU работает с помощью локалей (locales).

Чтобы установить локали, выполните:

```
1 mkdir -pv /usr/lib/locale
2 localedef -i POSIX -f UTF-8 C.UTF-8 2> /dev/null || true
3 localedef -i en_US -f UTF-8 en_US.UTF-8
```

Чтобы установить локаль для языка, на котором вы планируете использовать систему, выберите из списка необходимую и выполните комманду:

```
1 localedef -i cs_CZ -f UTF-8 cs_CZ.UTF-8
2 localedef -i de_DE -f ISO-8859-1 de_DE
3 localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
4 localedef -i de_DE -f UTF-8 de_DE.UTF-8
5 localedef -i el_GR -f ISO-8859-7 el_GR
6 localedef -i en_GB -f ISO-8859-1 en_GB
7 localedef -i en_GB -f UTF-8 en_GB.UTF-8
8 localedef -i en_HK -f ISO-8859-1 en_HK
9 localedef -i en_PH -f ISO-8859-1 en_PH
10 localedef -i en_US -f ISO-8859-1 en_US
11 localedef -i en_US -f UTF-8 en_US.UTF-8
12 localedef -i es_ES -f ISO-8859-15 es_ES@euro
13 localedef -i es_MX -f ISO-8859-1 es_MX
```

```

14 localedef -i fa_IR -f UTF-8 fa_IR
15 localedef -i fr_FR -f ISO-8859-1 fr_FR
16 localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
17 localedef -i fr_FR -f UTF-8 fr_FR.UTF-8
18 localedef -i is_IS -f ISO-8859-1 is_IS
19 localedef -i is_IS -f UTF-8 is_IS.UTF-8
20 localedef -i it_IT -f ISO-8859-1 it_IT
21 localedef -i it_IT -f ISO-8859-15 it_IT@euro
22 localedef -i it_IT -f UTF-8 it_IT.UTF-8
23 localedef -i ja_JP -f EUC-JP ja_JP
24 localedef -i ja_JP -f SHIFT_JIS ja_JP.SIJS 2> /dev/null || true
25 localedef -i ja_JP -f UTF-8 ja_JP.UTF-8
26 localedef -i nl_NL@euro -f ISO-8859-15 nl_NL@euro
27 localedef -i ru_RU -f KOI8-R ru_RU.KOI8-R
28 localedef -i ru_RU -f UTF-8 ru_RU.UTF-8
29 localedef -i se_NO -f UTF-8 se_NO.UTF-8
30 localedef -i ta_IN -f UTF-8 ta_IN.UTF-8
31 localedef -i tr_TR -f UTF-8 tr_TR.UTF-8
32 localedef -i zh_CN -f GB18030 zh_CN.GB18030
33 localedef -i zh_HK -f BIG5-HKSCS zh_HK.BIG5-HKSCS
34 localedef -i zh_TW -f UTF-8 zh_TW.UTF-8

```

Вы можете установить все локали, которые содержатся в файле {{ package.fileName }}/localesdata/SUPPORTED . Выполните следующую команду:

```
1 make localesdata/install-locales
```

8.4.7 Настройка

nsswitch.conf

Создайте nsswitch.conf :

```

1 cat > /etc/nsswitch.conf << "EOF"
2 # Begin /etc/nsswitch.conf
3
4 passwd: files
5 group: files
6 shadow: files
7
8 hosts: files dns
9 networks: files
10
11 protocols: files
12 services: files
13 ethers: files
14 rpc: files
15
16 # End /etc/nsswitch.conf
17 EOF

```

Установка tzdata

Установите информацию о часовых поясах:

```

tar -xf ../../{{tzdata.fileName}}
ZONEINFO=/usr/share/zoneinfo
mkdir -p $ZONEINFO/{posix,right}

for tz in etcetera southamerica northamerica europe africa antarctica \
         asia australasia backward; do
    zic -L /dev/null -d $ZONEINFO ${tz}
    zic -L /dev/null -d $ZONEINFO/posix ${tz}
    zic -L leapseconds -d $ZONEINFO/right ${tz}
done

cp -v zone.tab zone1970.tab iso3166.tab $ZONEINFO
zic -d $ZONEINFO -p America/New_York
unset ZONEINFO

```

Для выбора часового пояса запустите скрипт:

```
1 tzselect
```

Для сохранения выбранного часового пояса выполните:

```
1  ln -sfv /usr/share/zoneinfo/< xxx> /etc/localtime
```

Где < xxx> - путь к вашему часовому поясу.

ld.so.conf

Создайте файл /etc/ld.so.conf :

```
1  cat > /etc/ld.so.conf << "EOF"
2  # Begin /etc/ld.so.conf
3  /usr/local/lib
4  /opt/lib
5  include /etc/ld.so.conf.d/*.conf
6
7  EOF
8  mkdir -pv /etc/ld.so.conf.d
```

Обновите кэш библиотек:

```
1  ldconfig
```

8.4.8 Для multilib

Подготовка

Для multilib требуется установить 32-битную версию glibc. Удалите оставшиеся от 64-битной сборки glibc файлы:

```
1  rm -rf /*.a
2  find .. -name /*.a" -delete
```

Настройка

```
1  CC="gcc -m32" CXX="g++ -m32" \
2  ./configure \
3  --prefix=/usr \
4  --host=i686-pc-linux-gnu \
5  --build=$(../.scripts/config.guess) \
6  --enable-kernel=3.2 \
7  --with-headers=/usr/include \
8  --enable-multi-arch \
9  --libdir=/usr/lib32 \
10  --libexecdir=/usr/lib32 \
11  libc_cv_slibdir=/usr/lib32
```

8.4.9 Сборка

```
1  make
```

8.4.10 Установка

```
1  make DESTDIR=$PWD/DESTDIR install
2  cp -a DESTDIR/lib32/* /lib32/
3  cp -a DESTDIR/usr/lib32/* /usr/lib32/
4  install -vm64 DESTDIR/usr/include/gnu/{lib-names,stubs}-32.h \
5  /usr/include/gnu/
6  ln -sfv ..//lib32/ld-linux.so.2 /lib/ld-linux.so.2
```

Добавьте запись в ld.so.conf :

```
1  echo "/usr/lib32" >> /etc/ld.so.conf
```

Обновите кэш библиотек:

```
1  ldconfig
```

8.4.11 Установленные файлы

Программы: catchsegv, gencat, getconf, getent, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, makedb, mtrace, nsqd, pcprofiledump, pldd, sln, sotruss, sprof, tzselect, xtrace, zdump, zic

Библиотеки: ld-2.33.so, libBrokenLocale.{a,so}, libSegFault.so, libanl.{a,so}, libc.{a,so}, libc_nonshared.a, libcrypt.{a,so}, libdl.{a,so}, libg.a, libm.{a,so}, libmcheck.a, libmemusage.so, libmvec.{a,so}, libnsl.{a,so}, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss hesiod.so, libpcprofile.so, libpthread.{a,so}, libpthread_nonshared.a, libresolv.{a,so}, librt.{a,so}, libthread_db.so, libutil.{a,so}

Директории: /usr/include/arpa, /usr/include/bits, /usr/include/gnu, /usr/include/net, /usr/include/netash, /usr/include/netatalk, /usr/include/netax25, /usr/include/neteconet, /usr/include/netinet, /usr/include/netipx, /usr/include/netiucv, /usr/include/netpacket, /usr/include/netrom, /usr/include/netrose, /usr/include/nfs, /usr/include/protocols, /usr/include/rpc, /usr/include/sys, /usr/lib/audit, /usr/lib/gconv, /usr/lib/locale, /usr/libexec/getconf, /usr/share/i18n, /usr/share/zoneinfo, /var/cache/nsqd, /var/lib/nss_db

Краткое описание

`catchsegv` - Может использоваться для создания трассировки стека, когда программа завершается с ошибкой сегментации.

`gencat` - Создаёт каталоги сообщений.

`getconf` - Отображает значения конфигурации системы для специфичных переменных файловой системы.

`getent` - Получает записи из административной базы данных.

`iconv` - Выполняет преобразование набора символов в другую кодировку.

`iconvconfig` - Создает ускоренную загрузку iconv модулей файлов конфигурации.

`ldconfig` - Настраивает привязки динамического компоновщика.

`ldd` - помогает определить список разделяемых библиотек (shared libraries), от которых зависит программа.

`lddlibc4` - Помогает ldd с объектными файлами.

`locale` - Отображает всевозможную информацию о текущей локали.

`localedef` - Компилирует спецификации локали.

`makedb` - Создает простую базу данных из текстового ввода.

`mtrace` - Читает и интерпретирует файл трассировки памяти и отображает сводку в удобочитаемом формате.

`nsqd` - Служба (демон), которая предоставляет кэш для наиболее общих запросов службы имен.

`pldd` - Список динамических общих объектов, используемых запущенными процессами.

`sln` - Статически слинкованная программа `ln`.

`sotruss` - Выполняет трассировку вызовов процедуры разделяемой библиотеки для указанной команды.

`sprof` - Считывает и отображает данные профилирования общих объектов.

`tzselect` - Выясняет у пользователя его текущее местоположение и выводит описание часового пояса на устройство стандартного вывода.

`xtrace` - Трассировка выполняемой программы, и выводит в реальном времени на устройство стандартного вывода выполняемые функции.

`zdump` Распечатывает текущее время для каждого часового пояса, указанного в командной строке.

`zic` - Компилятор часовых поясов.

`ld.so` - Программа выполняет поиск и загружает динамические библиотеки, необходимые программам, а также подготавливает программы к запуску и запускают их.

`libBrokenLocale` - Используется внутри Glibc как грубый хак, чтобы обработать запущенную сломанную программу (например некоторые приложения Motif). Изучите комментарии в файле `locale/broken_cur_max.c` для получения более подробной информации.

`libSegFault` - Обработчик сигнала ошибки сегментации, используемый `catchsegv`.

`libanl` - Асинхронная библиотека поиска имен.

`libc` - Стандартная библиотека языка Си.

`libcrypt` - Криптографическая библиотека.

`libdl` - Интерфейс библиотеки динамической линковки.

`libg` - библиотека-заглушка, не содержащая функций. Раньше была библиотекой выполнения для `g++`.

`libm` - Математическая библиотека.

`libmcheck` - включает проверку распределения памяти при линковке.

`libmemusage` - Используется программой `memusage` чтобы помочь собрать информацию об использовании памяти в программе.

`libnsl` - библиотека сетевых сервисов.

`libnss` - Библиотеки коммутаторов имен, содержащие функции для разрешение имен хостов, имен пользователей, имен групп, псевдонимов, служб, протоколов и т.д.

`libpthread` - POSIX библиотека потоков.

`libresolv` - Содержит функции для создания, отправки и интерпретации пакетов на серверы доменных имен в Интернете.

`librt` - Содержит функции, обеспечивающие большую часть указанных интерфейсов в POSIX.1b расширении.

`libthread_db` - Содержит функции, полезные для построения отладчиков для многопоточных программ.

`libutil` - Содержит ключи для «стандартных» функций, используемых в большинстве различных утилит Unix.

8.5 zlib-ng

8.5.1 Настройка

[filename](#)

Значения параметров

`--zlib-compat` - включает полную совместимость с API `zlib`.

`--native` - использовать все доступные для вашего процессора оптимизации.

8.5.2 Сборка

[filename](#)

8.5.3 Тестирование

[filename](#)

8.5.4 Установка

[filename](#)

Удалите ненужную статическую библиотеку:

[filename](#)

8.5.5 При раздельной структуре каталогов

[filename](#)

8.5.6 Для multilib

Очистка

[filename](#)

Настройка

[filename](#)

Сборка

[filename](#)

Установка

[filename](#)

Удалите ненужную статическую библиотеку:

[filename](#)

8.5.7 Установленные файлы

Библиотеки: libz.so

Краткое описание

libz.so - Содержит функции сжатия, используемые многими программами

8.6 bzip2

8.6.1 Дополнительные необходимые файлы

[{{ patch.url }}](#)

8.6.2 Подготовка

Примените патч для правильной установки документации:

[filename](#)

Убедитесь, что будут созданы относительные символические ссылки, и исправьте путь установки man-страниц:

[filename](#)

8.6.3 Сборка

Для сборки динамической библиотеки `libbz2.so` и самого пакета:

[filename](#)

8.6.4 Установка

[filename](#)

Создайте необходимые символические ссылки:

[filename](#)

Если вы собираете систему с раздельной структурой каталогов, то пропустите шаг с созданием ссылок, перейдя к следующему.

8.6.5 При раздельной структуре каталогов

[filename](#)

8.6.6 Для multilib

Очистка

[filename](#)

Сборка

[filename](#)

Установка

[filename](#)

8.6.7 Установленные файлы

Программы: `bunzip2` (ссылка на `bzip2`), `bzcat` (ссылка на `bzip2`), `bzcmp` (ссылка на `bzdiff`), `bzdiff`, `bzegrep` (ссылка на `bzgrep`), `bzfgrep` (ссылка на `bzgrep`), `bzgrep`, `bzip2`, `bzip2recover`, `bzless` (ссылка на `bzmore`), и `bzmore`

Библиотеки: `libbz2.so`

Краткое описание

`bunzip2` - распаковывает файлы в формате `bzip`

`bzcat` - распаковывает в стандартный вывод

`bzcmp` - запускает `cmp` для файлов, сжатых с помощью `bzip`

`bzdiff` - запускает `diff` для файлов, сжатых с помощью `bzip`

`Bzgrep` - запускает `egrep` для файлов, сжатых с помощью `bzip`

`bzfgrep` - запускает `fgrep` для файлов, сжатых с помощью `bzip`

`bzgrep` - запускает `grep` для файлов, сжатых с помощью `bzip`

`bzip2` - сжимает файлы, используя алгоритм сжатия текста сортировки блоков Барроуза-Уиллера с кодированием Хаффмана; степень сжатия лучше, чем достигается более традиционными компрессорами, использующими алгоритмы «Lempel-Ziv», такие как `gzip`

`bzip2recover` - пытается восстановить данные из поврежденных `bzip`-файлов

`bzless` - работает меньше с файлами, сжатыми с помощью `bzip`

`bzmore` - работает больше с файлами, сжатыми с помощью `bzip`

`libbz2` - библиотека, реализующая сжатие данных без потерь с сортировкой по блокам с использованием алгоритма Барроуза-Уиллера.

8.7 xz

8.7.1 Настройка

[filename](#)

8.7.2 Сборка

[filename](#)

8.7.3 Тестирование

[filename](#)

8.7.4 Установка

[filename](#)

8.7.5 При раздельной структуре каталогов

[filename](#)

8.7.6 Для multilib

Очистка

[filename](#)

Настройка

[filename](#)

Сборка

[filename](#)

Установка

[filename](#)

8.7.7 Установленные файлы

Программы: lzcat (ссылка на xz), lzcmp (ссылка на xzdiff), lzdiff (ссылка на xzdiff), lzegrep (ссылка на xzgrep), lzfgrep (ссылка на xzgrep), lzgrep (ссылка на xzgrep), lzless (ссылка на xzless), lzma (ссылка на xz), lzmadec, lzmainfo, lzmore (ссылка на xzmore), unlzma (ссылка на xz), unxz (ссылка на xz), xz, xzcat (ссылка на xz), xzcmp (ссылка на xzdiff), xzdec, xzdiff, xzegrep (ссылка на xzgrep), xzfgrep (ссылка на xzgrep), xzgrep, xzless и xzmore

Библиотеки: liblzma.so

Краткое описание

`lzcat` - распаковывает в стандартный вывод

`lzcmp` - запускает cmp для сжатых файлов LZMA

`lzdiff` - запускает diff для сжатых файлов LZMA

`lzegrep` - запускает egrep для сжатых файлов LZMA

`lzfgrep` - запускает fgrep для сжатых файлов LZMA

`lzgrep` - запускает grep для сжатых файлов LZMA

`lzless` - Работает меньше с файлами, сжатыми LZMA

`lzma` - Сжимает или распаковывает файлы с использованием формата LZMA

`lzmadec` - Небольшой и быстрый декодер для сжатых файлов LZMA

`lzmainfo` - Показывает информацию, хранящуюся в заголовке сжатого файла LZMA

`lzmore` - Работает больше с файлами, сжатыми LZMA

`unlzma` - распаковывает файлы с использованием формата LZMA

`unxz` - распаковывает файлы в формате XZ

`xz` - Сжимает или распаковывает файлы в формате XZ

`xzcat` - распаковывает в стандартный вывод

`xzcmp` - запускает cmp для сжатых файлов XZ

`xzdec` - Небольшой и быстрый декодер для сжатых файлов XZ

`xzdiff` - запускает diff для сжатых файлов XZ

`xzegrep` - запускает egrep для сжатых файлов XZ

`xzfgrep` - запускает fgrep для сжатых файлов XZ

`xzgrep` - запускает grep для сжатых файлов XZ

`xzless` - Работает меньше с файлами, сжатыми XZ

`xzmore` - Работает больше с файлами, сжатыми XZ

`liblzma` - библиотека, реализующая сжатие данных без потерь с сортировкой по блокам с использованием цепного алгоритма Лемпеля-Зива-Маркова.

8.8 zstd

8.8.1 Сборка

[filename](#)

8.8.2 Тестирование

[filename](#)

8.8.3 Установка

[filename](#)

8.8.4 При раздельной структуре каталогов

[filename](#)

8.8.5 Для multilib

Очистка

[filename](#)

Сборка

[filename](#)

Установка

[filename](#)

8.8.6 Установленные файлы

Программы: zstd, zstdcat (link to zstd), zstdgrep, zstdless, zstdmt (ссылка на zstd), and unzstd (ссылка на zstd)

Библиотеки: libzstd.so

Краткое описание

`zstd` - Сжимает и распаковывает файлы с помощью алгоритма ZSTD

`libzstd` - библиотека для формата сжатия ZSTD

8.9 file

8.9.1 Настройка

8.9.2 Сборка

8.9.3 Тестирование

8.9.4 Установка

8.9.5 Установленные файлы

Программы: file

Библиотеки: libmagic.so

8.10 readline

8.10.1 Подготовка

Переустановка Readline приведет к переименованию старых библиотек в `<имя библиотеки>.old`. Хотя обычно это не проблема, в некоторых случаях это может вызвать ошибку в `ldconfig`. Этого можно избежать, выполнив следующие команды:

[filename](#)

8.10.2 Настройка

[filename](#)

Значения параметров

`--with-curses` - включает использование библиотеки `ncurses`

8.10.3 Сборка

[filename](#)

8.10.4 Установка

[filename](#)

8.10.5 При использовании раздельной структуры каталогов

[filename](#)

8.10.6 Для multilib

Очистка

[filename](#)

Настройка

[filename](#)

Сборка

[filename](#)

Установка

[filename](#)

8.10.7 Установленные файлы

Библиотеки: `libhistory.so` и `libreadline.so`

8.11 m4

8.11.1 Настройка

8.11.2 Сборка

8.11.3 Тестирование

8.11.4 Установка

8.11.5 Установленные программы:

m4

8.12 bc

8.12.1 Настройка

[filename](#)

8.12.2 Сборка

[filename](#)

8.12.3 Тестирование

[filename](#)

8.12.4 Установка

[filename](#)

8.12.5 Установленные файлы

Программы: `bc` и `dc`

8.13 flex

8.13.1 Настройка

8.13.2 Сборка

8.13.3 Тестирование

8.13.4 Установка

Некоторые программы обращаются к `lex`, а не `flex`, поэтому создайте ссылку:

8.13.5 Установленные файлы

Программы: `flex`, `flex++` (ссылка на `flex`) и `lex` (ссылка на `flex`)

Библиотеки: `libfl.so`

8.14 tcl

filename

8.14.1 Настройка

8.14.2 Сборка

8.14.3 Тестирование

?> В результатах теста есть несколько мест, связанных с `clock.test`, которые указывают на сбой, но сводка в конце указывает никаких ошибок. `clock.test` проходит на полной системе LX4.

8.14.4 Установка

Сделайте установленную библиотеку доступной для записи, чтобы отладочные символы можно было удалить позже, сделайте необходимую символьическую ссылку и переименуйте страницу руководства, которая конфликтует со страницей руководства Perl:

Сделайте необходимую ссылку:

```
1  ln -sfv tclsh8.6 /usr/bin/tclsh
```

8.15 expect

[filename](#)

8.15.1 Настройка

[filename](#)

8.15.2 Сборка

[filename](#)

8.15.3 Тестирование

[filename](#)

8.15.4 Установка

[filename](#)

8.16 dejagnu

[filename](#)

8.16.1 Настройка

[filename](#)

8.16.2 Сборка и установка

[filename](#)

8.16.3 Тестирование

[filename](#)

8.17 binutils

8.17.1 Подготовка

Удалите проблемный тест:

[filename](#)

Ошибка в системе сборки приводит к тому, что страницы руководства становятся пустыми. Можно обойти проблему, чтобы страницы руководства были созданы правильно:

```
1 sed -i '63d' etc/texi2pod.pl
2 find -name \*.1 -delete
```

8.17.2 Настройка

[filename](#)

Для multilib

[filename](#)

Значения параметров

```
--enable-gold - установить компоновщик gold.
--enable-ld=default - установить ld и ld.bfd .
--enable-plugins - включает поддержку плагинов для компоновщика.
--enable-64-bit-bfd - включает поддержку 64-битных систем.
--with-system-zlib - использовать системную версию zlib, а не включенную в пакет.
```

8.17.3 Сборка

[filename](#)

8.17.4 Тестирование

[filename](#)

 **Warning**

Известно, что четыре теста с меткой `Run property ...` могут дать сбои.

8.17.5 Установка

[filename](#)

Удалите бесполезные статические библиотеки:

[filename](#)

8.17.6 Установленные файлы

Программы: addr2line, ar, as, c++filt, dwp, elfedit, gprof, ld, ld.bfd, ld.gold, nm, objcopy, objdump, ranlib, readelf, size, strings и strip

Библиотеки: libbfd.so, libctf.so, libctf-nobfd.so и libopcodes.so

Директории: /usr/lib/ldscripts

8.18 gmp

8.18.1 Подготовка

?> По умолчанию gmp оптимизируется под ваш процессор. Для того чтобы её можно было запустить на другом процессоре, можете выполнить:

```
1 cp -v configsf.guess config.guess
2 cp -v configsf.sub config.sub
```

8.18.2 Настройка

Значения параметров

--enable-cxx - Собрать библиотеку C++

8.18.3 Сборка

8.18.4 Тестирование

8.18.5 Установка

8.18.6 Для multilib

Очистка и подготовка

Настройка

Сборка

Установка

!> Код gmp сильно оптимизирован для процессора, на котором построен. Изредка, если система LX4 была перенесена на ПК с другим процессором, при компиляции может выдавать ошибки `Illegal instruction` и прерывать сборку программы. Чтобы этого избежать, перекомпилируйте пакет `gmp`, добавив опцию `--build=x86_64-pc-linux-gnu`.

8.18.7 Установленные файлы

Библиотеки: `libgmp.so` `libgmpxx.so`

8.19 mpfr

8.19.1 Настройка

8.19.2 Сборка

8.19.3 Тестирование

8.19.4 Установка

8.19.5 Установленные файлы

Библиотеки: libmpfr.so

8.20 mpc

8.20.1 Настройка

8.20.2 Сборка

8.20.3 Тестирование

8.20.4 Установка

8.20.5 Установленные файлы

Библиотеки: libmpc.so

8.21 isl

8.21.1 Настройка

8.21.2 Сборка

8.21.3 Тестирование

8.21.4 Установка

Переместите неправильно установленные файлы:

8.22 attr

8.22.1 Настройка

[filename](#)

8.22.2 Сборка

[filename](#)

8.22.3 Тестирование

Warning

Тестирование нужно производить на файловой системе, поддерживающей расширенные атрибуты. Например, ext2-ext4.

[filename](#)

8.22.4 Установка

[filename](#)

8.22.5 При раздельной структуре каталогов

Добавьте к скрипту `configure` опцию `--bindir=/bin`.

[filename](#)

8.22.6 Для multilib

Очистка

[filename](#)

Настройка

[filename](#)

Сборка

[filename](#)

Установка

[filename](#)

8.22.7 Установленные файлы

Программы: `attr`, `getfattr`, `setfattr`

Библиотеки: `libattr.so`

8.23 acl

8.23.1 Настройка

[filename](#)

8.23.2 Сборка

[filename](#)

8.23.3 Установка

[filename](#)

8.23.4 При раздельной структуре каталогов

Добавьте к скрипту `configure` опцию `--bindir=/bin`.

[filename](#)

8.23.5 Для multilib

Очистка

[filename](#)

Настройка

[filename](#)

Сборка

[filename](#)

Установка

[filename](#)

8.23.6 Установленные файлы

Программы: `chacl`, `getfacl`, `setfacl`

Библиотеки: `libacl.so`

8.24 libcap

8.24.1 Подготовка

Отключите установку статических библиотек:

[filename](#)

8.24.2 Сборка

[filename](#)

8.24.3 Тестирование

[filename](#)

8.24.4 Установка

[filename](#)

Установите корректные права для библиотек:

[filename](#)

8.24.5 При раздельной структуре каталогов

Уберите `prefix=/usr` в сборке и установке.

[filename](#)

8.24.6 Для multilib

Очистка

[filename](#)

Сборка

[filename](#)

Установка

[filename](#)

8.24.7 Установленные файлы

Программы: `capsh`, `getcap`, `getpcaps` и `setcap`

Библиотеки: `libcap.so` и `libpsx.so`

8.25 shadow

8.25.1 Подготовка

- Отключите установку программы `groups`, так как `coreutils` предоставляет лучшую версию и содержащихся в пакете `man-pages` `man`-страниц
- Для использования более безопасного метода шифрования SHA-512 вместо стандартного `crypt` метода, а также использования `/var/mail` вместо устаревшего `/var/spool/mail` и устранения дублирования путей в переменной `PATH` по умолчанию
- Сделайте незначительное изменение, для того чтобы номера групп начинались с 1000

8.25.2 Настройка

8.25.3 Сборка

8.25.4 Установка

8.25.5 Настройка

Для включения теневых паролей и групп выполните:

В файле `/etc/default/useradd` можно настроить параметры утилиты `useradd`.

Задайте пароль на пользователя `root`:

```
1  passwd root
```

8.25.6 Установленные файлы

Программы: `chage`, `chfn`, `chgpasswd`, `chpasswd`, `chsh`, `expiry`, `faillog`, `gpasswd`, `groupadd`, `groupdel`, `groupmems`, `groupmod`, `grpck`, `grpconv`, `grpunconv`, `lastlog`, `login`, `logoutd`, `newgidmap`, `newgrp`, `newuidmap`, `newusers`, `nologin`, `passwd`, `pwck`, `pwconv`, `pwunconv`, `sg` (ссылка на `newgrp`), `su`, `useradd`, `userdel`, `usermod`, `vigr` (ссылка на `vipw`) и `vipw`

Директории: `/etc/default`

8.26 gcc

8.26.1 Подготовка

Примените патч, исправляющий несколько проблем:

[filename](#)

Исправьте пути установки библиотек:

[filename](#)

8.26.2 Настройка

Note

На данном этапе необходимы только компиляторы для С и С++, однако вы можете собрать компиляторы для любых поддерживаемых GCC языков программирования, перечислив их через запятые в опции `--enable-languages=c,c++`. GCC поддерживает следующие языки - `c,c++,d,fortran,go,objc,obj-c++`. вы можете собрать все доступные компиляторы, добавив параметр `--enable-languages=c,c++,d,fortran,go,objc,obj-c++`. Если позднее вам потребуется компилятор для какого либо языка из этого списка - пересоберите GCC с его поддержкой.

[filename](#)

Для `multilib`

[filename](#)

Значения параметров

`--disable-bootstrap` - предотвращает многократную пересборку GCC

`LD=ld` - сообщает GCC использовать ранее установленную версию компоновщика

8.26.3 Сборка

[filename](#)

8.26.4 Тестирование

- Увеличьте размер стека по умолчанию
- Произведите тестирование от непrivилегированного пользователя во избежание непредвиденных ситуаций с системой.

[filename](#)

Тестирование занимает достаточно много времени.

Для просмотра итогов теста выполните:

```
1  ../../contrib/test_summary
```

?> Известно, что 6 тестов, связанных с `get_time`, дают сбои. По видимому, это связано с локалью `en_NK`. Кроме того, тест `COSTEXPR-52830` не удается.

8.26.5 Установка

filename

- Удалите ненужную директорию,
- Убедитесь, что владелец установленных заголовков корректный,
- По историческим причинам некоторые программы могут пытаться найти `cpp` в директории `/lib`. Создайте ссылку,
- Для поддержки LTO требуется следующая символическая ссылка,
- Переместите файлы в правильное место:

filename

8.26.6 При использовании раздельных каталогов:

- Замените `ln -svr /usr/bin/cpp /usr/lib` из предыдущей команды на корректную для раздельной структуры.

8.26.7 Проверка работоспособности

Danger

Сейчас необходимо проверить работу `gcc`. Если всё нормально, то продолжайте сборку.

Выполните набор команд:

```
1 echo 'int main(){}' > dummy.c
2 cc dummy.c -v -fPIC -o dummy
3 readelf -l a.out | grep ': /lib'
```

Ошибка быть не должно, а результат команды (учитывая различия в имени динамического компоновщика, зависящие от платформы) будет следующий:

```
1 Requesting program interpreter: /lib64/ld-linux-x86-64.so.2
```

Проверим что задействованы правильные стартовые файлы. Выполните команду:

```
1 grep -o '/usr/lib.*crt[1in].*succeeded' dummy.log
```

Результат выполнения:

```
1 /usr/lib/gcc/x86_64-pc-linux-gnu/11.2.0/../../../../lib/crt1.o succeeded
2 /usr/lib/gcc/x86_64-pc-linux-gnu/11.2.0/../../../../lib/crti.o succeeded
3 /usr/lib/gcc/x86_64-pc-linux-gnu/11.2.0/../../../../lib/crtn.o succeeded
```

В зависимости от архитектуры, приведенное выше может немного отличаться. Разница будет в названии каталога после `/usr/lib/gcc`. Здесь важно обратить внимание на то, что `gcc` обнаружил все три файла `crt*.o` в каталоге `/usr/lib`.

Проверим то, что компилятор выполняет поиск корректных заголовочных файлов:

```
1 grep -B4 '^ /usr/include' dummy.log
```

Результат выполнения:

```
1 #include <...> search starts here:
2 /usr/lib/gcc/x86_64-pc-linux-gnu/11.2.0/include
3 /usr/local/include
4 /usr/lib/gcc/x86_64-pc-linux-gnu/11.2.0/include-fixed
```

Проверим, что компоновщик использует корректные пути поиска:

```
1 grep 'SEARCH.*/usr/lib' dummy.log | sed 's/|/\\n/g'
```

Результат выполнения:

```
1 SEARCH_DIR("/usr/x86_64-pc-linux-gnu/lib64")
2 SEARCH_DIR("/usr/local/lib64")
3 SEARCH_DIR("/lib64")
4 SEARCH_DIR("/usr/lib64")
5 SEARCH_DIR("/usr/x86_64-pc-linux-gnu/lib")
6 SEARCH_DIR("/usr/local/lib")
7 SEARCH_DIR("/lib")
8 SEARCH_DIR("/usr/lib");
```

Проверим, что используется корректная стандартная библиотека

```
1 grep "/lib.*/libc.so.6" dummy.log
```

Результат выполнения:

```
1 attempt to open /usr/lib/libc.so.6 succeeded
```

Проверим, что используется корректный динамический компоновщик:

```
1 grep "found ld-Linux*" dummy.log
```

Результат выполнения должен быть (учитывая различия в имени динамического компоновщика, зависящие от платформы):

```
1 found ld-Linux-x86-64.so.2 at /usr/lib/ld-Linux-x86-64.so.2
```

⚠ Warning

Если вывод не соответствует вышеуказанному, или вообще не получен, значит, что-то не так. Изучите и повторите шаги, чтобы выяснить, в чем проблема. Перед продолжением процесса необходимо решить любые проблемы.

Удалите тестовые файлы:

```
1 rm -v dummy.c a.out dummy.log
```

8.26.8 Установленные файлы

Программы: c++ (ссылка на g++), cc (ссылка на gcc), cpp, g++, gcc, gcc-ar, gcc-nm, gcc-ranlib, gcov, gcov-dump и gcov-tool

Библиотеки: libasan.{a,so}, libatomic.{a,so}, libcc1.so, libgcc.a, libgcc_eh.a, libgcc_s.so, libgcov.a, libgomp.{a,so}, libitm.{a,so}, liblsan.{a,so}, liblto_plugin.so, libquadmath.{a,so}, libssp.{a,so}, libssp_nonshared.a, libstdc++.{a,so}, libstdc++fs.a, libsupc++.a, libtsan.{a,so} и libubsan.{a,so}

Директории: /usr/include/c++, /usr/lib/gcc, /usr/libexec/gcc и /usr/share/gcc-11.2.0

8.27 pkg-config

8.27.1 Настройка

8.27.2 Сборка

8.27.3 Тестирование

8.27.4 Установка

8.27.5 Установленные файлы

Программы: `pkg-config`

8.28 ncurses

8.28.1 Настройка

Значения параметров

`--without-normal` - отключает установку большинства статических библиотек.

`--enable-pc-files` - включает установку файлов для `pkg-config`.

`--enable-widec` - включает сборку библиотек с широкими (многобайтовыми) символами. Они совместимы с обычными библиотеками `ncurses` при сборке из исходного кода, но не совместимы бинарно.

8.28.2 Сборка

filename

8.28.3 Установка

filename

Многие пакеты при компоновке ищут библиотеки без широких символов. Для компоновки с библиотеками содержащими широкие символы выполните:

```
1 for lib in ncurses form panel menu ; do
2     rm -vf             /usr/lib/lib${lib}.so
3     echo "INPUT(-l${lib}w)" > /usr/lib/lib${lib}.so
4     ln -sfv ${lib}w.pc   /usr/lib/pkgconfig/${lib}.pc
5 done
```

Для сборки старых программ использующих `-lcurses` выполните:

```
1 rm -vf             /usr/lib/libcursesw.so
2 echo "INPUT(-lncursesw)" > /usr/lib/libcursesw.so
3 ln -sfv libcursesw.so /usr/lib/libcursesw.so
```

Удалите ненужную статическую библиотеку:

```
1 rm -fv /usr/lib/libcursesw.a
```

?> Если для запуска старых бинарных программ требуется библиотека `ncurses` без широких символов - соберите её:

```
1 make distclean
2 ./configure --prefix=/usr \
3             --with-shared \
4             --without-normal \
5             --without-debug \
6             --without-cxx-binding \
7             --with-abi-version=5
8 make sources libs
9 cp -av lib/lib*.so.5* /usr/lib
```

8.28.4 При раздельной структуре каталогов

filename

8.28.5 Для multilib

Очистка

```
1 make distclean
```

Настройка[filename](#)**Сборка**[filename](#)**Установка**[filename](#)

?> Если для запуска старых бинарных программ требуется библиотека `ncurses` без широких символов - соберите её:

```

1  make distclean
2  CC="gcc -m32" CXX="g++ -m32" ./configure --prefix=/usr    \
3      --with-shared \
4      --without-normal \
5      --without-debug \
6      --without-cxx-binding \
7      --with-abi-version=5 --host=i686-pc-linux-gnu
8  make sources libs
9  cp -av lib/lib*.so.5* /usr/lib

```

8.28.6 Установленные файлы

Программы: `captioninfo` (ссылка на `tic`), `clear`, `infocmp`, `infotocap` (ссылка на `tic`), `ncursesw6-config`, `reset` (ссылка на `tset`), `tabs`, `tic`, `toe`, `tput` и `tset`

Библиотеки: `libcursesw.so` (ссылка на `libcursesw.so`), `libformw.so`, `libmenuw.so`, `libcursesw.so`, `libpanelw.so` и их версии без широких символов

Директории: `/usr/share/tabset` `/usr/share/terminfo`

8.29 sed

8.29.1 Настройка

[filename](#)

8.29.2 Сборка

[filename](#)

8.29.3 Сборка документации

[filename](#)

8.29.4 Тестирование

[filename](#)

8.29.5 Установка

[filename](#)

8.29.6 Установка документации

[filename](#)

8.30 psmisc

8.30.1 Настройка

[filename](#)

8.30.2 Сборка

[filename](#)

8.30.3 Установка

[filename](#)

8.30.4 При раздельной структуре каталогов

[filename](#)

8.31 gettext

8.31.1 Настройка

8.31.2 Сборка

8.31.3 Тестирование

Тестирование добавляет +3 SBU ко всему времени установки пакета.

8.31.4 Установка

8.32 bison

8.32.1 Настройка

[filename](#)

8.32.2 Сборка

[filename](#)

8.32.3 Тестирование

[filename](#)

Добавляет примерно 5,5 SBU к общему времени установки пакета

8.32.4 Установка

[filename](#)

8.33 grep

8.33.1 Настройка

[filename](#)

8.33.2 Сборка

[filename](#)

8.33.3 Тестирование

[filename](#)

8.33.4 Установка

[filename](#)

8.34 bash

8.34.1 Подготовка

Для многоядерных процессоров внесите исправление, которое устраняет проблему "сстояния гонки" при использовании нескольких ядер.

[filename](#)

8.34.2 Настройка

[filename](#)

Значения параметров

`--without-bash-malloc` - этот параметр отключает использование функции выделения памяти (malloc) Bash, которая вызывает ошибки сегментации. Отключив эту опцию, Bash будет использовать функции malloc из libc, которые более стабильны.

`--with-installed-readline` - указывает на то, что следует задействовать ранее установленную библиотеку readline, вместо использования внутренней.

8.34.3 Сборка

[filename](#)

8.34.4 Тестирование

Для корректного выполнения тестов, сделайте пользователя `tester` владельцем каталога и запустите тесты от пользователя `tester`

[filename](#)

8.34.5 Установка

[filename](#)

8.34.6 При раздельной структуре каталогов

Создайте ссылку на нужный бинарный файл.

[filename](#)

`bash` должен находиться в `/bin`. Для упрощённой структуры этого делать не требуется.

8.34.7 Запуск новой сессии

Запустите `bash` (заменив тот, который в настоящее время выполняется):

```
1 exec /bin/bash --login +h
```

8.35 libtool

8.35.1 Настройка

8.35.2 Сборка

8.35.3 Тестирование

?> Если у вас несколько ядер процессора, то можно значительно (иногда более чем на 60%) сократить время выполнения тестирования. Перед выполнением тестов объявит переменную: `TESTSUITEFLAGS=-j<N>`, где `<N>` - число ядер ЦП.

?> Пять тестов могут дать сбой, однако все тесты проходят после установки `automake`.

8.35.4 Установка

Удалите ненужную статическую библиотеку

8.35.5 Для multilib

Очистка

Настройка

Сборка

Установка

8.36 gdbm

8.36.1 Настройка

Значения параметров

`--enable-libgdbm-compat` - позволяет использовать библиотеку для совместимости с `libgdbm`. Некоторые пакеты могут использовать старые процедуры DBM, которые и предоставляет эта библиотека.

8.36.2 Сборка

8.36.3 Тестирование

?> Известно, что один тест (`version`) может дать сбой.

8.36.4 Установка

8.37 gperf

8.37.1 Настройка

8.37.2 Сборка

8.37.3 Тестирование

!> Тестирование должно производиться в один поток, так как при одновременном тестировании (многопотоковом) появляются всевозможные ошибки. Поэтому и используется ключ `-j1`:

8.37.4 Установка

8.38 expat

8.38.1 Настройка

8.38.2 Сборка

8.38.3 Тестирование

8.38.4 Установка

8.38.5 Установка документации

8.38.6 Для multilib

Очистка

Настройка

Сборка

Установка

8.39 inetutils

8.39.1 Настройка

[filename](#)

Значения параметров

`--disable-*` - отключает установку программ, лучшие версии которых предоставляют другие пакеты.

8.39.2 Сборка

[filename](#)

8.39.3 Тестирование

[filename](#)

?> Тест `libs.sh` даёт сбой, когда система LX4 находится в среде `chroot`, а тест `ping-localhost.sh` не проходит, если в хост-системе нет поддержки IPv6.

8.39.4 Установка

[filename](#)

8.39.5 При раздельной структуре каталогов

[filename](#)

8.39.6 Установленные файлы

Программы: `dnsdomainname`, `ftp`, `ifconfig`, `hostname`, `ping`, `ping6`, `talk`, `telnet`, `tftp` и `traceroute`

8.40 perl

8.40.1 Настройка

Для использования системных версий `zlib` и `bzip2` вместо встроенных выполните:

Запустите скрипт `configure` (Для полного контроля над настройкой удалите опцию `-des`)

Значения параметров

`-Dvendorprefix=/usr` - Устанавливать модули в `/usr`.
`-Dpager="/usr/bin/less -isR"` - использовать `less` вместо `more`.
`-Duseshrplib` - Установить динамическую библиотеку.
`-Dusethreads` - использовать многопоточность.

8.40.2 Сборка

8.40.3 Тестирование

Тесты добавляют 11 SBU ко всему времени установки пакета.

8.40.4 Установка

8.40.5 Установленные файлы

Программы: `corelist`, `cpan`, `enc2xs`, `encguess`, `h2ph`, `h2xs`, `instmodsh`, `json_pp`, `libnetcfg`, `perl`, `perl5.32.1` (жёсткая ссылка на `perl`), `perlbug`, `perldoc`, `perlivp`, `perlthanks` (жёсткая ссылка на `perlbug`), `piconv`, `pl2pm`, `pod2html`, `pod2man`, `pod2text`, `pod2usage`, `podchecker`, `podselect`, `prove`, `ptar`, `ptardiff`, `ptargrep`, `shasum`, `splain`, `xsubpp` и `zipdetails`

Библиотеки: Множество в директории `/usr/lib/perl5`

Директории: `/usr/lib/perl5`

8.41 XML::Parser

8.41.1 Настройка

8.41.2 Сборка

8.41.3 Тестирование

8.41.4 Установка

8.41.5 Установленные файлы

Библиотеки: Модуль perl - Expat.so

8.42 intltool

8.42.1 Настройка

Исправьте предупреждение, которое может появиться при использовании perl версии 5.22 и выше:

8.42.2 Сборка

8.42.3 Тестирование

8.42.4 Установка

8.42.5 Установка документации

8.43 autoconf

8.43.1 Настройка

[filename](#)

8.43.2 Сборка

[filename](#)

8.43.3 Тестирование

[filename](#)

 **Note**

Если у вас несколько ядер процессора, то можно значительно (иногда более чем на 60%) сократить время выполнения тестирования. Перед выполнением тестов объявит переменную: `TESTSUITEFLAGS=-j<N>`, где `<N>` - число ядер ЦП.

8.43.4 Установка

[filename](#)

8.44 automake

8.44.1 Настройка

Внесите исправление для некорректного теста: [filename](#) [filename](#)

8.44.2 Сборка

[filename](#)

8.44.3 Тестирование

[filename](#)

Warning

Известно, что тесты `t/subobj.sh`, `t/deprecated-acinit.sh` и `t/init.sh` не проходят в LX4.

8.44.4 Установка

[filename](#)

8.45 kmod

8.45.1 Настройка

[filename](#)

Значения параметров

`--with-xz, --with-zlib, --with-zstd`

Параметры позволяют `Kmod` обрабатывать сжатые модули ядра соответствующим алгоритмом сжатия.

8.45.2 Сборка

[filename](#)

8.45.3 Тестирование

Пакет не имеет тестов, которые можно запустить непосредственно сейчас. Необходимо дополнительно установить `git`, при этом, некоторые тесты также не будут выполнены вне репозитория.

8.45.4 Установка

[filename](#)

Необходимо создать символические ссылки (символинки) для совместимости с `Module-Init-Tools` (предыдущая реализация программы обработки модулей ядра).

[filename](#)

Заметьте, что эту команду не следует вводить, если вы используете систему с раздельной структурой каталогов, в таком случае перейдите к следующему шагу.

8.45.5 При раздельной структуре каталогов

Добавьте опции `--bindir=/bin` и `--with-rootlibdir=/lib` скрипту `configure` из пункта "Настройка".

Измените предыдущую команду (создающую символинки для совместимости с `Module-Init-Tools`) и создайте необходимую ссылку в `/bin`:

[filename](#)

8.45.6 Для multilib

Очистка

Очистите предыдущую сборку, но сохраните страницы руководства, так как они не могут быть воссозданы, поскольку пакет `xsltproc` не установлен:

[filename](#)

Подготовка

[filename](#)

Сборка

[filename](#)

Установка

[filename](#)

8.46 libelf

8.46.1 Настройка

[filename](#)

8.46.2 Сборка

[filename](#)

8.46.3 Тестирование

[filename](#)

8.46.4 Установка

Потребуется только установка библиотеки `libelf`.

[filename](#)

8.46.5 При раздельной структуре каталогов

Добавьте к скрипту `configure` опцию `--libdir=/lib`.

[filename](#)

8.46.6 Для multilib

Очистка

[filename](#)

Подготовка

[filename](#)

Сборка

[filename](#)

Установка

[filename](#)

8.47 libffi

8.47.1 Настройка

?> Libffi как и GMP оптимизируется под определённый процессор. Если планируется переносить систему на компьютер с другим ЦП, экспортируйте `CFLAGS` и `CXXFLAGS`, чтобы указать универсальную сборку для вашей архитектуры.

8.47.2 Сборка

8.47.3 Тестирование

8.47.4 Установка

8.47.5 Для multilib

Очистка

Подготовка

Сборка

Установка

8.47.6 Установленные файлы

Библиотеки: `libffi.so`

8.48 openssl

8.48.1 Настройка

8.48.2 Сборка

8.48.3 Тестирование

?> Известно, что один тест 30-test_afalg.t даст сбой в некоторых конфигурациях ядра (по-видимому, предполагает, что некоторые неопределенные параметры Crypto были выбраны).

8.48.4 Установка

8.48.5 Для multilib

Очистка

Подготовка

Сборка

Установка

8.48.6 Установленные файлы

Программы: c_rehash и openssl

Библиотеки: libcrypto.so и libssl.so

Директории: /etc/ssl, /usr/include/openssl и /usr/lib/engines

8.49 python

8.49.1 Настройка

8.49.2 Сборка

8.49.3 Установка

8.49.4 Тестирование

!> Тесты запускать не рекомендуется. Они могут зависнуть. При желании выполните тесты при переустановке Python, но уже в руководстве Extra .

8.49.5 Установленные файлы

Программы: 2to3, idle3, pip3, pydoc3, python3 и python3-config

Библиотеки: libpython3.9.so и libpython3.so

Директории: /usr/include/python3.9 и /usr/lib/python3

8.50 ninja

8.50.1 Подготовка

filename

?> `ninja` запускает максимальное количество процессов параллельно. По умолчанию это количество ядер в системе плюс два. В некоторых случаях это может привести к перегреву ЦП или нехватке памяти в системе. При запуске из командной строки передача параметра `-jN` ограничит количество параллельных процессов, но некоторые пакеты могут не передавать параметр `-j`.

Использование переменной окружения `NINJAJOBS` гарантирует ограничение на количество параллельных процессов.

Экспортируя эту переменную, укажите требуемое количество процессов, в соответствии с возможностями:

```
1  export NINJAJOBS=4
```

Для того чтобы задействовать значение переменной `NINJAJOBS`, выполните корректировку:

8.50.2 Сборка

Значения параметров

`--bootstrap` - параметр определяет, что необходимо выполнить сборку для данной системы.

8.50.3 Тестирование

8.50.4 Установка

8.50.5 Установленные файлы

Программы: `ninja`

8.51 meson

?> Хотя данный пакет предлагается использовать при установке `systemd`, однако он может понадобиться при установке многих других пакетов, за пределами создания базовой системы. Вернитесь к этой инструкции по мере необходимости.

8.51.1 Подготовка

Убедитесь, что директория для библиотек по умолчанию корректна:

8.51.2 Сборка

8.51.3 Установка

Значения параметров

`--root=dest` - по умолчанию, все файлы устанавливаются в Python Eggs. При указании параметра `--root=dest`, установка файлов будет выполнена в соответствующий каталог `dest` с сохранением иерархии файловой системы. Затем, выполняется копирование из неё в целевую.

8.51.4 Установленные файлы

Программы: `meson`

8.52 coreutils

8.52.1 Дополнительные необходимые файлы

`{{ patch.url}}`

8.52.2 Подготовка

Примените необязательный патч для поддержки локализации:

`filename`

Bug

В этом патче могут встречаться баги. При обнаружении новых ошибок обязательно отошлите отчёт об этом сопровождающим `coreutils`, перед этим проверив, воспроизводима ли ошибка без этого патча.

Удалите проблемный тест:

`filename`

8.52.3 Настройка

`filename`

Значения параметров

`autoreconf` - Требует патч поддержки локализации;

`FORCE_UNSAFE_CONFIGURE=1` - Разрешает запуск `configure` под пользователем `root`.

8.52.4 Сборка

`filename`

8.52.5 Тестирование

`filename`

Warning

Известно, что тест `test-getlogin` не проходит в LX4.

8.52.6 Установка

`filename`

8.52.7 При раздельной структуре каталогов

`filename`

8.52.8 Установленные файлы

Программы: [, b2sum, base32, base64, basename, basenc, cat, chcon, chgrp, chmod, chown, chroot, cksum, comm, cp, csplit, cut, date, dd, df, dir, dircolors, dirname, du, echo, env, expand, expr, factor, false, fmt, fold, groups, head, hostid, id, install, join, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mktemp, mv, nice, nl, nohup, nproc, numfmt, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, realpath, rm, rmdir, runcon, seq, sha1sum, sha224sum, sha256sum, sha384sum, sha512sum, shred, shuf, sleep, sort, split, stat, stdbuf, stty, sum, sync, tac, tail, tee, test, timeout, touch, tr, true, truncate, tsort, tty, uname, unexpand, uniq, unlink, users, vdir, wc, who, whoami и yes

Библиотеки: libstdbuf.so (в /usr/libexec/coreutils)

8.53 check

8.53.1 Настройка

[filename](#)

8.53.2 Сборка

[filename](#)

8.53.3 Тестирование

[filename](#)

Тесты добавляют 4 SBU ко всему времени установки пакета.

8.53.4 Установка

[filename](#)

8.54 diffutils

8.54.1 Настройка

[filename](#)

8.54.2 Сборка

[filename](#)

8.54.3 Тестирование

[filename](#)

8.54.4 Установка

[filename](#)

8.55 gawk

8.55.1 Подготовка

Следует убедиться, что ненужные файлы не будут установлены:

8.55.2 Настройка

8.55.3 Сборка

8.55.4 Тестирование

8.55.5 Установка

8.55.6 Установка документации

8.56 findutils

8.56.1 Настройка

`filename`

Значения параметров

`--localstatedir` - замена расположения базы данных `locate` в `/var/lib/locate` для соответствия FHS.

8.56.2 Сборка

`filename`

8.56.3 Тестирование

`filename`

8.56.4 Установка

`filename`

8.56.5 При раздельной структуре каталогов

`filename`

8.57 groff

8.57.1 Подготовка

Groff ожидает переменную окружения `PAGE`, значение которой должно содержать формат бумаги по умолчанию. Указание значение переменной `PAGE=A4` может быть более подходящим. Хотя значение по умолчанию задается во время компиляции, его можно переопределить позже, записав в файл `/etc/papersize`.

8.57.2 Настройка

Значения параметров

`PAGE=A4` - формат бумаги.

8.57.3 Сборка

Пакет не поддерживает параллельную сборку. Выполните компиляцию пакета:

8.57.4 Установка

8.58 less

8.58.1 Настройка

Значения параметров

--sysconfdir=/etc - настроить путь `/etc` для хранения конфигурации.

8.58.2 Сборка

8.58.3 Установка

8.59 gzip

8.59.1 Настройка

[filename](#)

8.59.2 Сборка

[filename](#)

8.59.3 Тестирование

[filename](#)

8.59.4 Установка

[filename](#)

8.59.5 При раздельной структуре каталогов

[filename](#)

8.60 iproute2

8.60.1 Подготовка

- Программа `arpd` требует установленную Berkeley DB. Отключите её,
- Отключите 2 модуля, требующие `iptables`,

8.60.2 Сборка

8.60.3 Установка

8.60.4 Установка документации

8.60.5 Установленные файлы

Программы: bridge, ctstat (ссылка на `lndstat`), genl, ifcfg, ifstat, ip, lndstat, nstat, routef, routel, rtacct, rtmon, rtpr, rtstat (ссылка на `lndstat`), ss, and tc

8.61 kbd

8.61.1 Дополнительные необходимые файлы

[{{ patch.url }}](#)

8.61.2 Подготовка

Примените патч для исправления работы клавиш backspace и delete:

Удалите ненужную программу `resizecons`, требующую `svgalib`:

8.61.3 Настройка

Значения параметров

`--disable-vlock` - Данная библиотека требует Linux-PAM.

8.61.4 Сборка

8.61.5 Тестирование

8.61.6 Установка

!> Пакет `kbd` не предоставляет некоторых рабочих раскладок клавиатуры (например, для белорусского языка). Загрузите (при необходимости) эти раскладки отдельно.

8.62 libpipeline

8.62.1 Настройка

8.62.2 Сборка

8.62.3 Тестирование

8.62.4 Установка

8.63 make

8.63.1 Настройка

8.63.2 Сборка

8.63.3 Тестирование

8.63.4 Установка

8.64 patch

8.64.1 Настройка

8.64.2 Сборка

8.64.3 Тестирование

8.64.4 Установка

8.65 tar

8.65.1 Настройка

[filename](#)

8.65.2 Сборка

[filename](#)

8.65.3 Тестирование

[filename](#)

Тестирование добавляет 3 SBU ко всему времени установки пакета

?> Известно, что тест `store/restore` даёт сбой.

8.65.4 Установка

[filename](#)

8.66 man-db

8.66.1 Настройка

8.66.2 Значения параметров

- `--disable-setuid` - запрещает задавать программе man setuid пользователю man.
- `--enable-cache-owner=bin` - задаёт права доступа общесистемному кэшу пользователю, который является владельцем каталога bin.
- `--with-...` - эти три аргумента используются для настройки программ по умолчанию. `lynx` текстовый веб-обозреватель, `vgrind` преобразовывает исходный код программ в входные данные, `Groff` и `grap` полезны для набора графов в документах `Groff`. Программы `vgrind` и `grap` обычно нужны для просмотра справочных страниц.

8.66.3 Сборка

8.66.4 Тестирование

8.66.5 Установка

8.66.6 Страницы руководств на других языках

В нижеприведенной таблице указано соответствие кодировки, которая допускается при использовании в Man-DB страницах, расположенных в каталоге `/usr/share/man/`. Кроме этого, Man-DB правильно определяет, имеют ли страницы руководства, установленные в этом каталоге, кодировку `UTF-8`.

Язык (код)	Кодировка
Belarusian (be)	CP1251
Bulgarian (bg)	CP1251
Croatian (hr)	ISO-8859-2
Czech (cs)	ISO-8859-2
Danish (da)	ISO-8859-1
Dutch (nl)	ISO-8859-1
English (en)	ISO-8859-1
Estonian (et)	ISO-8859-1
Finnish (fi)	ISO-8859-1
French (fr)	ISO-8859-1
Galician (gl)	ISO-8859-1
German (de)	ISO-8859-1
Greek (el)	ISO-8859-7
Hungarian (hu)	ISO-8859-2
Icelandic (is)	ISO-8859-1
Indonesian (id)	ISO-8859-1
Irish (ga)	ISO-8859-1
Italian (it)	ISO-8859-1
Japanese (ja)	EUC-JP
Korean (ko)	EUC-KR
Latvian (lv)	ISO-8859-13
Lithuanian (lt)	ISO-8859-13
Macedonian (mk)	ISO-8859-5
Norwegian (no)	ISO-8859-1
Norwegian Bokmal (nb)	ISO-8859-1
Norwegian Nynorsk (nn)	ISO-8859-1
Polish (pl)	ISO-8859-2
Portuguese (pt)	ISO-8859-1
Romanian (ro)	ISO-8859-2
Russian (ru)	KOI8-R
Serbian (sr)	ISO-8859-5
Serbian Latin (sr@latin)	ISO-8859-2
Simplified Chinese (zh_CN)	GBK
Simplified Chinese, Singapore (zh_SG)	GBK
Slovak (sk)	ISO-8859-2

Язык (код)	Кодировка
Slovenian (sl)	ISO-8859-2
Spanish (es)	ISO-8859-1
Swedish (sv)	ISO-8859-1
Traditional Chinese (zh_TW)	BIG5
Traditional Chinese, Hong Kong (zh_HK)	BIG5HKSCS
Turkish (tr)	ISO-8859-9
Ukrainian (uk)	KOI8-U
Vietnamese (vi)	TCVN5712-1

!> Страницы на других языках не поддерживаются.

8.67 texinfo

8.67.1 Настройка

8.67.2 Сборка

8.67.3 Тестирование

8.67.4 Установка

При желании установите компоненты TeX.

`TEXMF=/usr/share/texmf` - Переменная `TEXMF` содержит местоположение корня дерева TeX, если пакет TeX будет установлен позже.

Система документации Info использует простой текстовый файл для хранения списка пунктов меню. Файл находится в каталоге `/usr/share/info/dir`. К сожалению, из-за случайных проблем в Make-файлах различных пакетов, он иногда может выйти из синхронизации с информационными страницами, установленными в системе. Если каталог `/usr/share/info/dir` когда-либо потребуется создать заново, следующие команды выполняют эту задачу:

8.68 popt

8.68.1 Настройка

8.68.2 Сборка

8.68.3 Тестирование

8.68.4 Установка

8.69 freetype

8.69.1 Настройка

 **Note**

В руководстве extra, данный пакет следует переустановить после установки `harfbuzz` из-за циклической зависимости.

8.69.2 Сборка

8.69.3 Установка

8.69.4 Для multilib

Очистка

Настройка

Сборка

Установка

8.70 dosfstools

8.70.1 Настройка

8.70.2 Сборка

8.70.3 Установка

8.71 wget

8.71.1 Настройка

Значения параметров

- `--sysconfdir=/etc` - меняет путь к директории с настройками с `/usr/etc` на `/etc`
- `--with-ssl=openssl` - позволяет использовать openssl вместо GnuTLS.

8.71.2 Сборка

8.71.3 Установка

8.71.4 Конфигурационные файлы

- `/etc/wgetrc`
- `~/.wgetrc`

8.71.5 Установленные файлы

- **Установленные программы:** `wget`

8.72 libasn1

8.72.1 Настройка

8.72.2 Сборка

8.72.3 Тестирование

8.72.4 Установка

8.72.5 Для multilib

Очистка

8.72.6 Настройка

Сборка

Установка

8.73 p11-kit

8.73.1 Подготовка

Подготовьте специфичный для дистрибутива скрипт:

8.73.2 Настройка

Значение параметров configure

`--with-trust-paths=/etc/pki/anchors` - задает путь для доверенных сертификатов

8.73.3 Сборка

8.73.4 Установка

8.73.5 Для multilib

Очистка

8.73.6 Настройка

Сборка

Установка

8.74 make-ca

Сценарий загрузит и обработает сертификаты, включенные в файл certdata.txt. Любые локальные сертификаты, хранящиеся в `/etc/ssl/local`, будут импортированы в сгенерированные хранилища сертификатов (перекрывая доверие Mozilla). Кроме того, любые измененные значения будут скопированы из `/etc/ssl/local` до любых обновлений, с сохранением настраиваемых значений, которые отличаются от Mozilla при использовании утилиты p11-kit для работы с хранилищем доверенных сертификатов.

Перед установкой, вам необходимо скопировать из хост-системы файлы `/etc/{hosts,resolv.conf}` для правильной работы сети. Откройте вторую консоль/терминал и выполните:

```
1 cp -v /etc/{hosts,resolv.conf} $LIN/etc
```

Заменив `$LIN` на путь к собираемой ОС.

Чтобы установить различные хранилища сертификатов, установите сценарий make-ca в правильное место:

Загрузите источник сертификата и подготовьте к использованию в системе с помощью следующей команды:

При повторном запуске сценария с той же версией `certdata.txt`, например, для добавления дополнительных хранилищ по мере установки необходимого программного обеспечения, добавьте переключатель `-g` в командную строку. При упаковке запустите `make-ca --help`, чтобы увидеть все доступные параметры командной строки.

8.75 MarkupSafe

[filename](#)

[8.75.1 Сборка](#)

[8.75.2 Установка](#)

8.76 Jinja2

`filename`

8.76.1 Установка

8.77 Выбор текстового редактора

8.77.1 Выбор текстового редактора

Текстовый редактор — самостоятельная компьютерная программа или компонент программного комплекса (например, редактор исходного кода интегрированной среды разработки или окно ввода в браузере), предназначенная для создания и изменения текстовых данных в общем и текстовых файлов, в частности.

Текстовые редакторы предназначены для работы с текстовыми файлами в интерактивном режиме. Они позволяют просматривать содержимое текстовых файлов и производить над ними различные действия: вставку, удаление и копирование текста, контекстный поиск и замену, сортировку строк, просмотр кодов символов и конвертацию кодировок, печать и т. п.

Часто интерактивные текстовые редакторы содержат дополнительную функциональность, призванную автоматизировать действия по редактированию (от записываемых последовательностей нажатий клавиш до полноценных встроенных языков программирования), или отображают текстовые данные специальным образом (например, с подсветкой синтаксиса).

Многие текстовые редакторы являются редакторами исходного кода, то есть они ориентированы на работу с текстами программ.

Выбор текстового редактора - индивидуальное предпочтение. В данном разделе будут представлены некоторые, наиболее популярные и подходящие в контексте сборки базовой системы.

8.77.2 vim

Настройка

Измените расположение файла `vimrc` на `/etc`:

И сконфигурируйте пакет:

Сборка

Тестирование

?> Так как тестирование выводит на экран большое кол-во двоичных данных, это может вызвать проблемы с настройками текущего терминала. Именно поэтому используется перенаправление вывода в файл.

Установка

Настройка Vim

- Многие пользователи привыкли использовать `vi` вместо `vim`. Чтобы выполнять `vim`, когда пользователь обычно ввёл `vi`, создайте символьическую ссылку как для бинарного файла, так и для страницы руководства;
- Документация Vim устанавливается в `/usr/share/vim`. Для совместимости с другими пакетами, создайте символьическую ссылку;
- По умолчанию Vim работает в режиме, несовместимом с Vi. Это может быть неприятным для тех пользователей, которые использовали другие редакторы в прошлом. Параметр несовместимости включен, чтобы подчеркнуть факт о том, что используется новое поведение. Также он напоминает тем, кто перейдёт в совместимый режим, что это должен быть первый параметр в файле конфигурации (это необходимо, потому что при этом изменяются другие параметры; предопределения должны выполняться после этого параметра):
- `set noscompatible` - указывает Vim'у вести себя более удобным образом (по умолчанию), чем vi-совместимый. Удалите `no`, чтобы сохранить старое поведение `vi`.
- `set backspace=2` - позволяет делать обратный интервал при переносе строк, автоотступах и начале вставки. Синтаксис параметра включает подсветку синтаксиса vim.
- `set lbr` - включает перенос текста по словам.
- `set mouse=a` - позволяет правильно вставлять текст с помощью мыши при работе в chroot или через удаленное соединение.
- Оператор `if` с параметром `set background=dark` исправляет предположение vim о цвете фона некоторых эмуляторов терминала. Это даёт лучшую цветовую схему выделения для использования на чёрном фоне этих программ.

Установленные файлы

- **Установленные программы:** `ex` (ссылка на `vim`), `rview` (ссылка на `vim`), `rvim` (ссылка на `vim`), `vi` (ссылка на `vim`), `view` (ссылка на `vim`), `vim`, `vimdiff` (ссылка на `vim`), `vimtutor` и `xxd`
- **Установленные библиотеки:** нет
- **Установленные директории:** `/usr/share/vim`

8.77.3 emacs

Настройка

Сборка

Установка

Info

Emacs устанавливает файлы иконок в `/usr/share/icons/hiColor`. После того, как вы скомпилируете Xorg/Wayland и GTK+ 2.24.33 или 3.24.25, вы можете улучшить производительность и использование памяти, обновив файл `/usr/share/icons/hiColor/index.theme`. Выполните:

Установленные файлы

- **Установленные программы:** `ctags`, `ebrowse`, `emasc` (символическая ссылка на `emacs-27.1`), `emacs-27.1`, `emacsclient`, `etags` и `grep-changelog`
- **Установленные библиотеки:** нет
- **Установленные директории:** `/usr/libexec/emacs`, `/usr/share/emacs` и `/var/games/emacs`

8.77.4 nano

Настройка

Сборка

Тестирование

Установка

Установка документации

8.78 Выбор системы инициализации

8.78.1 Выбор системы инициализации

В операционной системе Linux после завершения загрузки ядра начинается инициализация Linux системы, сервисов и других компонентов.

За это отвечает процесс инициализации, он запускается ядром сразу после завершения загрузки, имеет PID 1, и будет выполняться, пока будет работать система. Обычно (согласно Filesystem Hierarchy Standard) располагается по пути `/sbin/init`. Существуют отличия в организации работы подсистемы в операционных системах, ведущих родословную от System V и систем в стиле BSD.

В процессе загрузки после инициализации ядра, как правило, запускается `/sbin/init`, как первый процесс пользовательского режима, и `init` отвечает за дальнейшую загрузку системы. Для этого запускаются стартовые сценарии, которые выполняют проверку и монтирование файловых систем, запуск необходимых сервисов, настройку ядра (в том числе загрузку модулей ядра согласно установленному оборудованию, настройку IP-адресов, таблиц маршрутизации и другие задачи), запуск графической оболочки. Основная информация для загрузки, как правило, размещается в `/etc/inittab`.

?> Данный раздел предоставляет выбор между несколькими системами инициализации. Вам необходимо самостоятельно выбрать желаемую.

8.78.2 SysVInit

SysVinit

В этой части приведены инструкции по сборке и настройки системы инициализации SysVinit. Несмотря на то, что это достаточно старый инид, он до сих пор популярен у многих пользователей Linux.

Раньше был де-факто стандартом системы инициализации. Но в начале 2010-х были совершены попытки замены SysVinit на другие. Например, Canonical представила миру `upstart`, Gentoo - `OpenRC`, a Red Hat - `systemd`. И только у Red Hat со своим `systemd` получилось создать тот инид, который сейчас устанавливается в большинстве дистрибутивов Linux.

ДОСТОИНСТВА SYSVINIT:

- Устоявшаяся и хорошо понятная система
- Простая настройка
- Стабильная и надёжная работа

НЕДОСТАТКИ SYSVINIT

- Неудобная (а для некоторых ещё и сложная) работа с сервисами
- Последовательная обработка задач загрузки, что может в некоторых случаях замедлить скорость старта ОС

SysVinit неплохо подойдёт для владельцев старого и/или слабого ПК, так как это быстрый, простой и понятный инид. Либо же тем, кому не нравится `systemd` за свою монолитную структуру. В остальных же случаях советуется собрать `systemd`.

eudev

ДОПОЛНИТЕЛЬНЫЕ НЕОБХОДИМЫЕ ФАЙЛЫ

`{{ patch.url}}`

НАСТРОЙКА

Если вы собираете систему с раздельной структурой каталогов, то пропустите этот шаг и приступите к следующему.

`filename`

ПРИ РАЗДЕЛЬНОЙ СТРУКТУРЕ КАТАЛОГОВ

`filename`

СБОРКА

`filename`

ТЕСТИРОВАНИЕ

`filename`

УСТАНОВКА

`filename`

Установите необходимые файлы:

`filename`

ДЛЯ MULTILIB

Очистка

`filename`

Настройка

`filename`

Сборка

`filename`

Установка

`filename`

procps-ng

НАСТРОЙКА

filename

СБОРКА

filename

ТЕСТИРОВАНИЕ

filename

?> Пять тестов, связанных с `pkill` дают сбои, так как они (тесты) не были обновлены.

УСТАНОВКА

filename

ПРИ РАЗДЕЛЬНОЙ СТРУКТУРЕ КАТАЛОГОВ

filename

util-linux**НАСТРОЙКА**[filename](#)**СБОРКА**[filename](#)

Если вы собираете систему с раздельной структурой каталогов, уберите аргумент `--libdir=/usr/lib`!

ТЕСТИРОВАНИЕ

!> Тестирование пакета от имени пользователя `root` может сломать Вашу систему. Для того чтобы этого не случилось, производите тесты от имени непrivилегированного пользователя. Для запуска тестов параметр `CONFIG_SCSI_DEBUG` для ядра должен быть доступен в текущей системе и должен быть собран в виде модуля. Также должны быть установлены некоторые другие пакеты из руководства extra. При желании этот тест может быть запущен после перезагрузки в завершенную систему LX4: `bash tests/run.sh --srcdir=$PWD --builddir=$PWD`

[filename](#)**УСТАНОВКА**[filename](#)**ДЛЯ MULTILIB****Очистка**[filename](#)**Настройка**[filename](#)**Сборка**[filename](#)**Установка**[filename](#)

sysklogd**ПОДГОТОВКА**

Исправьте ошибку, приводящую к краху программы:

[filename](#)

СБОРКА

[filename](#)

ТЕСТИРОВАНИЕ

[filename](#)

УСТАНОВКА

[filename](#)

Если вы собираете систему с раздельной структурой каталогов, то пропустите шаг с установкой, перейдя к следующему

ПРИ РАЗДЕЛЬНОЙ СТРУКТУРЕ КАТАЛОГОВ

[filename](#)

НАСТРОЙКА

Создайте конфигурационный файл:

[filename](#)

sysvinit

ДОПОЛНИТЕЛЬНЫЕ НЕОБХОДИМЫЕ ФАЙЛЫ

`{{ patch.url}}`

ПОДГОТОВКА

Примените необходимый патч:

СБОРКА

УСТАНОВКА

bootscripts

УСТАНОВКА

8.78.3 SystemD

systemd

systemd - современная система инициализации от Red Hat. Она намного удобнее большинства известных систем инициализации, именно поэтому её использует большое число дистрибутивов.

ДОСТОИНСТВА SYSTEMD:

- агрессивная параллелизация загрузки сервисов, что позволяет ускорить время запуска системы и её полной загрузки;
- запуск сервисов по расписанию (аналог cron);
- быстрая смена корня (аналог chroot);
- простой и лаконичный синтаксис служб;
- контроль за каждой службой, анализ загрузки как каждого сервиса, так и всех их вместе.

НЕДОСТАТКИ SYSTEMD:

- не Unix Way. systemd - монолитная система, заменяющая собой не только систему инициализации, но и планировщик, менеджер сети, утилиту по смене корня системы, просмотрщик логов и пр, что не особо нужно многим пользователям;
- systemd требуется несколько больше ресурсов, чем его менее прожорливым товарищам, что так же не нравится любителям альтернативных систем инициализации.

systemd подходит для тех пользователей, которым важна простота работы со службами, юнитами и пр.

Создание пользователей и групп для systemd

systemd требует множество пользователей и групп для своей работы. Создайте их:

[filename](#)

systemd**ПОДГОТОВКА**

Удалите ненужную группу `render` из правил `udev`:

[filename](#)

НАСТРОЙКА

[filename](#)

ПРИ РАЗДЕЛЬНОЙ СТРУКТУРЕ КАТАЛОГОВ

Добавьте к `meson` ключи:

- `-Dkmod-path=/bin/kmod`
- `-Dmount-path=/bin/mount`
- `-Drootlibdir=/lib`
- `-Dsplit-usr=true`
- `-Dsulogin-path=/sbin/sulogin`
- `-Dsulogin-path=/sbin/sulogin`
- `-Dumount-path=/bin/umount`

СБОРКА

[filename](#)

УСТАНОВКА

[filename](#)

- Удалите ненужный каталог;
- Создайте файл `/etc/machine-id`, необходимый для `systemd-journald`;
- Настройте базовую целевую структуру;
- Отключите службу, которая, как известно, вызывает проблемы с системами, использующими конфигурацию сети, отличную от той, которая предоставляется `systemd-networkd`:

[filename](#)

ДЛЯ MULTILIB**Очистка**

[filename](#)

Настройка

[filename](#)

Сборка

[filename](#)

Установка

[filename](#)

d-bus

НАСТРОЙКА

filename

СБОРКА

filename

ТЕСТИРОВАНИЕ

filename

УСТАНОВКА

filenameСоздайте символическую ссылку, чтобы `systemd` и D-Bus использовали один и тот же файл `machine-id`:**filename**

ПРИ РАЗДЕЛЬНОЙ СТРУКТУРЕ КАТАЛОГОВ

filename

ДЛЯ MULTILIB

Очистка

filename

Настройка

filename

Сборка

filename

Установка

filename

procps-ng

НАСТРОЙКА

filename

СБОРКА

filename

ТЕСТИРОВАНИЕ

filename

?> Пять тестов, связанных с `pkill` дают сбои, так как они (тесты) не были обновлены.

УСТАНОВКА

filename

ПРИ РАЗДЕЛЬНОЙ СТРУКТУРЕ КАТАЛОГОВ

filename

util-linux**НАСТРОЙКА****filename**

Если вы собираете систему с раздельной структурой каталогов, уберите аргумент `--libdir=/usr/lib`!

СБОРКА**filename****ТЕСТИРОВАНИЕ**

!> Тестирование пакета от имени пользователя `root` может сломать Вашу систему. Для того чтобы этого не случилось, производите тесты от имени непrivилегированного пользователя. Для запуска тестов параметр `CONFIG_SCSI_DEBUG` для ядра должен быть доступен в текущей системе и должен быть собран в виде модуля. Также должны быть установлены некоторые другие пакеты из руководства extra. При желании этот тест может быть запущен после перезагрузки в завершенную систему LX4: `bash tests/run.sh --srcdir=$PWD --builddir=$PWD`

filename**УСТАНОВКА****filename****ДЛЯ MULTILIB****Очистка****filename****Настройка****filename****Сборка****filename****Установка****filename**

8.79 e2fsprogs

8.79.1 Настройка

[filename](#)

8.79.2 При раздельной структуре каталогов

Добавьте к скрипту `configure` ключ `--bindir=/bin`.

Значения параметров

`--enable-elf-shlibs` Это создает общие библиотеки, которые используются некоторыми программами пакета.

`--disable-*` Исключает установку библиотек `libuuid` и `libblkid`, службы `uuid` и `fsck` обертки, которые содержатся в пакете `Util-Linux` и являются более актуальными.

`--enable-symlink-install` `--enable-relative-symlinks` - использовать относительные символические ссылки вместо жестких.

8.79.3 Сборка

[filename](#)

8.79.4 Тестирование

[filename](#)

Тесты могут занять довольно продолжительное время на HDD (около 4 SBU). На SSD же это будет быстрее - 1,5 SBU.

?> Тест `m_rootdir_acl` может дать сбой.

8.79.5 Установка

[filename](#)

Пакет установит запакованный файл `.info` и не обновит системный файл `dir`. Распакуйте файл и обновите файл `dir`, выполнив следующую команду:

[filename](#)

При необходимости создайте и установите дополнительную документацию выполнив следующие команды:

[filename](#)

8.80 GRUB

8.80.1 О загрузчике

Инструкции по установке GRUB, наиболее популярного системного загрузчика, будут сильно отличаться в зависимости от того, используете вы UEFI или Legacy BIOS boot.

?> вы можете использовать другой загрузчик, например, LILO, Syslinux или systemd-boot.

8.80.2 EFI

Установка GRUB на UEFI

Новые компьютеры используют прошивку UEFI вместо традиционного BIOS. Обе эти программы – примеры ПО низкого уровня, запускающегося при старте компьютера перед тем, как загрузится операционная система. UEFI – более новое решение, оно поддерживает жёсткие диски большего объёма, быстрее загружается, а так же более безопасен.

В данном разделе пойдёт речь об установке загрузчика GRUB2 на UEFI.

!> Если у вас обычный BIOS, пропустите этот раздел, перейдя к [BIOS](#).

efivar

ДОПОЛНИТЕЛЬНЫЕ НЕОБХОДИМЫЕ ФАЙЛЫ

`{{ patch.url}}`

ПОДГОТОВКА

Примените патч, исправляющий ошибку сборки новыми версиями GCC:

СБОРКА

УСТАНОВКА

efibootmgr

ПОДГОТОВКА

Исправьте ошибку сборки:

СБОРКА

УСТАНОВКА

ЗНАЧЕНИЯ НОВЫХ КОМАНД

- `EFIDIR=LIN` - указывает имя подкаталога дистрибутива в `/boot/efi/EFI`. Это нужно указать явно.
- `EFI_LOADER=grubx64.efi` - указывает имя загрузчика EFI по умолчанию.
- `sbindir=/usr/bin` - установить `efibootmgr` в `/usr/bin`.

GRUB

ДОПОЛНИТЕЛЬНЫЕ НЕОБХОДИМЫЕ ФАЙЛЫ

<https://unifoundry.com/pub/unifont/unifont-13.0.06/font-builds/unifont-13.0.06.pcf.gz>

ПОДГОТОВКА

Установите шрифт, используемый grub:

НАСТРОЙКА**ЗНАЧЕНИЯ ПАРАМЕТРОВ**

`--disable-efiemu` - отключает установку большого и ненужного компонента

`--with-platform=efi` - использовать EFI

`--enable-grub-mkfont` - включает установку шрифтов

СБОРКА**УСТАНОВКА**

8.80.3 BIOS/Legacy

Установка GRUB в MBR для BIOS

BIOS — это Basic Input-Output system, базовая система ввода-вывода. Это программа низкого уровня, хранящаяся на чипе материнской платы вашего компьютера. BIOS загружается при включении компьютера и отвечает за пробуждение его аппаратных компонентов, проверяет, что они правильно работают, а потом запускает программу-загрузчик, запускающую операционную систему Linux, BSD, MacOS, Windows или любую другую, установленную у вас.

На экране настройки BIOS вы можете изменять множество параметров. Аппаратная конфигурация компьютера, системное время, порядок загрузки. Этот экран можно вызвать в начале загрузки компьютера по нажатию определённой клавиши — на разных компьютерах она разная, но часто используются клавиши Esc, F2, F10, Delete. Сохраняя настройку, вы сохраняете её в памяти материнской платы. При загрузке компьютера BIOS настроит его так, как указано в сохранённых настройках.

Перед загрузкой операционной системы BIOS проходит через POST, или Power-On Self Test, самотестирование после включения. Она проверяет корректность настройки аппаратного обеспечения и его работоспособность. Если что-то не так, на экране вы увидите серию сообщений об ошибках или услышите из системного блока загадочный писк. Что именно означают звуковые сигналы описано в инструкции к компьютеру.

При загрузке компьютера по окончанию POST BIOS ищет Master Boot Record, или MBR — главную загрузочную запись. Она хранится на загрузочном устройстве и используется для запуска загрузчика ОС.

В данном разделе пойдёт речь об установке GRUB в MBR. О загрузке компьютера на BIOS смотрите [здесь](#).

GRUB

НАСТРОЙКА

ЗНАЧЕНИЯ ПАРАМЕТРОВ

`--disable-efiemu` - отключает установку большого и ненужного компонента

СБОРКА

УСТАНОВКА

8.81 Очистка системы

Во время тестов может создаваться большое количество временных файлов. Удалите их:

```
1 rm -rf /tmp/*
```

Перезайдите в среду `chroot`:

```
1 logout
```

filename

Файлы `.la` потеряли свою актуальность и могут вызвать проблемы при обновлении. Удалите их:

```
1 find /usr/lib /usr/libexec /usr/lib32 -name *.la -delete
```

Удалите временный компилятор:

```
1 find /usr -depth -name $(uname -m)-lin-linux-gnu* | xargs rm -rf
```

Удалите кросс-компилятор:

```
1 rm -rf /tools
```

Удалите созданного для тестов пользователя:

```
1 userdel -r tester
```

Вы можете удалить ненужные символы из исполняемых файлов:

```
1 find /lib /usr/lib{,32} -type f -name *.a \
2     -exec strip --strip-debug {} ';' \
3
4 find /lib /usr/lib{,32} -type f -name *.so* ! -name *.dbg \
5     -exec strip --strip-unneeded {} ';' \
6
7 find /{bin,sbin} /usr/{bin,sbin,libexec} -type f \
8     -exec strip --strip-all {} ';'
```

9. Настройка системы

9.1 Настройка системы

В данном разделе будут даны инструкции по настройке ОС.

9.2 Создание файла fstab

При загрузке системы, исходя из данных в файле `/etc/fstab`, монтируются различные разделы и диски, в том числе - корневой раздел.

`Fstab` (сокр. от англ. file systems table) — один из конфигурационных файлов, который содержит информацию о различных файловых системах и устройствах хранения информации компьютера, описывает, как диск будет использоваться или как будет интегрирован в систему. Файл `/etc/fstab` делает возможным автоматическое монтирование определенных файловых систем, что особенно нужно при загрузке системы. Он содержит ряд строк, описывающих файловые системы, их точки монтирования и другие параметры.

Строки содержат, по порядку:

- устройство монтируемой файловой системы;
- точку монтирования;
- тип файловой системы;
- параметры монтирования;
- флаг для `dump`, утилиты создания резервных копий;
- порядок проверки для `fsck` (File System ChecK).

Здесь всегда есть запись о корневой файловой системе. Раздел `swap` является специальным, поэтому его не видно в древовидной структуре, и в поле точки монтирования для таких разделов всегда содержится ключевое слово `swap`.

9.2.1 Для systemd

`filename`

Замените `sdX` на нужное значение.

Если вы используете SysVInit, выполните:

```
1 echo "proc      /proc      proc    nosuid,noexec,nodev 0      0
2 sysfs     /sys      sysfs   nosuid,noexec,nodev 0      0
3 devpts    /dev/pts  devpts  gid=5,mode=620      0      0
4 tmpfs     /run      tmpfs   defaults      0      0
5 devtmpfs  /dev      devtmpfs mode=0755,nosuid  0      0" >> /etc/fstab
```

Для использования UEFI выполните:

```
1 echo "/dev/sdN  /boot/efi  vfat  umask=0077      0      0" >> /etc/fstab
```

Заменив `sdN` на нужное значение.

Для использования `swap` выполните:

```
1 echo "/dev/sdY  swap      swap    pri=1      0      0" >> /etc/fstab
```

Заменив `sdY` на нужное значение.

При необходимости можете добавить другие разделы в данный файл, руководствуясь примером выше.

Файловым системам MS-DOS или Windows (vfat, ntfs, smbfs, cifs, iso9660, udf) требуется специальный параметр `utf8`, чтобы символы, не входящие в ASCII, в именах файлов интерпретировались должным образом. Для языковых стандартов, отличных от UTF-8, значение `iocharset` должно быть установлено таким же, как набор символов языкового стандарта, настроенного таким образом, чтобы ядро его понимало. Это работает, если соответствующее определение набора символов (находится в `File systems -> Native Language Support` при настройке ядра) было скомпилировано в ядро или построено как модуль. Однако, если набор символов локали - UTF-8, соответствующий параметр `iocharset = utf8` сделает файловую систему чувствительной к регистру. Чтобы исправить это, используйте специальный параметр `utf8` вместо `iocharset = utf8` для локалей UTF-8. Параметр «`codepage`» также необходим для файловых систем `vfat` и `smbfs`. Он должен быть установлен

на номер кодовой страницы, используемый в MS-DOS в вашей стране. Например, чтобы смонтировать USB-накопители, пользователю ru_RU.KOI8-R потребуется следующее в части параметров его строки монтирования в `/etc/fstab` :

```
1 noauto,user,quiet,showexec,codepage=866,iocharset=koi8r
```

Соответствующий фрагмент опций для пользователей ru_RU.UTF-8:

```
1 noauto,user,quiet,showexec,codepage=866,utf8
```

Обратите внимание, что использование `iocharset` является значением по умолчанию для `iso8859-1` (что делает файловую систему нечувствительной к регистру), а опция `utf8` указывает ядру преобразовать имена файлов с использованием `UTF-8`, чтобы их можно было интерпретировать в локали `UTF-8`.

Можно сделать файловую систему `ext4` надежной при сбоях питания для некоторых типов жестких дисков. Для этого добавьте параметр монтирования `barrier=1` в соответствующую запись в `/etc/fstab`. Чтобы проверить, поддерживает ли диск этот параметр, запустите `hdparm` на соответствующем диске:

```
1 hdparm -I /dev/sda | grep NCQ
```

Если результат будет пустой - данная опция не поддерживается.

Мы настоятельно советуем использовать вместо метки раздела диска (например, `/dev/sda1`, `/dev/hdc2`, etc.) его UUID. Если метка диска, прописанного в `/etc/fstab` изменится, то могут возникнуть проблемы с загрузкой ОС, либо же она не загрузится вообще.

Узнать UUID для нужного раздела можно, выполнив:

```
1 blkid /dev/sdX
```

Заменив `sdX` на нужное значение, например, `sda2`.

Для того чтобы использовать вместо метки UUID, запись должна иметь следующий вид:

```
1 UUID=93c7b617-7558-4d1b-ab4b-a56880811037 / ext4 defaults 1 1
```

Т. е. вместо метки `/dev/sdX` используется UUID нужного раздела: `UUID="нужное_значение"`. Всё остальное без изменений.

!> После выполнения данных инструкций внимательно проверьте данный файл на наличие ошибок.

9.3 Создание файла /etc/shells

Некоторые программы требуют файл `/etc/shells`, в котором должны быть перечислены доступные командные оболочки.

Создайте его:

```
1  cat > /etc/shells << "EOF"
2  # Begin /etc/shells
3
4  /bin/sh
5  /bin/bash
6
7  # End /etc/shells
8  EOF
```

9.4 systemd

9.4.1 Настройка аппаратных часов (для systemd)

При загрузке считывается информация из аппаратных часов - CMOS. В них может стоять как местное время, так и универсальное (UTC). Программно нельзя определить, какой часовой пояс используют часы CMOS, однако вы можете выполнить команду `hwclock --localtime --show` и сравнить результат с местным временем. Если он не совпадает - ваши часы скорее всего используют UTC.

Если аппаратные часы используют местное время, создайте файл `/etc/adjtime`:

```
1  cat > /etc/adjtime << "EOF"
2  0.0 0 0.0
3  0
4  LOCAL
5  EOF
```

9.4.2 Настройка сети

Начиная с версии 209, в `systemd` имеется служба `systemd-networkd`, которую можно использовать для настройки сети. Кроме того, начиная с версии 213, разрешение имен DNS можно обрабатывать с помощью `systemd-resolved` вместо статического файла `/etc/resolv.conf`. Обе службы включены по умолчанию.

Файлы конфигурации для `systemd-networkd` (и `systemd-resolved`) можно разместить в `/usr/lib/systemd/network` или `/etc/systemd/network`. Файлы в `/etc/systemd/network` имеют более высокий приоритет, чем файлы в `/usr/lib/systemd/network`.

Существует три типа файлов конфигурации: файлы `.link`, `.netdev` и `.network`.

Именование сетевых устройств

`Udev` обычно назначает имена интерфейсов сетевых карт на основе физических характеристик системы, таких как `enp2s1`. Если вы не уверены, как называется ваш интерфейс, вы всегда можете запустить `ip link` после загрузки вашей системы.

Для большинства систем существует только один сетевой интерфейс для каждого типа подключения. Например, классическое имя интерфейса для проводного подключения - `eth0`. Беспроводное соединение обычно имеет имя `wifi0` или `wlan0`.

Если вы предпочитаете использовать классические или настраиваемые имена сетевых интерфейсов, есть три способа это сделать:

- Замаскируйте файл `.link` `udev` для политики по умолчанию:

```
1 ln -s /dev/null /etc/systemd/network/99-default.link
```

- Создайте схему именования вручную, например, присвоив интерфейсам что-то вроде `internet0`, `dmz0` или `lan0`. Для этого создайте файлы `.link` в `/etc/systemd/network/`, которые выберут явное имя или лучшую схему именования для ваших сетевых интерфейсов. Например:

```
1 cat > /etc/systemd/network/10-ether0.link << "EOF"
2 [Match]
3 # Change the MAC address as appropriate for your network device
4 MACAddress=12:34:45:78:90:AB
5
6 [Link]
7 Name=ether0
8 EOF
```

- В файле `/boot/grub/grub.cfg` передайте опцию `net.ifnames=0` в командной строке ядра.

Настройка DHCP

Приведенная ниже команда создаёт файл базовой конфигурации для настройки IPv4 DHCP:

```
1 cat > /etc/systemd/network/10-eth-dhcp.network << "EOF"
2 [Match]
3 Name=<network-device-name>
4
5 [Network]
6 DHCP=ipv4
7
8 [DHCP]
9 UseDomains=true
10 EOF
```

Настройка статического IP-адреса

Приведенная ниже команда создает базовый файл конфигурации для настройки статического IP-адреса (с использованием как `systemd-networkd`, так и `systemd-resolved`):

```
1 cat > /etc/systemd/network/10-eth-static.network << "EOF"
2 [Match]
3 Name=<network-device-name>
4
```

```

5 [Network]
6 Address=192.168.0.2/24
7 Gateway=192.168.0.1
8 DNS=192.168.0.1
9 Domains=<Your Domain Name>
10 EOF

```

Если у вас несколько DNS-серверов, можно добавить несколько записей DNS. (Замените ключ DNS в команде, расположенной выше)

```
1 DNS=192.168.0.1 8.8.8.8 8.8.4.4
```

Не включайте записи DNS или доменов, если вы собираетесь использовать статический файл `/etc/resolv.conf`.

Создание файла `/etc/resolv.conf`

Если система будет подключена к сети Интернет, ей потребуются некоторые средства разрешения доменных имен (DNS) для преобразования доменных имён в IP-адреса и наоборот. Лучше всего это достигается помещением IP-адреса DNS-сервера, доступного у интернет-провайдера или сетевого администратора, в `/etc/resolv.conf`.

НАСТРОЙКА SYSTEMD-RESOLVED

?> При использовании методов, несовместимых с `systemd-resolved` для настройки сетевых интерфейсов (например, `ppp` и т. д.), или при использовании любого типа локального преобразователя (например, `bind`, `dnsmasq`, `unbound` и т. д.) или любого другого программного обеспечения, которое генерирует `/etc/resolv.conf` (например, программа `resolvconf`, отличная от той, которая предоставляется `systemd`), службу `systemd-resolved` использовать не следует.

При использовании службы `systemd-resolved` для настройки DNS, будет создан файл `/run/systemd/resolve/resolv.conf`.

Создайте символьическую ссылку, чтобы использовать этот файл в каталоге `/etc`

```
1 ln -s /run/systemd/resolve/resolv.conf /etc/resolv.conf
```

СТАТИЧЕСКАЯ НАСТРОЙКА ФАЙЛА RESOLV.CONF

Если требуется статический файл `/etc/resolv.conf`, создайте его, выполнив следующую команду:

```

1 cat > /etc/resolv.conf << "EOF"
2 # Begin /etc/resolv.conf
3
4 domain <Ваше доменное имя>
5 nameserver <IP-адрес вашего основного сервера имен>
6 nameserver <IP-адрес вашего вторичного сервера имен>
7
8 # End /etc/resolv.conf
9 EOF

```

`domain` - можно опустить или заменить на `search`

Замените `<Ваше доменное имя>` IP-адресом DNS-сервера. Может присутствовать несколько записей (требования предусматривают наличия вторичных серверов для возможности восстановления). Если вам нужен только один DNS-сервер, удалите вторую строку `nameserver` из файла. IP-адрес также может быть маршрутизатором в локальной сети.

В качестве DNS можно указать адреса, предложенные ниже:

Yandex DNS IPv4 77.88.8.8 и 77.88.8.1

The Google Public IPv4 DNS адреса 8.8.8.8 и 8.8.4.4

The Google Public IPv6 DNS адреса 2001:4860:4860::8888 и 2001:4860:4860::8844

Настройка `/etc/hostname`

Во время процесса загрузки файл `/etc/hostname` используется для определения имени хоста системы.

Создайте файл `/etc/hostname` и введите имя хоста, запустив:

```
1 echo "MyHostName" > /etc/hostname
```

"MyHostName" необходимо заменить именем, присвоенным компьютеру. Не вводите здесь полное доменное имя (FQDN). Эта информация помещается в файл `/etc/hosts`.

Настройка `/etc/hosts`

Выберите полное доменное имя (FQDN) и возможные псевдонимы для использования в файле `/etc/hosts`. Если вы используете статические IP-адреса, вам также необходимо выбрать IP-адрес. Синтаксис записи файла `hosts`:

```
1 IP_address myhost.example.org aliases
```

Если компьютер не должен быть видимым для Интернета (существует зарегистрированный домен и действительный блок назначенных IP-адресов - у большинства пользователей этого нет), убедитесь, что IP-адрес находится в диапазоне IP-адресов частной сети. Допустимые диапазоны:

1	Диапазон сети	Префикс
2	10.0.0.1 - 10.255.255.254	8
3	172.x.0.1 - 172.x.255.254	16
4	192.168.y.1 - 192.168.y.254	24

x - может быть любым числом в диапазоне от 16 до 31

y - может быть любым числом в диапазоне от 0 до 255

Корректный локальный IP адрес может быть вида 192.168.1.1. FQDN, например mylin.example.org .

Даже если не используется сетевая карта, всё равно требуется действительное полное доменное имя. Это необходимо для правильной работы определенных программ, таких как MTA (почтовый сервер).

Создайте файл `/etc/hosts`

```
1 cat > /etc/hosts << "EOF"
2 # Begin /etc/hosts
3
4 127.0.0.1 localhost.localdomain localhost
5 127.0.1.1 <FQDN> <HOSTNAME>
6 <192.168.0.2> <FQDN> <HOSTNAME> [alias1] [alias2] ...
7 ::1 localhost ip6-localhost ip6-loopback
8 ff02::1 ip6-allnodes
9 ff02::2 ip6-allrouters
10
11 # End /etc/hosts
12 EOF
```

Значения <192.168.0.2> , <FQDN> и <HOSTNAME> необходимо изменить в соответствии с потребностями (если IP-адрес назначен сетевым / системным администратором, и оборудование будет подключено к существующей сети).

Необязательные псевдонимы можно опустить, а строку 192.168.0.2 можно не указывать, если вы используете соединение, настроенное с помощью DHCP или автоконфигурации IPv6.

Запись ::1 является эквивалентом IPv6 127.0.0.1 и представляет loopback интерфейс IPv6 .

127.0.1.1 - так называемый, «местный» от англ. local, или «локальный хост», по смыслу — это компьютер, зарезервированный специально для FQDN.

9.5 SysVInit

9.5.1 Настройка SysVInit

При загрузке `init` читает файл `/etc/inittab`. Создайте его:

```

1  cat > /etc/inittab << "EOF"
2  # Begin /etc/inittab
3
4  id:3:initdefault:
5
6  si::sysinit:/etc/rc.d/init.d/rc S
7
8  l0:0:wait:/etc/rc.d/init.d/rc 0
9  l1:1:wait:/etc/rc.d/init.d/rc 1
10 l2:2:wait:/etc/rc.d/init.d/rc 2
11 l3:3:wait:/etc/rc.d/init.d/rc 3
12 l4:4:wait:/etc/rc.d/init.d/rc 4
13 l5:5:wait:/etc/rc.d/init.d/rc 5
14 l6:6:wait:/etc/rc.d/init.d/rc 6
15
16 ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
17
18 su:S016:once:/sbin/sulogin
19
20 1:2345:respawn:/sbin/agetty --noclear tty1 9600
21 2:2345:respawn:/sbin/agetty tty2 9600
22 3:2345:respawn:/sbin/agetty tty3 9600
23 4:2345:respawn:/sbin/agetty tty4 9600
24 5:2345:respawn:/sbin/agetty tty5 9600
25 6:2345:respawn:/sbin/agetty tty6 9600
26
27 # End /etc/inittab
28 EOF

```

Настройка системного времени

При загрузке считывается информация из аппаратных часов - CMOS. В них может стоять как местное время, так и универсальное (UTC). Программно нельзя определить, какой часовой пояс используют часы CMOS, однако вы можете выполнить команду `hwclock --localtime --show` и сравнить результат с местным временем. Если он не совпадает - ваши часы скорее всего используют UTC.

Создайте файл, определяющий использует ли CMOS UTC (если нет - замените UTC=1 на UTC=0):

```

1  cat > /etc/sysconfig/clock << "EOF"
2  # Begin /etc/sysconfig/clock
3
4  UTC=1
5
6  # Set this to any options you might need to give to hwclock,
7  # such as machine hardware clock type for Alphas.
8  CLOCKPARAMS=
9
10 # End /etc/sysconfig/clock
11 EOF

```

Настройка клавиатуры в TTY

Настройка клавиатуры и шрифта консоли производится в файле `/etc/sysconfig/console`. Параметры в этом файле:

- `UNICODE` : при значении 1, yes или true переводит консоль в режим UTF-8, что важно для русского языка.
- `KEYMAP` : аргументы для программы `loadkeys`, которая загружает раскладки клавиатуры. Соответственно, значение этого параметра равно имени загружаемой раскладки.
- `FONT` : аргументы для программы `setfont`, которая устанавливает шрифт терминала. Соответственно, значение этого параметра есть имя шрифта.

Раскладки клавиатуры содержатся в каталоге `/usr/share/keymaps/`, а шрифт - в `/usr/share/consolefonts`. Оттуда берутся нужные файлы и указываются в файле `/etc/sysconfig/console`.

УСТАНОВКА РАСКЛАДКИ КЛАВИАТУРЫ

Для поиска нужной раскладки выполните:

```
1 find /usr/share/keymaps --type f
```

Для русской (на примере qwerty) это может быть:

- `ru` - переключение раскладки не установлено (либо же вовсе отсутствует)
- `ruwin_alt_sh-UTF-8` - переключение раскладки по Alt+Shift.
- `ruwin_cplk-UTF-8` - переключение раскладки по Caps Lock.
- `ruwin_ct_sh-UTF-8` - переключение раскладки по Ctrl+Shift.

Выберите из списка нужную и запишите в файл `/etc/sysconfig/console` как значение переменной `KEYMAP`. Значение не должно включать в себя путь до раскладки и её расширение.

Пример:

```
1 KEYMAP="ruwin_alt_sh-UTF-8"
```

УСТАНОВКА ШРИФТА КОНСОЛИ

Следующим шагом будет установка шрифта. Он должен поддерживать кириллицу. Все шрифты находятся в `/usr/share/consolefonts`. Выполните:

```
1 ls /usr/share/consolefonts
```

Из списка выберите нужный шрифт. Мы вам советуем `cyr-sun16`. Теперь запишите имя нужного шрифта в конфигурационный файл как значение переменной `FONT`. Оно не должно включать в себя путь до раскладки и её расширение.

Пример:

```
1 FONT="cyr-sun16"
```

Итоговый конфиг теперь выглядит так:

```
1 # Begin /etc/sysconfig/console
2
3 UNICODE="1"
4 KEYMAP="ruwin_alt_sh-UTF-8"
5 FONT="cyr-sun16"
6
7 # End /etc/sysconfig/console
```

10. Настройка и установка ядра

10.1 Настройка и установка ядра

Ядро Linux — основной компонент операционной системы, соответствующий стандартам POSIX. Именно ядро выступает промежуточным звеном между пользовательскими программами и оборудованием. При включении компьютера ядро - первая часть операционной системы, которая будет загружена. Ядро обнаруживает и инициализирует все компоненты оборудования компьютера, делает их доступными в виде дерева каталогов с файлами для доступа к ним программам.

10.1.1 Процесс установки

Процесс установки ядра состоит из нескольких этапов: настройка, компиляция и установка. Каждый этап будет максимально подробно разъяснён в следующих главах.

10.1.2 Настраиваемые параметры ядра

Ядро Linux разрабатывалось таким образом, чтобы всегда была возможность его максимально гибко настроить, адаптируя его к требуемым условиям эксплуатации и аппаратному окружению. Причём так, чтобы это было возможно динамически на готовой сборке ядра. Другими словами - можно в любой момент времени вносить корректирующие параметры, влияющие на работу как самого ядра, так и его отдельных компонентов.

Ознакомьтесь с материалами [Строение GNU/Linux. Часть 1](#), чтобы получить больше информации как про ядро, так и процесс загрузки.

10.2 Настройка

В данной главе подробно рассказано о настройке ядра из исходного кода.

10.2.1 Подготовка

Подготовьте пакет к компиляции, выполнив следующую команду:

```
1 make mrproper
```

Выполнение этой команды гарантирует, что дерево исходных кодов ядра будет абсолютно чистым. Разработчики ядра рекомендуют, чтобы эта команда выполнялась перед каждым процессом компиляции.

!> Обратите внимание что после распаковки пакета с исходным кодом не следует полагаться на его "чистоту".

10.2.2 Инструменты настройки параметров ядра

Конфигурация ядра хранится в файле `.config`. Именно этот файл следует отредактировать, указав необходимые опции, в соответствии с вашим оборудованием и предпочтениями.

Для наглядности и облегчения восприятия, настройку ядра можно произвести при помощи утилит, предоставляющих графический или псевдографический интерфейс:

- `make xconfig` – при использовании графической среды KDE
- `make gconfig` – при использовании графической среды GNOME
- `make menuconfig` – псевдографический режим
- `make config` – вариант настройки, выводящий запросы на задание значений каждого параметра ядра. (Не позволяет изменить уже заданные параметры)

Практически все варианты (за исключением последнего) позволяют получать краткую справку по каждому параметру, производить поиск нужного параметра (или раздела), добавлять в конфигурацию дополнительные компоненты, драйверы, а также показывают, каким образом конкретный компонент может быть сконфигурирован — как компонент, встроенный в ядро или как загружаемый модуль, а также поддерживает ли он вообще вариант компиляции в качестве загружаемого модуля.

10.2.3 Создание конфигурации

Хорошей отправной точкой для настройки ядра может стать запуск команды `make defconfig`.

Выполните команду:

```
1 make defconfig
```

Будет создана базовая конфигурация с настройками по умолчанию с учётом архитектуры машины. Параметры берутся из архитектурно-зависимых `defconfig` файлов.

10.2.4 Настройка параметров

!> Убедитесь в том, что вы **включили/отключили/указали** указанные ниже параметры настройки. В ином случае система может работать неправильно или вовсе не загрузится:

```
1 General setup -->
2   [ ] Auditing Support [CONFIG_AUDIT]
3   [*] Control Group support [CONFIG_CGROUPS]
4   [ ] Enable deprecated sysfs features to support old userspace tools [CONFIG_SYSFS_DEPRECATED]
5   [*] Configure standard kernel features (expert users) [CONFIG_EXPERT] --->
6     [*] open by fhandle syscalls [CONFIG_FHANDLE]
7 Processor type and features --->
8   [*] Enable seccomp to safely compute untrusted bytecode [CONFIG_SECCOMP]
9 Firmware Drivers --->
```

```

10 [*] Export DMI identification via sysfs to userspace [CONFIG_DMIID]
11 Networking support ---> [CONFIG_NET]
12 Networking options --->
13   <-> Packet socket [CONFIG_PACKET]
14   <-> The IPv6 Protocol ---> [CONFIG_IPV6]
15 Device Drivers --->
16 Generic Driver Options --->
17 [ ] Support for uevent helper [CONFIG_UEVENT_HELPER]
18 [*] Maintain a devtmpfs filesystem to mount at /dev [CONFIG_DEV TMPFS]
19 Firmware Loader --->
20 [ ] Enable the firmware sysfs fallback mechanism [CONFIG_FW LOADER USER_HELPER]
21 File systems --->
22 [*] Inotify support for userspace [CONFIG_INotify_USER]
23 Pseudo filesystems --->
24 [*] Tmpfs POSIX Access Control Lists [CONFIG_TMPFS_POSIX_ACL]

```

?> указанные ниже параметры используйте на своё усмотрение.

Поддержка файловых систем

Обратите внимание на раздел `File systems`. Включите поддержку требуемых файловых систем. Обязательно включите `The Extended 4 (ext4) filesystem` для её поддержки.

EFI

Если вы будете использовать EFI, необходимо обеспечить его поддержку:

```

1 Processor type and features --->
2   [*] EFI runtime service support [CONFIG_EFI]
3   [*] EFI stub support [CONFIG_EFI_STUB]
4 Firmware Drivers --->
5   EFI (Extensible Firmware Interface) Support --->
6     <-> EFI Variable Support via sysfs [CONFIG_EFI_VARS]
7     [*] Export efi runtime maps to sysfs [CONFIG_EFI_RUNTIME_MAP]
8   Enable the block layer --->
9     Partition Types --->
10    [*] Advanced partition selection [CONFIG_PARTITION_ADVANCED]
11    [*] EFI GUID Partition support [CONFIG_EFI_PARTITION]
12 Device Drivers --->
13   Graphics support --->
14     Frame buffer Devices --->
15       Support for frame buffer devices ---> [CONFIG_FB]
16       [*] EFI-based Framebuffer support [CONFIG_FB_EFI]
17     Console display driver support --->
18       [*] Framebuffer Console support [CONFIG_FRAMEBUFFER_CONSOLE]
19 File systems --->
20 Pseudo filesystems --->
21   <*/M> EFI Variable filesystem [CONFIG_EFIVAR_FS]

```

FUSE

FUSE (англ. `filesystem in userspace` — «файловая система в пользовательском пространстве») — свободный модуль для ядер Unix-подобных операционных систем, позволяющий разработчикам создавать новые типы файловых систем, доступные для монтирования пользователями без привилегий (прежде всего — виртуальных файловых систем); это достигается за счёт запуска кода файловой системы в пользовательском пространстве, в то время как модуль FUSE предоставляет связующее звено для актуальных интерфейсов ядра. С использованием средств FUSE разработаны, в частности, SSHFS, NTFS-3G, GlusterFS, ZFS.

```

1 File systems --->
2   <*/M> FUSE (Filesystem in Userspace) support [CONFIG_FUSE_FS]

```

Файловые системы Windows

```

1 File systems --->
2   <DOS/FAT/EXFAT/NT Filesystems --->
3     <*/M> MSDOS fs support [CONFIG_MS DOS_FS]
4     <*/M> VFAT (Windows-95) fs support [CONFIG_VFAT_FS]

```

XFS

XFS — высокопроизводительная 64-битная журналируемая файловая система, созданная компанией Silicon Graphics для собственной операционной системы IRIX. 1 мая 2001 года Silicon Graphics выпустила XFS под GNU General Public License

(Linux версия 2.2). XFS отличается от других файловых систем тем, что она изначально была рассчитана для использования на дисках большого объёма (более 2 терабайт).

```
1  File systems --->
2    <*>/M> XFS filesystem support [CONFIG_XFS_FS]
```

LVM

При необходимости включите поддержку LVM.

LVM — это метод распределения пространства жёсткого диска по логическим томам, размер которых можно легко менять, в отличие от разделов.

С LVM пространство жёсткого диска или набора дисков распределяется по физическим томам. Физический том не может располагаться более чем на одном диске.

Физические тома собираются в группы логических томов, за исключением раздела /boot. Раздел /boot не может находиться в группе логических томов, так как в этом случае загрузчику не удастся его прочитать. Если корневой раздел (/) находится на логическом томе, создайте отдельный раздел /boot вне группы томов.

```
1  Device Drivers --->
2    [*] Multiple devices driver support (RAID and LVM) ---> [CONFIG_MD]
3      <*>/M> Device mapper support [CONFIG_BLK_DEV_DM]
4      <*>/M> Crypt target support [CONFIG_DM_CRYPT]
5      <*>/M> Snapshot target [CONFIG_DM_SNAPSHOT]
6      <*>/M> Thin provisioning target [CONFIG_DM_THIN_PROVISIONING]
7      <*>/M> Mirror target [CONFIG_DM_MIRROR]
8  Kernel hacking --->
9    Generic Kernel Debugging Instruments --->
10   [*] Magic SysRq key [CONFIG_MAGIC_SYSRQ]
```

Cryptographic API

Cryptographic API предлагает богатый набор криптографических шифров, а также другие механизмы и методы преобразования данных для их вызова. Cryptographic API предоставляет различные вызовы API для следующих типов шифров:

- Симметричные шифры
- Шифры AEAD
- Дайджест сообщения, включая дайджест сообщения с ключом
- Генерация случайных чисел
- Интерфейс пользовательского пространства

```
1  Device Drivers --->
2    [*] Multiple devices driver support (RAID and LVM) ---> [CONFIG_MD]
3      <*>/M> Device mapper support [CONFIG_BLK_DEV_DM]
4      <*>/M> Crypt target support [CONFIG_DM_CRYPT]
5
6  Cryptographic API --->
7    <*>/M> XTS support [CONFIG_CRYPTO_XTS]
8    <*>/M> SHA224 and SHA256 digest algorithm [CONFIG_CRYPTO_SHA256]
9    <*>/M> AES cipher algorithms [CONFIG_CRYPTO_AES]
10   <*>/M> User-space interface for symmetric key cipher algorithms [CONFIG_CRYPTO_USER_API_SKCIPHER]
11
12  For tests:
13    <*>/M> Twofish cipher algorithm [CONFIG_CRYPTO_TWOFISH]
```

Драйверы устройств

В разделе **Device Drivers** - нужно пройтись по разделам и включить драйвера для своего оборудования - нестандартные жёсткие диски, мышки, USB устройства, веб-камеры, Bluetooth, Wi-Fi адаптеры, принтеры и т. д.

Посмотреть, какое оборудование подключено к вашей системе можно командой:

```
1  lspci
```

10.2.5 Сборка

Когда все параметры настроены, можно приступать к сборке.

Выполните команду:

```
1 make
```

?> Обратите внимание, что процесс сборки ядра может проходить длительное время (от 4.4 до 66.0 SBU). Это во многом зависит от установленных параметров конфигурации.

Установите модули ядра:

```
1 make modules_install
```

10.2.6 Установка

После завершения сборки необходимо выполнить еще несколько шагов. Некоторые файлы должны быть скопированы в каталог `/boot`.

Путь к образу ядра зависит от используемой платформы. Имя файла, указанное ниже, может иметь произвольное наименование, на ваш вкус, но имя файла должно начинаться с `vmlinuz` для обеспечения совместимости автоматической настройки процесса загрузки. При выполнении следующей команды будет считаться, что используется архитектура x86:

```
cp -iv arch/x86/boot/bzImage /boot/vmlinuz-{{ package.version }}-my-kernel
```

`System.map` файл, внутри которого находится символьная таблица адресов функций и процедур, используемых ядром операционной системы Linux. В этой таблице перечислены имена переменных и функций и их адреса в памяти компьютера. Эта таблица весьма полезна при отладке ядра в случае Kernel panic или Linux oops. `System.map` генерируется при компиляции ядра. Выполните следующую команду для установки `System.map`:

```
cp -iv System.map /boot/System.map-{{ package.version }}
```

Файл конфигурации ядра `.config`, полученный в результате настройки `make menuconfig` содержит в себе все опции конфигурации скомпилированного ядра. Хорошей идеей будет оставить этот файл для будущей работы:

```
cp -iv .config /boot/config-{{ package.version }}
```

Для облегчения обновления ядра создайте символьическую ссылку:

```
ln -svf vmlinuz-{{ package.version }}-my-kernel /boot/vmlinuz
```

При обновлении ядра будет достаточно установить новую версию и обновить символьическую ссылку.

10.2.7 Установка документации

Установите документацию, если она необходима

```
install -d /usr/share/doc/linux-{{ package.version }}
cp -rv Documentation/* /usr/share/doc/linux-{{ package.version }}
```

10.2.8 Настройка каталога с исходным кодом

Важно отметить, что файлы в каталоге исходных кодов ядра не принадлежат пользователю `root`. Всякий раз, когда пакет распаковывается от пользователя `root` (как это и выполнялось внутри среды `chroot`), файлы имеют те идентификаторы пользователя и группы, которые были назначены при распаковке. Обычно это не вызывает проблем для других устанавливаемых пакетов, так как каталог с исходными кодами удаляется после установки пакета. Однако исходный код ядра Linux часто сохраняется в течение длительного времени. Из-за этого существует вероятность того, что идентификатор пользователя, используемый при распаковке, будет назначен другому пользователю. В таком случае этот пользователь будет иметь доступ на запись в этот каталог.

?> Во многих случаях конфигурация ядра должна быть обновлена для пакетов, которые будут установлены вами позднее. В отличие от других пакетов удалять дерево исходного кода ядра не требуется после компиляции и установки.

Если вы планируете оставить каталог с исходным кодом ядра, выполните команду:

```
chown -R 0:0 /usr/src/linux-{{ package.version }}
```

!> Заголовочные файлы, расположенные в системном каталоге `/usr/include`, должны всегда быть те, которые использовались при компиляции `Glibc`. Их никогда не следует заменять на чистые заголовочные файлы ядра или любые другие подготовленные заголовочные файлы.

10.2.9 Настройка порядка загрузки модулей Linux

Обычно модули Linux загружаются автоматически, но иногда требуется определённый порядок. Программа, которая загружает модули, `modprobe` или `insmod`, использует файл `/etc/modprobe.d/usb.conf` как раз для этой цели. Этот файл должен быть создан так, что, если USB-драйверы (`ehci_hcd`, `ohci_hcd` и `uhci_hcd`) были созданы в виде модулей, то они будут загружены в требуемом порядке: `ehci_hcd` должен быть загружен до `ohci_hcd` и `uhci_hcd` для того, чтобы избежать предупреждений во время загрузки.

Создайте новый файл `/etc/modprobe.d/usb.conf`, выполнив следующую команду:

[filename](#)

10.3 О прошивках

Многие драйвера в ядре Linux (например, для видеокарт и сетевых адаптеров) требуют маленькие proprietарные компоненты - firmware (прошивки).

10.3.1 Определение необходимых прошивок

Для того чтобы выяснить, какие прошивки нужны - загрузитесь в операционную систему (инструкции о том, как это сделать даны в следующих 2 разделах) и выполните команду `dmesg`. Если для вашего ядра требуются какие либо прошивки - вы увидите связанные с этим ошибки.

?> Если система не загружается из-за ошибки, связанной с видео драйвером - поищите в интернете, какие драйвера необходимы для вашего видео адаптера.

10.3.2 Установка прошивок

Их можно установить двумя способами:

Первый - скомпилировать требующий прошивку драйвер как модуль и поместить её в `/lib/firmware`.

Второй - задать опцию `CONFIG_EXTRA_FIRMWARE`. В ней должны быть перечислены через пробел все прошивки, которые необходимо включить в ядро. Можно изменить путь поиска включенных в опцию `CONFIG_EXTRA_FIRMWARE` прошивок, задав опцию `CONFIG_EXTRA_FIRMWARE_DIR` (по умолчанию `/lib/firmware`)

Например:

```
1 CONFIG_EXTRA_FIRMWARE="amdgpu/picasso_asd.bin amdgpu/picasso_gpu_info.bin amdgpu/picasso_mec2.bin amdgpu/picasso_pfp.bin amdgpu/picasso_rlc.bin amdgpu/picasso_ta.bin amdgpu/picasso_ce.bin amdgpu/picasso_me.bin amdgpu/picasso_mec.bin amdgpu/picasso_rlc_am4.bin amdgpu/picasso_sdma.bin amdgpu/picasso_vcn.bin"
2 CONFIG_EXTRA_FIRMWARE_DIR="/lib/firmware"
```

В этом случае подключаются firmware для видеокарт AMDGPU семейства picasso (например, Vega 8). Они в таком случае находятся в `/lib/firmware/amdgpu/`

Найти и загрузить большую часть прошивок можно здесь <https://git.kernel.org/pub/scm/linux/kernel/git/firmware/linux-firmware.git/tree/>

11. Делаем систему загрузочной

11.1 Делаем систему загрузочной

В данной главе содержится информация о том, как обеспечить возможность созданной Linux системе загрузиться.

11.2 Создание загрузочной системы EFI

⚠ Warning

Отключите `secure boot`! На данный момент эта технология не поддерживается. Чтобы настроить процесс загрузки с помощью GRUB для UEFI, необходимо отключить её в интерфейсе конфигурации прошивки. Прочтите документацию, предоставленную производителем вашей системы, чтобы узнать, как это сделать.

Убедитесь, что вы не пропустили раздел по настройке ядра, для поддержки [EFI](#).

11.2.1 Поиск, или создание системного раздела EFI

В системе на основе EFI загрузчики устанавливаются в специальный раздел FAT32, называемый системным разделом EFI (ESP). Если ваша система поддерживает EFI и предустановлен дистрибутив Linux и (или) Windows, скорее всего, ESP уже создан. Посмотрите все разделы на вашем жёстком диске (замените `sda` на нужное устройство):

```
1  fdisk -l /dev/sda
```

Столбец `ESP` `type` должен быть `EFI System`.

Например:

```
1  Устр-во    начало    Конец    Секторы Размер Тип
2  /dev/sda1    4096    618495    614400  300M EFI
3  /dev/sda2    618496  268430084 267811589 127,76 Файловая система Linux
```

Если система или жёсткий диск новые, или если вы впервые устанавливаете ОС, загружаемую через UEFI, ESP может не существовать. В этом случае создайте новый раздел, создайте на нем файловую систему `vfat` и установите тип раздела `EFI system`.

🐞 Bug

Некоторые (старые) реализации UEFI могут требовать, чтобы ESP был первым разделом на диске.

Создайте точку монтирования для `ESP` и смонтируйте ее (замените `sda1` на соответствующий `ESP`):

```
1  mkdir -p /boot/efi &&
2  mount -v -t vfat /dev/sda1 /boot/efi
```

Добавьте запись для `ESP` в `/etc/fstab`, чтобы он автоматически монтировался во время загрузки системы:

```
1  cat >> /etc/fstab << EOF
2  /dev/sda1 /boot/efi vfat defaults 0 1
3  EOF
```

11.2.2 Монтирование EFI Variable File System

Для установки GRUB на UEFI необходимо смонтировать файловую систему EFI Variable, `efivarfs`. Если она еще не была смонтирована ранее, выполните команду:

```
1  mountpoint /sys/firmware/efi/efivars || mount -v -t efivarfs efivarfs /sys/firmware/efi/efivars
```

Добавьте запись для `efivarfs` в `/etc/fstab`, чтобы она автоматически монтировалась во время загрузки системы:

```
1  cat >> /etc/fstab << EOF
2  efivarfs /sys/firmware/efi/efivars efivarfs defaults 0 0
3  EOF
```

Note

Если система не загружается с UEFI, каталог `/sys/firmware/efi` будет отсутствовать. В этом случае вы должны загрузить систему в режиме UEFI с аварийным загрузочным диском.

11.2.3 Настройка

В системах на основе UEFI GRUB работает устанавливая приложение EFI (особый вид исполняемого файла) в `/boot/efi/EFI/[id] sizes/grubx64.efi`, где `/boot/efi` - точка монтирования ESP, а `[id]` заменяется идентификатором, указанным в командной строке `grub-install`. GRUB создаст запись в переменных EFI, содержащую путь `EFI/[id]/grubx64.efi`, чтобы прошивка EFI могла найти `grubx64.efi` и загрузить его.

`grubx64.efi` очень легкий (136 Кб), поэтому он не будет занимать много места в ESP. Типичный размер ESP составляет 100 Мб (для диспетчера загрузки Windows, который использует около 50 Мб в ESP). Как только `grubx64.efi` загружен прошивкой, он загрузит модули GRUB в загрузочный раздел. Расположение по умолчанию - `/boot/grub`.

Установите файлы GRUB в `/boot/efi/EFI/LFS/grubx64.efi` и `/boot/grub`. Затем настройте загрузочную запись в переменных EFI:

```
1 grub-install --bootloader-id=LIN --recheck
```

Если установка прошла успешно, вывод должен быть:

```
1 Installing for x86_64-efi platform.
2 Installation finished. No error reported.
```

Запустите `efibootmgr`, чтобы ещё раз проверить конфигурацию загрузки EFI.

```
1 efibootmgr
```

Пример вывода:

```
1 BootCurrent: 0000
2 Timeout: 1 seconds
3 BootOrder: 0005,0000,0002,0001,0003,0004
4 Boot0000* ARCH
5 Boot0001* UEFI:CD/DVD Drive
6 Boot0002* Windows Boot Manager
7 Boot0003* UEFI:Removable Device
8 Boot0004* UEFI:Network Device
9 Boot0005* LIN
```

Обратите внимание, что `0005` является первым в `BootOrder`, а `Boot0005` - это `LIN`. Это означает, что при следующей загрузке системы будет использоваться версия GRUB, установленная в `LIN`.

11.3 Создание файла конфигурации GRUB

Создайте `/boot/grub/grub.cfg` для настройки меню загрузки GRUB:

```
1 cat > /boot/grub/grub.cfg << EOF
2 # Begin /boot/grub/grub.cfg
3 set default=0
4 set timeout=5
5
6 insmod part_gpt
7 insmod ext2
8 set root=(hd0,2)
9
10 if loadfont /boot/grub/fonts/unicode.pf2; then
11   set gfxmode=auto
12   insmod all_video
13   terminal_output gfxterm
14 fi
15
16 menuentry "GNU/Linux, Linux 5.10.17-1fs-10.1" {
17   linux /boot/vmlinuz root=/dev/sda2 ro
18 }
19
20 menuentry "Firmware Setup" {
21   fwsetup
```

```

22 }
23 EOF

```

(hd0,2) , sda2 следует заменить в соответствии с вашей конфигурацией.

Note

Для GRUB файлы используются относительно раздела. Если вы использовали отдельный раздел `/boot`, удалите `/boot` из указанных выше путей (к ядру и к `unicode.pf2`). Вам также нужно будет изменить строку корневого раздела, чтобы она указывала на загрузочный раздел.

11.3.1 Загрузка вместе с Windows

Добавьте запись в файл конфигурации `grub.cfg` :

```

1  cat >> /boot/grub/grub.cfg << EOF
2  # Begin Windows addition
3
4  menuentry "Windows 10" {
5      insmod fat
6      insmod chain
7      set root=(hd0,1)
8      chainloader /EFI/Microsoft/Boot/bootmgfw.efi
9  }
10 EOF

```

(hd0,1) следует заменить назначенным GRUB именем для ESP. Директива `chainloader` может использоваться, чтобы указать GRUB запустить другой исполняемый файл EFI, в данном случае диспетчер загрузки Windows. вы можете поместить больше используемых инструментов в исполняемом формате EFI (например, оболочку EFI) в ESP и создать для них записи GRUB.

11.4 Создание загрузочной системы Legacy Boot MBR

⚠ Warning

Если ваша система поддерживает UEFI, вам следует пропустить эту страницу и настроить GRUB с поддержкой UEFI.

⚡ Danger

Неправильная настройка GRUB может сделать вашу систему неработоспособной без альтернативного загрузочного устройства, такого как CD-ROM или загрузочный USB-накопитель. Этот раздел не требуется для загрузки вашей созданной системы. вы можете просто изменить свой текущий загрузчик, например, GRUB Legacy, GRUB2 или LILO.

11.4.1 Соглашения о наименованиях GRUB

GRUB использует собственную структуру именования дисков и разделов в виде (`hdn, m`), где `n` - номер жёсткого диска, а `m` - номер раздела. Номер жёсткого диска начинается с нуля, но номер раздела начинается с единицы для обычных разделов и пяти для расширенных разделов.

Например, раздел `sda1` - это (`hd0,1`) для GRUB, а `sdb3` - (`hd1,3`). В отличие от Linux, GRUB не считает приводы CD-ROM жёсткими дисками. Например, если вы используете компакт-диск на `hdb` и второй диск на `hdc` , этот второй диск всё равно будет (`hd1`).

11.4.2 Настройка

GRUB записывает данные на первую физическую дорожку диска. Эта область не является частью какой-либо файловой системы. Программы получают доступ к модулям GRUB в загрузочном разделе. Расположение по умолчанию - `/boot/grub/` .

Расположение загрузочного раздела - это выбор пользователя, который влияет на конфигурацию. Одна из рекомендаций - создать отдельный небольшой (рекомендуемый размер 200 МБ) раздел только для загрузочной информации. Таким образом, любая система может получить доступ к одним и тем же файлам загрузки. Если вы решите это сделать, вам нужно будет смонтировать отдельный раздел, переместить все файлы в текущем каталоге `/boot` (например, ядро Linux, которое вы только что создали в предыдущем разделе) в новый раздел. Затем вам нужно будет размонтировать раздел и перемонтировать его как `/boot` . Если вы это сделаете, обязательно обновите `/etc/fstab` .

Использование с текущим разделом будет работать, но настройка для нескольких систем будет сложнее.

Используя приведенную выше информацию, определите подходящее обозначение для корневого раздела (или загрузочного раздела, если используется отдельный). В следующем примере предполагается, что корневой (или отдельный загрузочный) раздел - это `sda2` .

Установите файлы GRUB в `/boot/grub/` и настройте загрузочную дорожку:

⚠ Warning

Следующая команда перезапишет текущий загрузчик. Не запускайте команду, если это нежелательно, например, при использовании стороннего диспетчера загрузки для управления основной загрузочной записью (MBR).

✍ Note

Если система была загружена с использованием UEFI, `grub-install` попытается установить файлы для цели `x86_64-efi` , но эти файлы не были установлены. Если это так, добавьте `--target i386-pc` к команде.

```
1  grub-install /dev/sda
```

11.4.3 Создание конфигурационного файла

Danger

Существует команда `grub-mkconfig`, которая может автоматически записывать файл конфигурации. Она использует набор скриптов в `/etc/grub.d/` и уничтожит любые сделанные вами настройки. Эти сценарии предназначены в первую очередь для дистрибутивов без исходного кода и не рекомендуются к использованию. Если вы установите коммерческий дистрибутив Linux, есть большая вероятность, что эта программа будет запущена. Обязательно сделайте резервную копию файла `grub.cfg`.

Создайте файл `/boot/grub/grub.cfg`:

```
1  cat > /boot/grub/grub.cfg << "EOF"
2  # Begin /boot/grub/grub.cfg
3  set default=0
4  set timeout=5
5
6  insmod ext2
7  set root=(hd0,2)
8
9  menuentry "GNU/Linux, Linux" {
10    linux /boot/vmlinuz root=/dev/sda2 ro
11  }
12 EOF
```

Note

Для GRUB файлы ядра относятся к используемому разделу. Если вы использовали отдельный раздел `/boot`, удалите `/boot` из указанной выше строки `linux`. Вам также нужно будет изменить строку установленного корня, чтобы она указывала на загрузочный раздел.

GRUB - чрезвычайно мощная программа, которая предоставляет огромное количество вариантов загрузки с самых разных устройств, операционных систем и типов разделов. Существует также множество опций для настройки, таких как графические заставки, воспроизведение звуков, ввод мыши и т.д.

12. Заключительная часть

12.1 Заключительная часть

Примите поздравления! Ваша базовая Linux система настроена и готова к работе! Прежде чем выполнить загрузку, выполните некоторые заключительные шаги.

12.1.1 Создание файла /etc/os-release

Файл необходим для идентификации операционной системы. Он необходим при использовании `systemd`, а также для некоторых графических оболочек.

Базовый формат файла `os-release` - это список назначений переменных, совместимых с оболочкой, разделенных новой строкой. Можно получить конфигурацию из сценариев оболочки, однако, помимо простого назначения переменных, функции оболочки не поддерживаются (это означает, что расширение переменных явно не поддерживается), что позволяет приложениям читать файл без реализации механизма выполнения, совместимого с оболочкой. Значения присвоения переменных должны быть заключены в двойные или одинарные кавычки, если они включают пробелы, точки с запятой или другие специальные символы за пределами A - Z, a - z, 0-9. Специальные символы оболочки («\$», кавычки, обратная косая черта) должны быть экранированы обратной косой чертой в соответствии со стилем оболочки. Все строки должны быть в формате UTF-8, и нельзя использовать непечатаемые символы. Объединение нескольких строк в индивидуальных кавычках не поддерживается. Строки, начинающиеся с "#", игнорируются как комментарии. Пустые строки разрешены и игнорируются.

Создайте файл, выполнив команду:

```
cat > /etc/os-release << "EOF"
NAME="MyLinux"
VERSION="{{ book.revision }}"
ID=mylinux
PRETTY_NAME="MyLinux {{ book.revision }}"
VERSION_CODENAME="MyLinuxCodeName"
EOF
```

Параметры идентификации

NAME

Строка, идентифицирующая операционную систему, без компонента версии и подходящая для представления пользователю. Если не установлен, по умолчанию используется «NAME = Linux». Пример: `NAME=Fedora` или `NAME="Debian GNU/Linux"`.

VERSION

Строка, идентифицирующая версию операционной системы, исключая любую информацию об имени ОС, возможно, включая кодовое имя выпуска, и подходящая для представления пользователю. Это поле не является обязательным. Пример: `VERSION=17` или `VERSION=17 (Muscular miracle)`.

ID

Строка в нижнем регистре (без пробелов и других символов, кроме 0-9, a - z, «.», «_» И «-»), идентифицирующая операционную систему, исключая любую информацию о версии и подходящая для обработки скриптами или использования в сгенерированных именах файлов. Если не установлен, по умолчанию используется `ID=linux`.

?> Пример: `ID = fedora` или `ID = debian`.

ID_LIKE

Список идентификаторов операционных систем, разделенных пробелами, в том же синтаксисе, что и параметр `ID =`. В нем должны быть перечислены идентификаторы операционных систем, которые тесно связаны с локальной операционной системой в отношении упаковки и программных интерфейсов, например, перечисление одного или нескольких

идентификаторов ОС, производными от которых является локальная ОС. Обычно операционная система должна перечислять только другие идентификаторы ОС, производными от которых она сама является, а не какие-либо производные от нее операционные системы, хотя возможны симметричные отношения. Скрипты сборки и т.п. должны проверять эту переменную, если им нужно идентифицировать локальную операционную систему, а значение ID = не распознается. Операционные системы следует перечислять в порядке близости локальной операционной системы к перечисленным, начиная с ближайшей. Это поле не является обязательным. Пример: для операционной системы с ID = centos подходящим будет присвоение ID_LIKE="rhel fedora". Для операционной системы с ID = ubuntu подходит значение ID_LIKE = debian .

VERSION_CODENAME

Строка в нижнем регистре (без пробелов и других символов, кроме 0–9, a – z, «.», «_» И «-»), идентифицирующая кодовое имя выпуска операционной системы, исключая любую информацию об имени ОС или версию выпуска, и подходящая для обработки скриптами или использования в сгенерированных именах файлов. Это поле является необязательным и может быть реализовано не во всех системах.

?> Примеры: VERSION_CODENAME = buster , VERSION_CODENAME = xenial .

VERSION_ID

Строка в нижнем регистре (в основном числовая, без пробелов и других символов, кроме 0–9, a – z, «.», «_» И «-»), идентифицирующая версию операционной системы, исключая любую информацию об имени ОС или код выпуска. Подходит для обработки скриптами или использования в сгенерированных именах файлов. Это поле не является обязательным.

?> Пример: VERSION_ID = 34 или VERSION_ID = 21.04 .

PRETTY_NAME

Красивое название операционной системы в формате, удобном для представления пользователю. Может содержать или не содержать кодовое название выпуска или версию ОС, в зависимости от обстоятельств. Если не установлен, по умолчанию используется PRETTY_NAME =" Linux " .

?> Пример: PRETTY_NAME =" Fedora 34 (Workstation Edition) " .

ANSI_COLOR

Предлагаемый цвет презентации при отображении названия ОС в консоли. Это должно быть указано как строка, подходящая для включения в escape-код ESC [m ANSI / ECMA-48 для установки графического представления. Это поле не является обязательным. Пример: "ANSI_COLOR =" 0; 31 "" для красного, "ANSI_COLOR =" 1; 34 "" для голубого или "ANSI_COLOR =" 0; 38; 2; 60; 110; 180 "" для синего цвета Fedora.

CPE_NAME

Имя CPE для операционной системы в синтаксисе привязки URI в соответствии со спецификацией перечисления Common Platform, предложенной NIST. Это поле не является обязательным.

?> Пример: CPE_NAME =" cpe: / o: fedoraproject: fedora: 34 " .

HOME_URL, DOCUMENTATION_URL, SUPPORT_URL, BUG_REPORT_URL, PRIVACY_POLICY_URL

Ссылки на ресурсы в Интернете, связанные с операционной системой. HOME_URL = должен ссылаться на домашнюю страницу операционной системы или, альтернативно, на домашнюю страницу конкретной версии операционной системы. DOCUMENTATION_URL = следует ссылаться на главную страницу документации для этой операционной системы. SUPPORT_URL = следует ссылаться на главную страницу поддержки операционной системы, если таковая имеется. Это в первую очередь предназначено для операционных систем, для которых поставщики предоставляют поддержку. BUG_REPORT_URL = должен ссылаться на главную страницу отчетов об ошибках для операционной системы, если таковые имеются. Это в первую очередь предназначено для операционных систем, которые полагаются на обеспечение качества сообщества. PRIVACY_POLICY_URL = должен ссылаться на главную страницу политики конфиденциальности для операционной системы, если таковая имеется. Эти параметры являются необязательными, и обычно предоставляется только некоторые из этих параметров. Эти URL-адреса предназначены для отображения в пользовательских интерфейсах «Об этой системе» за ссылками с такими заголовками, как «Об этой операционной системе», «Получить поддержку», «Сообщить об ошибке»

или «Политика конфиденциальности». Значения должны быть в формате RFC3986 и должны быть URL-адресами «`http:`» или «`https:`» и, возможно, «`mailto:`» или «`tel:`». В каждой настройке должен быть указан только один URL. Если необходимо указать несколько ресурсов, рекомендуется предоставить целевую страницу в Интернете, связывающую все доступные ресурсы.

?> Пример: `HOME_URL = "https://fedoraproject.org/ "`.

BUILD_ID

Строка, однозначно определяющая образ системы, используемый в качестве источника для распределения (он не обновляется с обновлениями системы). Поле может быть идентичным для разных `VERSION_ID`, поскольку `BUILD_ID` - это только уникальный идентификатор для конкретной версии. В дистрибутивах, которые выпускают каждое обновление как новую версию, потребуется использовать только `VERSION_ID`, поскольку каждая сборка уже отличается на основе `VERSION_ID`. Это поле не является обязательным.

?> Пример: `BUILD_ID =« 2013-03-20.3 »` или `BUILD_ID = 201303203`.

VARIANT

Строка, идентифицирующая конкретный вариант или редакцию операционной системы, подходящую для представления пользователю. Это поле может использоваться для информирования пользователя о том, что конфигурация этой системы подчиняется определённому расходящемуся набору правил или настройкам конфигурации по умолчанию. Это поле является необязательным и может быть реализовано не во всех системах. Примеры: `VARIANT = "Server Edition "`, `VARIANT = "Smart Refrigerator Edition "`

!> Примечание: это поле предназначено только для отображения. Поле `VARIANT_ID` следует использовать для принятия программных решений.

VARIANT_ID

Строка в нижнем регистре (без пробелов и других символов, кроме 0-9, a - z, «.», «_» И «-»), определяющая конкретный вариант или выпуск операционной системы. Это может быть интерпретировано другими пакетами для определения расходящейся конфигурации по умолчанию. Это поле является необязательным и может быть реализовано не во всех системах.

?> Примеры: `VARIANT_ID = server`, `VARIANT_ID = embedded`.

LOGO

Строка, определяющая имя значка, как определено в Спецификации темы значков freedesktop.org. Это может использоваться графическими приложениями для отображения логотипа операционной системы или дистрибутора. Это поле является необязательным и может быть реализовано не во всех системах.

?> Пример: `LOGO=fedora-logo-icon`.

DEFAULT_HOSTNAME

Строка, определяющая имя хоста, если имя хоста отсутствует и никакой другой источник конфигурации не указывает имя хоста. Должна быть либо одна метка DNS (строка, состоящая из 7-битных символов нижнего регистра ASCII и без пробелов и точек, ограниченная форматом, разрешённым для меток имен доменов DNS), либо последовательность таких меток, разделённых одиночными точками, которые образуют действительное полное доменное имя DNS. Имя хоста должно состоять не более чем из 64 символов, что является ограничением Linux (DNS допускает более длинные имена).

SYSEXT_LEVEL

Строка в нижнем регистре (в основном числовая, без пробелов и других символов, кроме 0-9, a - z, «.», «_» И «-»), определяющая уровень поддержки расширений операционной системы, чтобы указать, какие образы расширений поддерживается.

?> Пример: `SYSEXT_LEVEL = 2` или `SYSEXT_LEVEL = 15,14`.

12.1.2 Создание файла /etc/lsb-release

Это файл, в который некоторые дистрибутивы Linux помещают информацию для использования старыми программами. Однако, этот файл не является частью стандарта LSB.

Создайте файл, выполнив команду:

```
cat > /etc/lsb-release << "EOF"
DISTRIB_ID="MyLinux"
DISTRIB_RELEASE="{{ book.revision }}"
DISTRIB_CODENAME="mylinux"
DISTRIB_DESCRIPTION="MyLinux"
EOF
```

Если система совместима с LSB, файл `/etc/lsb-release` должен содержать поле `LSB_VERSION`. Значение поля должно представлять собой список поддерживаемых версий модулей, разделённых двоеточиями, с указанием модулей спецификации LSB, которым соответствует установка. Если установка не соответствует требованиям, вышеуказанное поле должно отсутствовать.

!> Замените значения переменных в этом файле на нужные вам. Например, значение `MyLinux` переменной `DISTRIB_ID` заменине на ваше уникальное название дистрибутива.

12.1.3 Перезагрузка

Теперь, когда всё программное обеспечение установлено, можно перезагрузить компьютер. Однако вам следует знать о нескольких вещах. Система, которую вы создали, весьма минимальна. Скорее всего потребуется множество других программ, например, графическое окружение, и в [следующей части](#) будут предоставлены инструкции по их сборке.

На этом этапе также уместно проверить следующие файлы конфигурации:

```
1  /etc/bashrc
2  /etc/dircolors
3  /etc/fstab
4  /etc/hosts
5  /etc/inputrc
6  /etc/profile
7  /etc/resolv.conf
8  /etc/vimrc
9  /root/.bash_profile
10 /root/.bashrc
```

?> Файлы `/root/.bash_profile` и `/root/.bashrc` можете скопировать в каталог `/etc/skel` - из него файлы копируются в домашние директории пользователей, которых вы создадите в будущем.

Выполните выход из среды chroot:

```
1  logout
```

Отключим виртуальные файловые системы:

```
1  umount $LIN/dev{/pts,}
2  umount $LIN/{sys,proc,run}
```

Размонтируем иерархию каталогов созданной Linux системы:

```
1  umount -Rv $LIN
```

Выполните перезагрузку

```
1  shutdown -r now
```

После перезагрузки система будет готова к использованию.

13. Вспомогательные материалы

13.1 Вспомогательные материалы

Это необязательная часть, служащая дополнительным источником знаний. В некоторых случаях здесь могут быть решения возникающих проблем и важные заметки по процессу работы в целом.

Этот раздел создан, чтобы свести к минимуму Ваши поиски необходимой информации и выполнить аккумулирование самых важных вопросов в одном месте.

13.2 Строение GNU/Linux

13.2.1 Строение GNU/Linux. Часть 1.

Дистрибутив по Linux for yourself собрать нельзя, не умея читать документацию, искать нужную информацию в интернете и не зная строение Linux. Именно о строении и пойдёт речь в этой статье. Первая часть даст базовые сведения, а в следующих частях структура типичного Linux-дистрибутива будет рассмотрена подробнее.

Книга *Linux for yourself* старается придерживаться стандартов FHS и LSB (о них далее).

Термины

- FHS (Filesystem Hierarchy Standard) - документ, который определяет схему директорий в UNIX-системах. FHS разработан, чтобы предоставить общую схему для упрощения независимой от дистрибутива разработки программного обеспечения (далее - ПО). Другими словами, это стандарт, унифицирующий местонахождение файлов и каталогов с общим назначением в файловой системе UNIX. Таблицу директорий и другую информацию вы сможете лицезреть [здесь](#).
- LSB (Linux Standard Base) - совместный проект дистрибутивов Linux при организации Linux Foundation, целью которого является стандартизация их внутренней инфраструктуры. Цель LSB - разработать и продвигать набор стандартов, который увеличит совместимость различных дистрибутивов Linux и даст возможность запускать приложения на любой совместимой системе; помимо этого, LSB помогает сконцентрировать усилия в привлечении разработчиков к написанию и портированию ПО. LSB сертифицирует стандартные библиотеки, пару утилит, структуру иерархии файловой системы (далее - ФС), уровни запуска и et cetera.
- конфиг - сленговое. Означает "конфигурационный файл".
- юзер - сленговое. Означает "пользователь".
- ФС - сокращение от "файловая система".

Filesystem Hierarchy Standard

В данном разделе речь пойдёт про иерархию ФС Linux. Может быть применимо и к другим UNIX-Like системам и дистрибутивам, главное, чтобы они соответствовали стандарту FHS. Всякие NixOS и Gobo Linux не рассматриваем.

Независимые классификации в FHS

Спецификация FHS основывается на идее существования двух независимых классификаций файлов: разделяемых и неразделяемых, а также изменяемые и статичные. Разделяемые данные могут распространяться на несколько хостов; неразделяемые специфичны для конкретного хоста (например, конфиги). Соответственно, изменяемые файлы изменяются, а статичные - нет (за исключением установки и обслуживания системы).

Резюме. 4 возможные комбинации + нужные директории.

	Разделяемые	Неразделяемые
статичные	/usr /opt	/etc /boot
изменяемые	/var/mail	/var/run

Само строение ОС

ПРОЦЕСС ЗАГРУЗКИ

В данной статье загрузка ПК, BIOS, UEFI и прочее будет пропущено, ибо это не тема этой статьи.

Начнём с загрузчика. Их очень много - GRUB2, rEFInd, systemd-boot, syslinux и другие. Но самым популярным является, конечно, GRUB2. Задача загрузчика - инициализировать ядро Linux (в нашем случае). В помощь ядру загрузчик обычно использует начальный образ загрузки - `initrd` или `initramfs`, представляющий собой архив с образом файловой системы, разворачивающейся в ОЗУ в начале загрузки. В `initrd` / `initramfs` находятся нужные драйверы, скрипты и прочее, что необходимо для инициализации оборудования и прочих целей.

После начального образа загружается ядро. Находится в `/boot`, в своём названии имеет `vmlinuz`:

- `vm` - поддержка виртуальной памяти;
- `linu` - когда-то называлось `linux`, но позже сократилось до текущего состояния, потому что необходимо указать факт сжатия ядра (следующий пункт);
- `z` - указатель того, что файл с ядром сжат (формат сжатия обычно `zlib`. Не всегда используется именно это сжатие, иногда можно встретить `LZMA` или `BZIP2`, поэтому некоторые ядра называют просто `zImage`).

Ядро Linux является монолитным.

Плюсы монолитного ядра

- Более прямой доступ к аппаратным средствам
- Проще обмен данными между процессами
- Процессы реагируют быстрее

Минусы монолитного ядра

- Большой размер
- Занимает много оперативной памяти
- Менее безопасно

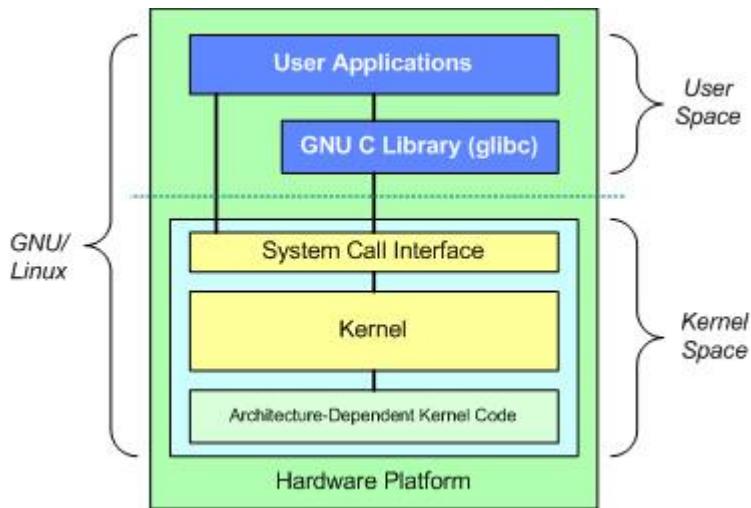
Монолитное ядро больше и несколько сложнее других видов ядер. Ну и вспомните спор Таненбаума и Торвальдса. Но у Linux есть достоинство - это *модули ядра*. Подключать нужные модули можно буквально на лету. вы можете запускать процессы сервера, подключать виртуализацию, а также полностью заменить ядро без перезагрузки. Например, команда

```
1 modprobe qemu-nbd
```

Подключит соответствующий модуль `qemu-nbd`.

GNU/Linux состоит из четырёх основных частей:

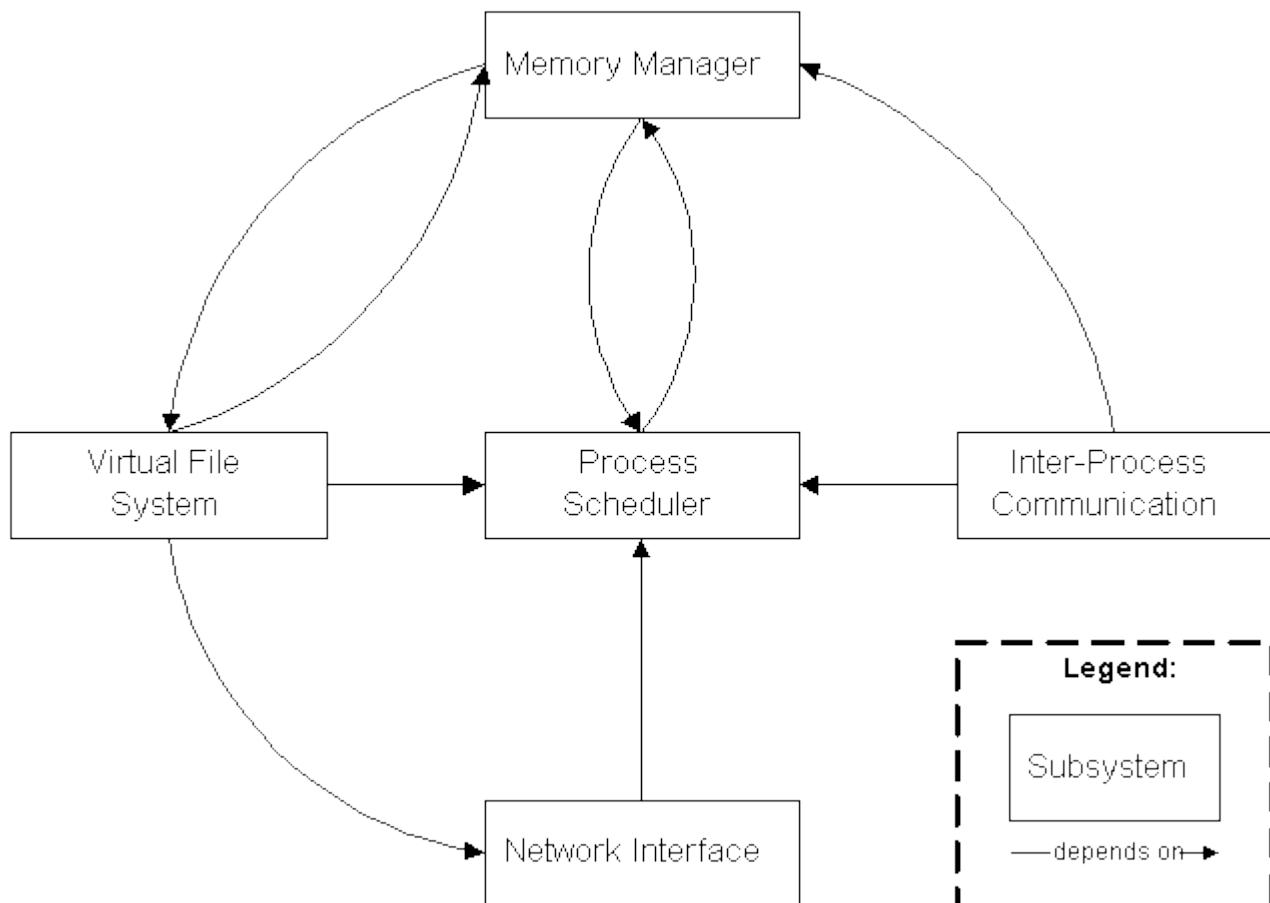
- Ядро Linux — основная интересующая нас часть; ядро создаёт абстрактный слой и является «посредником» между первыми двумя частями и hardware-частью компьютера;
- Hardware controllers (контроллеры оборудования) — подсистема, охватывающая все возможные физические устройства, такие как CPU, устройства памяти, жёсткие диски, сетевые карты — все они являются частью этой подсистемы
- O/S services (службы операционной системы) — службы, которые обычно считаются частью операционной системы. Например, оконные менеджеры. Ну и программный интерфейс ядра (компиляторы и прочее);
- User applications (пользовательские приложения) — набор пользовательских приложений, который в разных дистрибутивах Linux может быть разным. Например, в Linux For Yourself предложен самый небольшой набор ПО, которого хватит для корректной работы операционной системы.

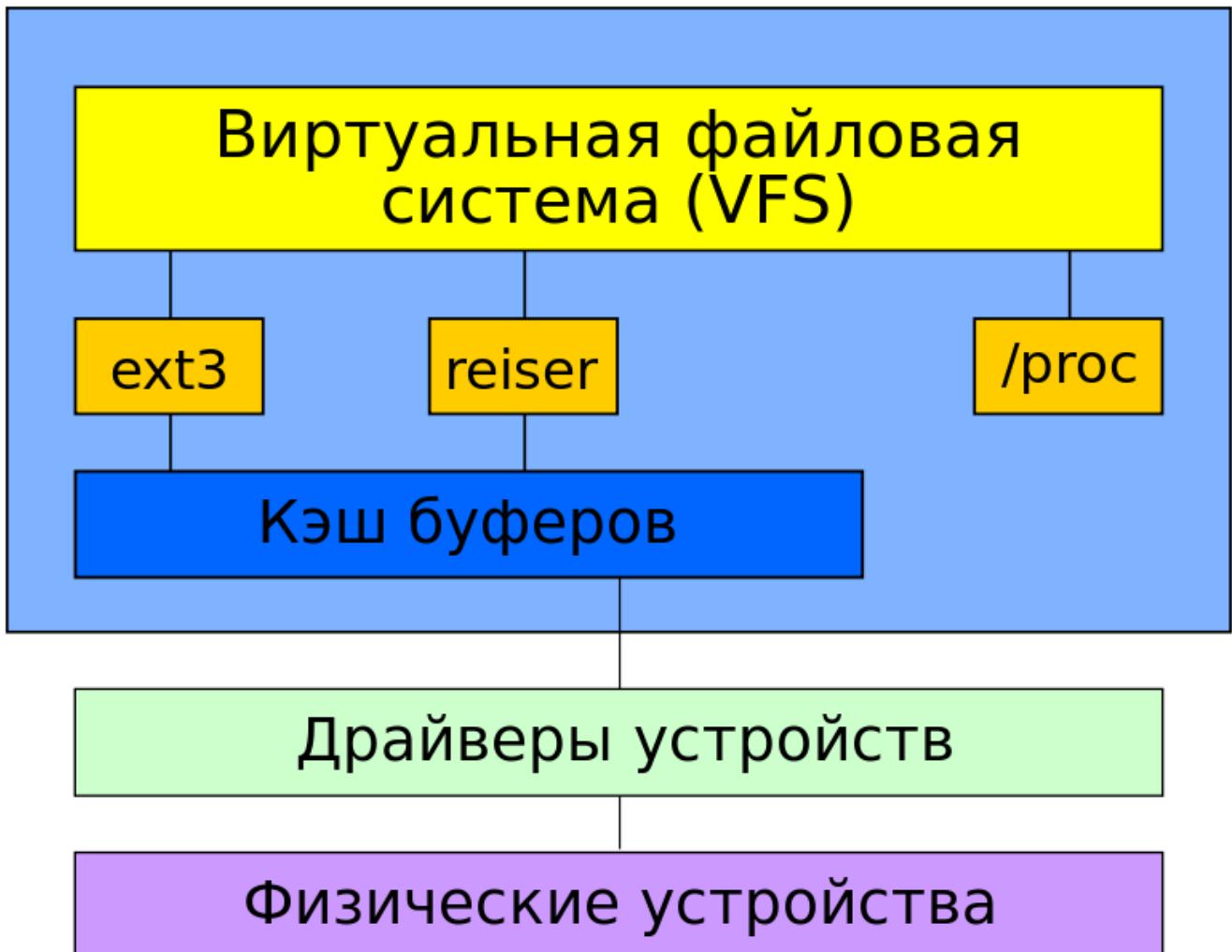


Каждая подсистема может взаимодействовать только с двумя соседними, расположеннымми непосредственно «выше» и «ниже» её уровня. Кроме того, зависимости между этими подсистемами направлены сверху вниз: слои, расположенные выше — зависят от частей ниже, но части, расположенные ниже — не зависят от частей выше их.

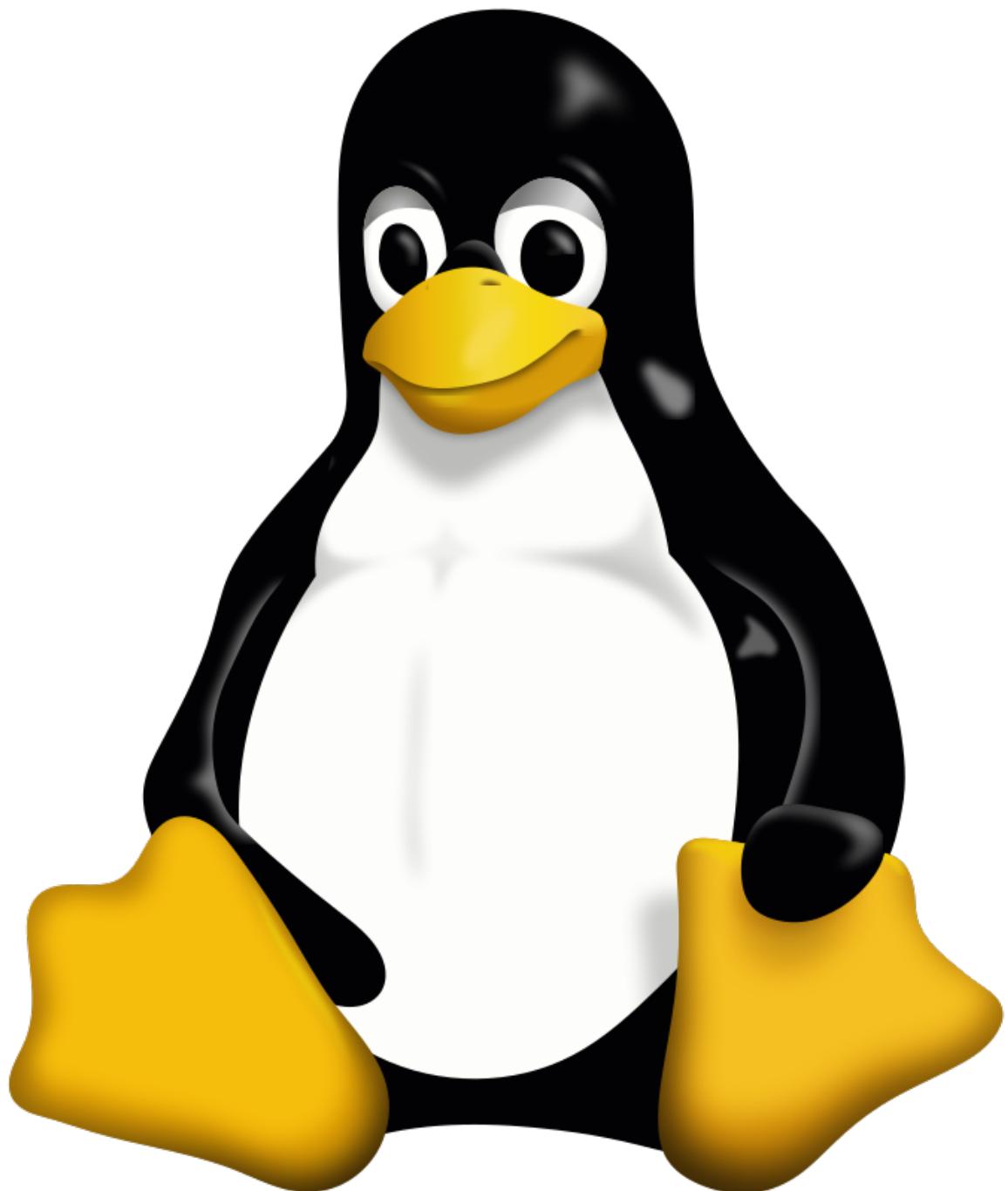
Ядро Linux состоит из пяти основных подсистем:

- Process Scheduler (SCHED) — планировщик процессов, отвечает за контроль над доступом процессов к CPU. Планировщик обеспечивает такое поведения ядра, при котором все процессы имеют справедливый доступ к центральному процессору;
- Memory Manager (MM) — менеджер памяти, обеспечивает различным процессам безопасный доступ к основной памяти системы. Кроме того, MM обеспечивает работу виртуальной памяти, которая позволяет процессам использовать больше памяти, чем реально доступно в системе. Выделенная, но неиспользуемая память вытесняется на файловую систему, и при необходимости — возвращается из неё обратно в память (swapping);
- Virtual File System (VFS) — виртуальная файловая система, создаёт абстрактный слой, скрывая детали оборудования, предоставляя общий файловый интерфейс для всех устройств. Кроме того, VFS поддерживает несколько форматов файловых систем, которые совместимы с другими операционными системами;
- Network Interface (NET) — сетевые интерфейсы, обеспечивает работу с различными сетевыми стандартами и сетевым оборудованием;
- Inter-Process Communication (IPC) — межпроцессная подсистема, поддерживающая несколько механизмов для process-to-process связей в единой Linux-системе.





Символом ядра Linux является Tux, отличающийся от «обычных» пингвинов жёлтым цветом клюва и лап. Однако, в качестве символа Linux 2.6.29 был принят Tuz (тасманский дьявол), изображение которого ранее служило талисманом конференции linux.conf.au 2009. В следующих версиях ядра используется предыдущий пингвин Tux. Его можно наблюдать и поныне.



Tux



Tuz

После ядра стартует `init` - подсистема инициализации в UNIX-ах, которая запускает все остальные процессы. Имеет `PID = 1`. Раньше, де-факто стандартным инитом в GNU/Linux был SysVinit. Однако, в самом начале 10-ых были попытки заменить SysVinit на другую систему инициализации. Например, в Ubuntu первое время использовался `upstart` от Canonical, а в Gentoo Linux используется `OpenRC`. Также был создан `systemd`, на который в данный момент перешло большинство дистрибутивов Linux.

Достоинства `systemd`

- Агрессивная параллелизация и прочее, что позволяет существенно ускорить загрузку ОС
- Запуск сервисов по расписанию (вместо `cron`)
- Смена корня (вместо `chroot`)
- Простой и лаконичный синтаксис служб

Недостатки `systemd`

- Не *Unix Way*. `systemd` - монолитная и сложная система, заменяющая собой не только ини, но и планировщик, менеджер сети, утилиту по смене корня системы, просмотрщик логов и пр, что не особо нужно многим пользователям
- `systemd` требуется несколько больше ресурсов, чем его менее прожорливым товарищам, из-за чего на старом железе лучше использовать дистрибутив с другой системой инициализации, например, классическим `SysVinit`

И теперь про `SysVinit`.

Достоинства `SysVinit`

- Устоявшаяся и хорошо понятная система
- Простая настройка
- Стабильная и надёжная работа

Недостатки `SysVinit`

- Неудобная (а для некоторых ещё и сложная) работа с сервисами
 - Последовательная обработка задач загрузки, что может в некоторых случаях замедлить скорость старта ОС
-

После иниата загружается `командная оболочка`, коих так же довольно много:

- `bourne shell (sh)` - "тот самый";
- `bourne again shell (bash)` - ставшая классической в Linux оболочка, которая используется во многих дистрибутивах по умолчанию. В *Linux for yourself* `/bin/sh` является символьской ссылкой на `/bin/bash`;
- `BusyBox` - представляет собой целое пользовательское окружение, в том числе, командную оболочку (вызов: `busybox sh`). Достоинство: малый размер и небольшие требования к аппаратуре;
- `Z shell (zsh)` - мощная современная оболочка. Она почти полностью совместима с `bash`, но имеет преимущества, такие как улучшенное завершение на клавишу `Tab` (англ. `tab completion`), автоматическое исправление опечаток и огромное количество разработанных сообществом плагинов. К слову, именно она по умолчанию используется в *macOS*.
- `friendly interactive shell (fish)` - удобная оболочка, которая "из коробки" имеет подсветку синтаксиса, автодополнения, генерируемые из мануалов, и т. п. Из минусов нужно выделить несовместимость со спецификацией `POSIX sh`;
- `et cetera`.

Командная оболочка прописана в стандарте `POSIX` и необходима для работы системы. В *Linux for yourself* список оболочек содержится в файле `/etc/shells`.

Далее идёт графика. О ней можно говорить очень много, ибо тема довольно объёмная, но я опишу всё вкратце.

Состоит из следующих компонентов:

- `Display Server` - дисплейный сервер;
- `Display Manager (DM)` - дисплейный менеджер;
- `Desktop Manager (DE)` - рабочее окружение.

Рабочее окружение (DE) можно разделить на следующее:

- WM (Window Manager) - программа, которая управляет отображением окон в системе GUI (Graphic User Interface) - графическом интерфейсе пользователя;
- Средство запуска ПО;
- Панель, на которой располагаются апплеты/виджеты:
 - средство запуска приложений;
 - апплет переключения между открытыми окнами;
 - апплет системного трея;
 - другие апплеты/виджеты по усмотрению пользователя.
- Файловый менеджер, задача которого - не только работа с файлами, но и управление иконками на рабочем столе;
- Менеджер обоев рабочего стола;
- Приложения по умолчанию (эмулятор терминала, файловый менеджер (см. по списку выше), настройки системы и рабочего окружения, et cetera);
- et cetera.

Example

Хочу заострить ваше внимание на том, что оконный менеджер, панель и некоторые другие компоненты можно использовать и отдельно от рабочего окружения. Например, оконные менеджеры Fluxbox, Openbox, i3-wm, Awesome, Sway и др. используются отдельно от DE. Также существуют отдельные файловые менеджеры, которые так же поставляются отдельно от рабочего окружения.

Warning

Не путайте понятия "рабочий стол" и "рабочее окружение". Хотя эти два термина похожи, но значения немного разные.

Теперь описание компонентов графики.

DISPLAY SERVER

Дисплейный сервер - программа, отвечающая за координацию ввода и вывода своих клиентов с ОС, а также между оборудованием и ОС. Именно благодаря Display Server вы используете Linux в графическом режиме, а не в TTY. Когда говорят "дисплейный сервер", нередко имеют ввиду Xorg, Wayland, Mir и прочие.

DISPLAY MANAGER

Первая программа, которая запускается при старте графики. Её основные задачи:

- запросить аутентификационные данные нужного пользователя (имя, пароль или отпечаток пальца);
- выбрать, какую среду рабочего стола запустить.

Примеры DM:

- LightDM;
- XDM;
- GDM (из состава GNOME);
- SDDM (из состава KDE);
- TDM (из состава Trinity);
- WDM (из состава Window Maker).

Как и упоминалось ранее, дисплейные менеджеры можно использовать и отдельно от DE.

WINDOW MANAGER

Я не случайно начал именно с WM. Во-первых, про DE было сказано выше, а во-вторых, оконный менеджер можно использовать отдельно от всего громоздкого рабочего окружения. Что такое WM было так же сказано выше. Можно выделить три вида оконных менеджеров:

- Стековые (плавающие , англ. *stacking* , *floating*) следуют классической метафоре, которая на данный момент самая удобная и популярная. Классическое расположение окон, которые могут накладываться и перекрывать друг друга;
- Фреймовые (англ. *tiling*) WM располагают окна в виде фреймов (плиток), эти фреймы не способны перекрывать друг друга, подобное поведение встречается в графическом интерфейсе *Windows 1.x*. Наиболее удобно использовать такие оконные менеджеры посредством клавиатуры, хотя поддержка мыши во многих из них также присутствует;
- Динамические (например, i3wm) - динамически переключаются между двумя режимами, описанными выше (стековый и фреймовый режимы). Поддержка мыши есть в большинстве из них.

13.2.2 Строение Linux - часть 2. Принцип "всё есть файл", виды файлов.

В первой части статьи речь была об общем строении системы. Так сказать, галопом по европам. А в следующих частях строение Linux будет описано подробнее. Каждая статья описывает только одну тему.

Для начала стоит написать о концепции "всё есть файл". Концепция была перенята разработчиками Linux из Unix. Это было сделано для предоставления простого доступа ко всем возможностям ОС, не разрабатывая отдельных костылей. Т.е., преимущество такого принципа в том, что не надо реализовывать отдельный API для каждого устройства, в результате чего с ним (с файлом) могут работать все стандартные программы и API-интерфейсы. В Linux есть корневая ФС (корневая файловая система), куда монтируются раздел жёсткого диска, где установлена система, другие разделы, флешки, диски, псевдо-ФС и пр. Посмотрите на файл `/etc/os-release`:

```
1 file /etc/os-release
```

```
liveuser@localhost-live:~
```

```
liveuser@localhost-live:/home/liv... x liveuser@localhost-live:~ x
```

```
[liveuser@localhost-live ~]$ file /etc/os-release
/etc/os-release: symbolic link to ../../usr/lib/os-release
[liveuser@localhost-live ~]$ file /usr/lib/os-release
/usr/lib/os-release: ASCII text
[liveuser@localhost-live ~]$
```

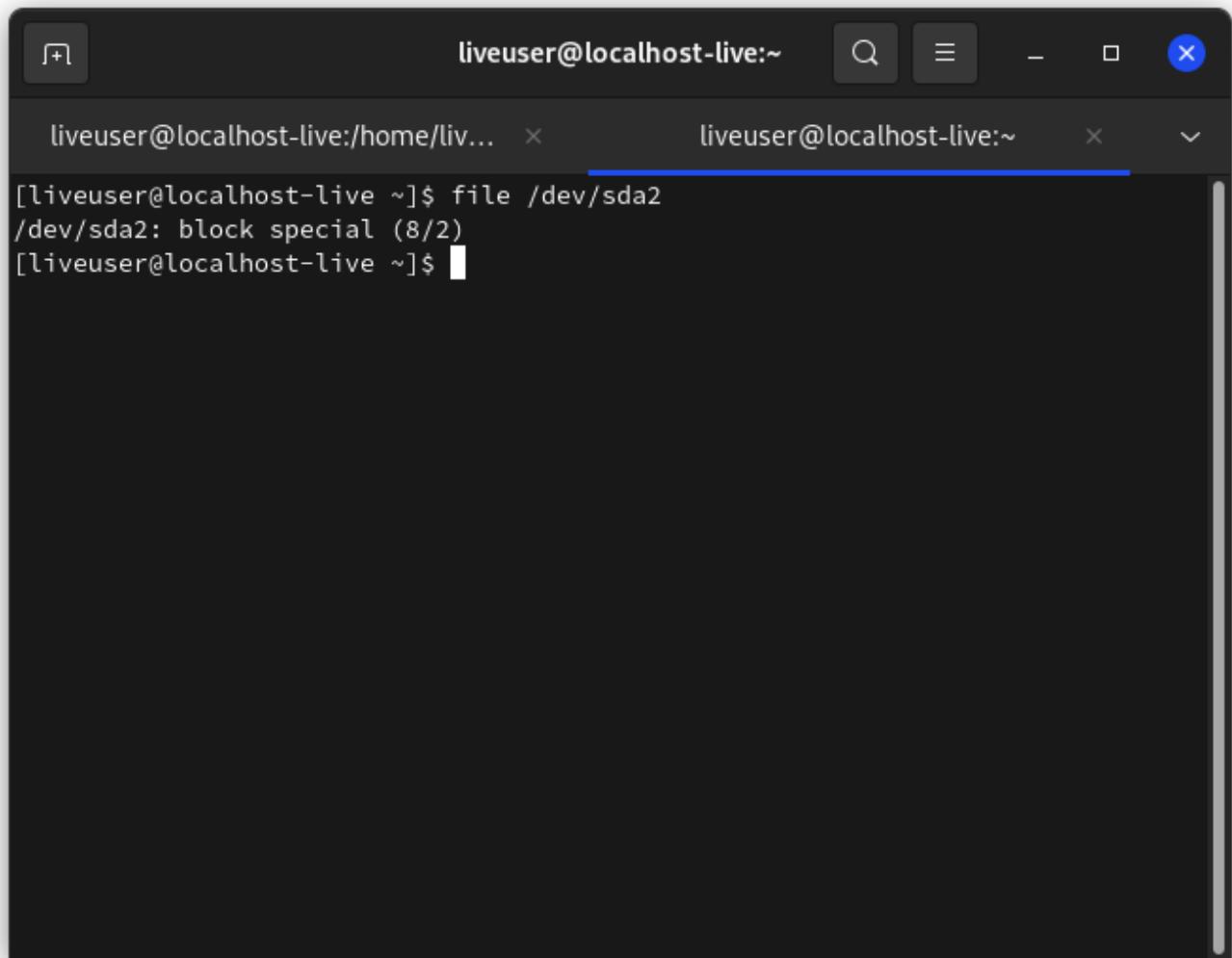
Самое яркое представление этого принципа - устройства. Просмотрите содержимое каталога `/dev`:

```
1 ls /dev |less
```

В этот каталог подключаются все устройства: флеш-карты, мыши, клавиатуры, микрофоны, жёсткие диски и пр.

А теперь просмотрите информацию о каком-нибудь файле в `/dev`:

```
1 file /dev/sda2
```



liveuser@localhost-live:~

liveuser@localhost-live:/home/liv... × liveuser@localhost-live:~ ×

```
[liveuser@localhost-live ~]$ file /dev/sda2
/dev/sda2: block special (8/2)
[liveuser@localhost-live ~]$
```

В этом файле находятся двоичные данные, поэтому открыть его в каком-то текстовом редакторе бесполезно.

Однако, самое главное достоинство Linux в том, что и в обычном файле можно создать файловую систему вместо содержимого файла. Например, тот же файл подкачки `/swapfile`. Это файл, но с ФС `swap`.

Все конфиги, находящиеся в директориях `/etc`, `~/.local`, `~/.config` - тоже файлы.

Типы файлов

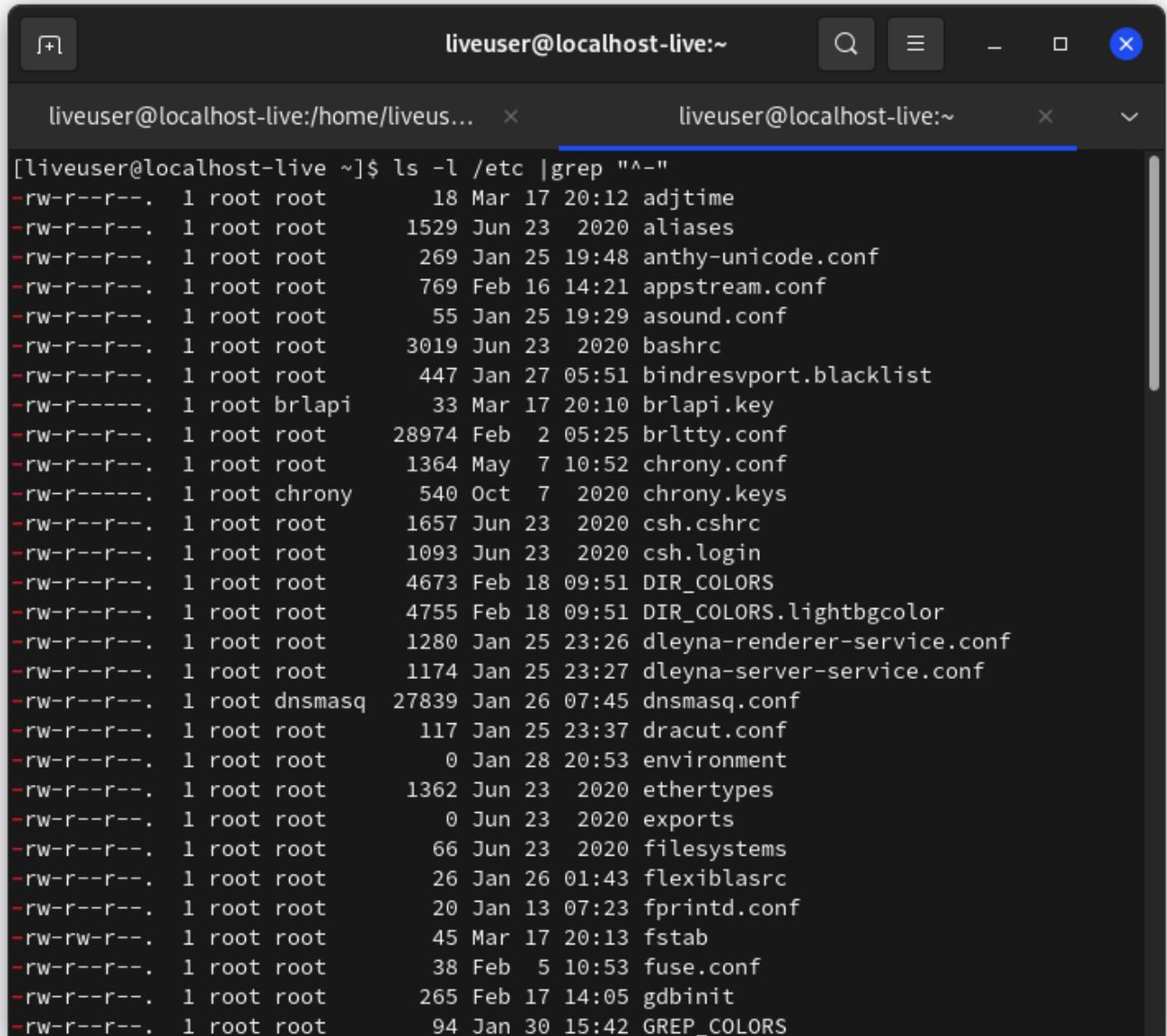
Есть 3 типа файлов:

- **Обыкновенные**, которые используются для хранения информации;
- **Специальные** (для туннелей и устройств);
- **Директории** (их ещё называют *папками* или *каталогами*).

С обычными файлами пользователь работает чаще всего. Это документы, текстовые файлы, музыка, видео и пр.

Для того чтобы просмотреть эти файлы, выполните:

```
1  ls -l /etc |grep "^-"
```



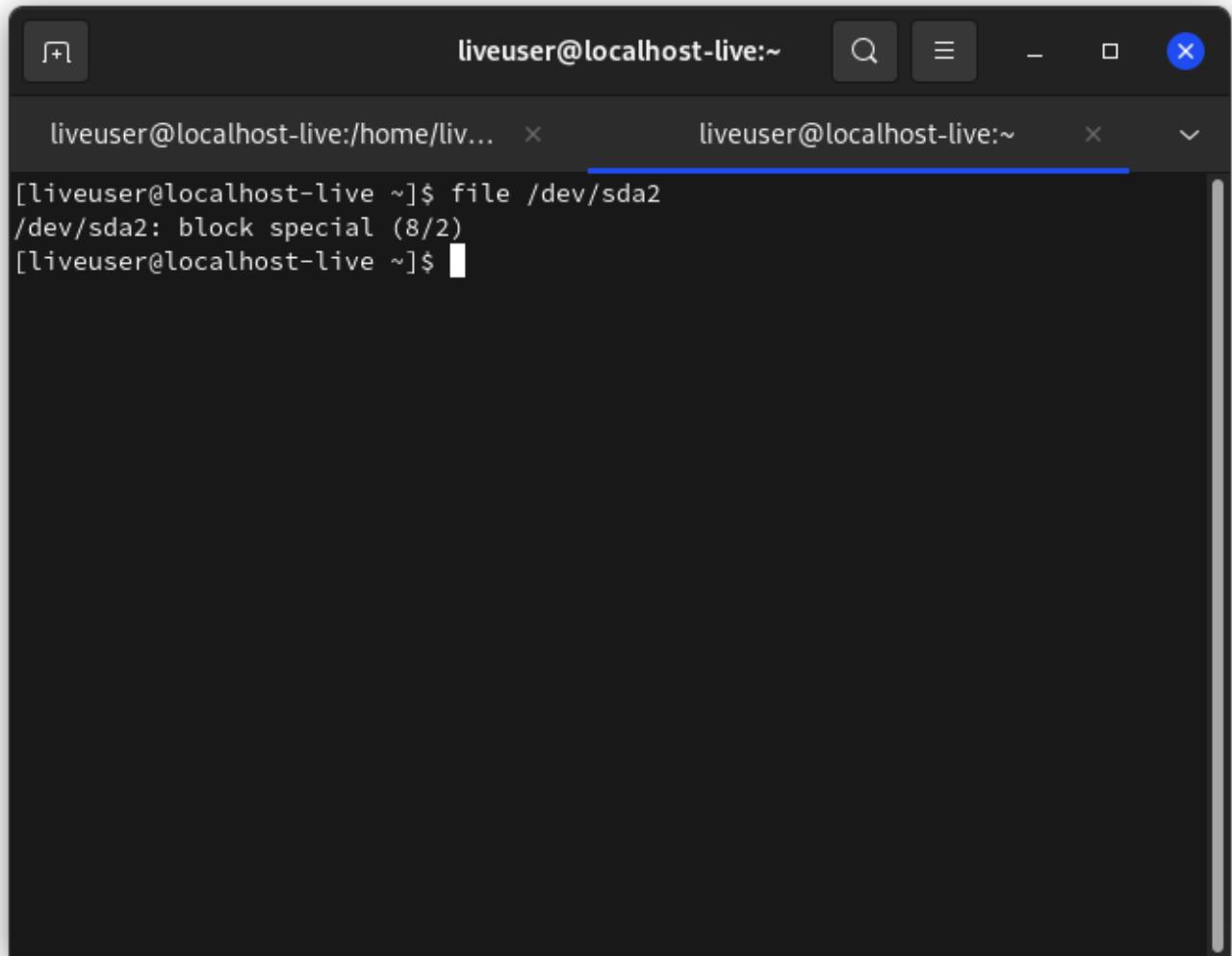
```
[liveuser@localhost-live ~]$ ls -l /etc | grep "^-"
-rw-r--r--. 1 root root 18 Mar 17 20:12 adjtime
-rw-r--r--. 1 root root 1529 Jun 23 2020 aliases
-rw-r--r--. 1 root root 269 Jan 25 19:48 anthy-unicode.conf
-rw-r--r--. 1 root root 769 Feb 16 14:21 appstream.conf
-rw-r--r--. 1 root root 55 Jan 25 19:29 asound.conf
-rw-r--r--. 1 root root 3019 Jun 23 2020 bashrc
-rw-r--r--. 1 root root 447 Jan 27 05:51 bindresvport.blacklist
-rw-r-----. 1 root brlapi 33 Mar 17 20:10 brlapi.key
-rw-r--r--. 1 root root 28974 Feb 2 05:25 brltty.conf
-rw-r--r--. 1 root root 1364 May 7 10:52 chrony.conf
-rw-r-----. 1 root chrony 540 Oct 7 2020 chrony.keys
-rw-r--r--. 1 root root 1657 Jun 23 2020 csh.cshrc
-rw-r--r--. 1 root root 1093 Jun 23 2020 csh.login
-rw-r--r--. 1 root root 4673 Feb 18 09:51 DIR_COLORS
-rw-r--r--. 1 root root 4755 Feb 18 09:51 DIR_COLORS.lightbgcolor
-rw-r--r--. 1 root root 1280 Jan 25 23:26 dleyna-renderer-service.conf
-rw-r--r--. 1 root root 1174 Jan 25 23:27 dleyna-server-service.conf
-rw-r--r--. 1 root dnsmasq 27839 Jan 26 07:45 dnsmasq.conf
-rw-r--r--. 1 root root 117 Jan 25 23:37 dracut.conf
-rw-r--r--. 1 root root 0 Jan 28 20:53 environment
-rw-r--r--. 1 root root 1362 Jun 23 2020 ethertypes
-rw-r--r--. 1 root root 0 Jun 23 2020 exports
-rw-r--r--. 1 root root 66 Jun 23 2020 filesystems
-rw-r--r--. 1 root root 26 Jan 26 01:43 flexiblasrc
-rw-r--r--. 1 root root 20 Jan 13 07:23 fprintd.conf
-rw-rw-r--. 1 root root 45 Mar 17 20:13 fstab
-rw-r--r--. 1 root root 38 Feb 5 10:53 fuse.conf
-rw-r--r--. 1 root root 265 Feb 17 14:05 gdbinit
-rw-r--r--. 1 root root 94 Jan 30 15:42 GREP_COLORS
```

ЗНАЧЕНИЕ КОМАНДЫ

- `ls` - просматривает каталог, а ключ `-l` добавляет отображение прав на файл. `/etc` замените на нужную директорию.
- `grep "^-"` - так как "обычные" файлы обозначаются чертой (в первой колонке вывода `ls`, где отображаются права на файл), то эта команда выведет только эти файлы по маске `^-`.

По поводу специальных файлов. Они обеспечивают обмен информации с ядром, работу с устройствами и пр. Собственно, делятся ещё на несколько видов:

- **Символьные файлы** - любые специальные системные, например `/dev/null`, или периферийные устройства (последовательные/параллельные порты). Такие файлы идентифицированы символом `c`.
- **Блочные** - периферийные устройства, но в отличие от предыдущего типа, содержание блочных файлов буферизируется. Эти файлы идентифицированы символом `b`.



liveuser@localhost-live:~

```
[liveuser@localhost-live ~]$ file /dev/sda2
/dev/sda2: block special (8/2)
[liveuser@localhost-live ~]$
```

- **Символические ссылки** (символические ссылки) - указывают на другие файлы по их имени, указывают и на другие файлы, в т.ч. каталоги. Обозначены символом `l`. В выводе команды `ls -l /путь/до/директории |grep "^\!"` можно увидеть, на какой файл ссылается символическая ссылка - в последней колонке название имеет следующий вид: НАЗВАНИЕ ФАЙЛА \rightarrow НА ЧТО ССЫЛАЕТСЯ
- **Туннели** (каналы/именованные каналы) - очень похожи на туннели из `Shell`, но разница в том, что именованные каналы имеют название. Они очень редки. Обозначены символом `r`.

Информация о файлах

lsof

Список всех открытых файлов можно просмотреть с помощью команды `lsof` - ListOpenFiles. Эта информация поможет узнать о многом происходящем в системе, об устройстве и работе Linux, а также решить проблемы, например, когда вы не можете размонтировать диск из-за того, что устройство используется, но вы не можете найти, какой именно программой.

COMMAND		PID	TID	TASKCMD	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
systemd		1			root	cwd	DIR	253,0	4096	2	/
systemd		1			root	rtd	DIR	253,0	4096	2	/
systemd		1			root	txt	REG	253,0	1801440	9435	/usr/lib/systemd/systemd
systemd		1			root	DEL	REG	253,0		263574	/etc/selinux/targeted/contexts/files/file_contexts.bin
systemd		1			root	mem	REG	253,0	146320	16290	/usr/lib64/libnl-3.so.200.26.0
systemd		1			root	mem	REG	253,0	544704	16298	/usr/lib64/libnl-route-3.so.200.26.0
systemd		1			root	mem	REG	253,0	134936	16527	/usr/lib64/libibverbs.so.1.12.34.0
systemd		1			root	mem	REG	253,0	40192	16222	/usr/lib64/libffi.so.6.0.2
systemd		1			root	mem	REG	253,0	318624	16533	/usr/lib64/libcap.so.1.10.0
systemd		1			root	mem	REG	253,0	153216	16215	/usr/lib64/libpg-error.so.0.32.0
systemd		1			root	mem	REG	253,0	28656	16810	/usr/lib64/libattr.so.1.1.2501
systemd		1			root	mem	REG	253,0	103184	16053	/usr/lib64/libz.so.1.2.11
systemd		1			root	mem	REG	253,0	32696	16245	/usr/lib64/libcap-ng.so.0.0.0
systemd		1			root	mem	REG	253,0	1911064	15752	/usr/lib64/libm-2.33.so
systemd		1			root	mem	REG	253,0	36768	17490	/usr/lib64/libcconf.so.0.3.8
systemd		1			root	mem	REG	253,0	618856	16437	/usr/lib64/libpcre2-8.so.0.10.1
systemd		1			root	mem	REG	253,0	37288	15750	/usr/lib64/libdl-2.33.so
systemd		1			root	mem	REG	253,0	178496	16068	/usr/lib64/liblzma.so.5.2.5
systemd		1			root	mem	REG	253,0	807552	16092	/usr/lib64/libzstd.so.1.4.9
systemd		1			root	mem	REG	253,0	1289072	16232	/usr/lib64/libp11-kit.so.0.3.0
systemd		1			root	mem	REG	253,0	3096376	17703	/usr/lib64/libcrypto.so.1.1.1k
systemd		1			root	mem	REG	253,0	144152	16395	/usr/lib64/liblz4.so.1.9.3
systemd		1			root	mem	REG	253,0	37512	17533	/usr/lib64/libip4tc.so.2.0.0
systemd		1			root	mem	REG	253,0	1255824	16218	/usr/lib64/libgcrypt.so.20.3.2
systemd		1			root	mem	REG	253,0	201896	16066	/usr/lib64/libcrypt.so.2.0.0
systemd		1			root	mem	REG	253,0	37672	16168	/usr/lib64/libcap.so.2.48
systemd		1			root	mem	REG	253,0	217320	17697	/usr/lib64/libblkid.so.1.1.0
systemd		1			root	mem	REG	253,0	41216	16812	/usr/lib64/libacl.so.1.1.2301
systemd		1			root	mem	REG	253,0	3228320	15748	/usr/lib64/libc-2.33.so
systemd		1			root	mem	REG	253,0	301592	15762	/usr/lib64/libpthread-2.33.so
systemd		1			root	mem	REG	253,0	108336	15611	/usr/lib64/libgcc_s-11-20210324.so.1
systemd		1			root	mem	REG	253,0	107368	17715	/usr/lib64/libkmod.so.2.3.6
systemd		1			root	mem	REG	253,0	131552	16249	/usr/lib64/libaudit.so.1.0.0
systemd		1			root	mem	REG	253,0	69976	17727	/usr/lib64/libpam.so.0.85.1
systemd		1			root	mem	REG	253,0	274464	17695	/usr/lib64/libmount.so.1.1.0
systemd		1			root	mem	REG	253,0	171376	16548	/usr/lib64/libselinux.so.1

Выход lsof состоит из нескольких колонок с информацией:

- **COMMAND** - имя команды, которая открыла или использует файл;
- **PID** - PID процесса;
- **TID** - идентификационный номер задачи (потока). Пустой столбец означает, что это не задача, а процесс;
- **TASKCMD** - имя команды задачи. Обычно имеет то же самое название, что и процесс, названный в столбце **COMMAND**, но некоторые реализации задач (например, Linux) позволяют задаче изменить имя своей команды;
- **USER** - имя пользователя, которому соответствует процесс, либо тот пользователь, которому принадлежит директория `/proc`, откуда lsof берёт информацию о процессе;
- **FD** - показывает файловый дескриптор файла;
- **TYPE** - тип узла, связанного с файлом;
- **DEVICE** - содержит номера устройств, разделённые запятыми, для специальных символьных, специальных блочных, обычных файлов, каталогов или NFS. Também может отображаться базовый адрес или имя устройства с сокетом Linux AX.25;
- **SIZE/OFF** - размер файла/смещение файла в байтах;
- **NODE** - показывает номер узла локального файла или номер узла NFS-файла на хосте сервера или тип интернет-протокола. Может отображаться STR для потока, IRQ или номер инода устройства с сокетом Linux AX.25;
- **NAME** - имя точки монтирования и файловой системы, в которой находится файл;

ОПЦИИ LSOF

- **-u** - список файлов, открытых конкретным пользователем. Например, список открытых файлов пользователем `liveuser`:

```
1 lsof -u liveuser
```

liveuser@localhost-live:~ — lsof -u liveuser							
COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE NAME
systemd	1380	liveuser	cwd	DIR	253,0	4096	2 /
systemd	1380	liveuser	rtd	DIR	253,0	4096	2 /
systemd	1380	liveuser	txt	REG	253,0	1801440	9435 /usr/lib/systemd/systemd
systemd	1380	liveuser	DEL	REG	253,0	263574	/etc/selinux/targeted-contexts/files/file_contexts.bin
systemd	1380	liveuser	mem	REG	253,0	146320	16290 /usr/lib64/libnl-3.so.200.26.0
systemd	1380	liveuser	mem	REG	253,0	544704	16298 /usr/lib64/libnl-route-3.so.200.26.0
systemd	1380	liveuser	mem	REG	253,0	134936	16527 /usr/lib64/libibverbs.so.1.12.34.0
systemd	1380	liveuser	mem	REG	253,0	40192	16222 /usr/lib64/libffi.so.6.0.2
systemd	1380	liveuser	mem	REG	253,0	318624	16533 /usr/lib64/libpcap.so.1.10.0
systemd	1380	liveuser	mem	REG	253,0	153216	16215 /usr/lib64/libgpg-error.so.0.32.0
systemd	1380	liveuser	mem	REG	253,0	28656	16810 /usr/lib64/libattr.so.1.1.2501
systemd	1380	liveuser	mem	REG	253,0	103184	16053 /usr/lib64/libz.so.1.2.11
systemd	1380	liveuser	mem	REG	253,0	32696	16245 /usr/lib64/libcap-ng.so.0.0.0
systemd	1380	liveuser	mem	REG	253,0	1911064	15752 /usr/lib64/libm-2.33.so
systemd	1380	liveuser	mem	REG	253,0	36768	17490 /usr/lib64/libeconf.so.0.3.8
systemd	1380	liveuser	mem	REG	253,0	618856	16437 /usr/lib64/libpcre2-8.so.0.10.1
systemd	1380	liveuser	mem	REG	253,0	37288	15750 /usr/lib64/libdl-2.33.so
systemd	1380	liveuser	mem	REG	253,0	178496	16068 /usr/lib64/liblzma.so.5.2.5
systemd	1380	liveuser	mem	REG	253,0	807552	16092 /usr/lib64/libzstd.so.1.4.9
systemd	1380	liveuser	mem	REG	253,0	1289072	16232 /usr/lib64/libp11-kit.so.0.3.0
systemd	1380	liveuser	mem	REG	253,0	3096376	17703 /usr/lib64/libcrypto.so.1.1k
systemd	1380	liveuser	mem	REG	253,0	144152	16395 /usr/lib64/liblz4.so.1.9.3

- **-U** - вывести все файлы сокетов домена Unix

liveuser@localhost-live:~ — lsof -U							
COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE NAME
systemd	1380	liveuser	1u	unix 0x00000000379fe46e	0t0	28901	type=STREAM (CONNECTED)
systemd	1380	liveuser	2u	unix 0x00000000379fe46e	0t0	28901	type=STREAM (CONNECTED)
systemd	1380	liveuser	3u	unix 0x00000000571947e2	0t0	28918	type=DGRAM (UNCONNECTED)
systemd	1380	liveuser	12u	unix 0x00000000737a679c	0t0	28023	/run/user/1000/bus type=STREAM (LISTEN)
systemd	1380	liveuser	16u	unix 0x0000000052f4f62	0t0	29856	/run/user/1000/systemd/notify type=DGRAM (UNCONNECTED)
systemd	1380	liveuser	17u	unix 0x00000000786eb39	0t0	29857	type=DGRAM (UNCONNECTED)
systemd	1380	liveuser	18u	unix 0x000000001695ea0c	0t0	29858	type=DGRAM (UNCONNECTED)
systemd	1380	liveuser	19u	unix 0x00000000f5ae567e	0t0	29859	/run/user/1000/systemd/private type=STREAM (LISTEN)
systemd	1380	liveuser	20u	unix 0x0000000085a0af4a	0t0	29861	type=STREAM (CONNECTED)
systemd	1380	liveuser	26u	unix 0x00000000dad12885	0t0	28025	/run/user/1000/pulse/native type=STREAM (LISTEN)
systemd	1380	liveuser	27u	unix 0x000000007149659d	0t0	26916	/run/user/1000/pipewire-0 type=STREAM (LISTEN)
systemd	1380	liveuser	28u	unix 0x000000002d06b039	0t0	28986	type=STREAM (CONNECTED)
gdm-wayla	1411	liveuser	1u	unix 0x000000004034074b	0t0	28979	type=STREAM (CONNECTED)
gdm-wayla	1411	liveuser	2u	unix 0x0000000023f05c4f	0t0	28980	type=STREAM (CONNECTED)
gdm-wayla	1411	liveuser	5u	unix 0x00000000fa37989d	0t0	28091	type=STREAM (CONNECTED)
gdm-wayla	1411	liveuser	7u	unix 0x0000000086a2370e	0t0	28988	type=STREAM (CONNECTED)
dbus-brok	1413	liveuser	1u	unix 0x00000000e41f370	0t0	26950	type=STREAM (CONNECTED)
dbus-brok	1413	liveuser	2u	unix 0x00000000e41f370	0t0	26950	type=STREAM (CONNECTED)
dbus-brok	1413	liveuser	3u	unix 0x00000000737a679c	0t0	28023	/run/user/1000/bus type=STREAM (LISTEN)
dbus-brok	1413	liveuser	4u	unix 0x000000007869fbc4	0t0	29972	type=DGRAM (UNCONNECTED)
dbus-brok	1413	liveuser	9u	unix 0x00000000e2060eb4	0t0	29976	type=STREAM (CONNECTED)
dbus-brok	1413	liveuser	10u	unix 0x000000007bb139e	0t0	29978	type=STREAM (CONNECTED)

- **+d** - выяснить, какие папки и файлы открыты в некоей директории (но не в её поддиректориях): `lsof +d /usr/bin`
- **-d** - задать список дескрипторов файлов, разделённых запятой, которые надо включить в вывод или исключить из него

Example

Список исключается из вывода, если все записи в наборе начинаются со знака `^`. Список будет включён в вывод, если ни одна запись не начинается с `^`. Смешивание записей разных видов не разрешается.

Note

В списке может присутствовать диапазон номеров дескрипторов файлов при условии, что ни один из его членов не пуст, оба члена являются числами, и завершающий член больше начального - то есть: «0-7» или «3-10».

☰ Example

Диапазоны могут быть использованы для исключения записей из вывода, если перед ними стоит префикс `^`, то есть `- ^0-7` исключает все дескрипторы с 0 по 7.

- `-p` - вывести все файлы, открытые процессом с указанным при вызове команды PID
- И другие ключи. Перечислять их всех я не вижу смысла. Зайдите [сюда](#), чтобы узнать больше. И, конечно же, `man lsof`.

13.2.3 Строение Linux - часть 3. Права доступа.

Во второй части речь была про концепцию "всё есть файл" и типы файлов, поэтому в этой части статьи будет рассказано про права доступа к файлам.

История

В отличии от DOS и подобных систем, Unix проектировался как многопользовательская ОС. Поэтому в Unix должна быть хорошая система управления доступа к файлам. А Linux - *Unix-подобное ядро*, поэтому и управление доступом было взято именно из Unix.

Теория

Чтобы получить доступ к определённому файлу в Linux, используются разрешения, которые назначаются трём объектам: **файлу**, **группе** и другому объекту (т.е. всем остальным). Но перед этим нужно знать про владельца файла (директории не упоминаются специально, так как это тоже файл - концепция "Всё есть файл" в UNIX-подобных/образных).

В Linux у каждого файла есть два владельца: пользователь и группа. Они устанавливаются при создании файла. Каждый файл имеет три категории пользователей, для которых можно устанавливать различные сочетания прав доступа: *

Владелец - набор прав для владельца файла - пользователя, который создал его или сейчас установлен его владельцем. Обычно владелец имеет все права: **чтение**, **запись** и **исполнение** * **Группа** - любая группа пользователей, существующая в ОС и привязанная к файлу * **Остальные** - все пользователи (кроме владельца и юзеров, входящих в группу файла)

Только пользователь *root* (он же суперпользователь) может работать со всеми файлами независимо от набора их полномочий.

Пользователь, создавший файл, становится его владельцем, так же как и первичная группа, в которую он входит. Чтобы определить, есть ли у вас, как у пользователя, права доступа к файлу, оболочка проверяет владение им. Принцип работы таков: * Оболочка проверяет, являетесь ли вы владельцем файла, к которому запрашивается доступ. Если владельцем является, то оболочка прекращает проверку и вы получаете разрешения. Если вы не являетесь владельцем, но входите в группу, у которой есть доступ к файлу, то вы получаете доступ к файлу с теми же разрешениями, что и у той группы. * Если же вы не являетесь ни пользователем, ни владельцем группы, то вы получаете права других пользователей (*other*).

Каждый пользователь может получить полный доступ к файлу в том случае, если доступ ему разрешён. Либо же, если он является владельцем файла.

Чтобы увидеть пользователя и группу-владельца файла, выполните:

```
1  ls -l
```

Чтобы просмотреть тоже самое, но для конкретного файла:

```
1  ls -l 'FILE'
```

FILE замените на нужный файл.

```
[liveuser@administrator-300e5c Downloads]$ ls -l
total 624
-rw-r--r--. 1 root      liveuser 137265 May  9 18:43 1.jpg
-rw-r--r--. 1 liveuser  liveuser 178595 May  9 18:43 2.jpg
-rw-r--r--. 1 liveuser  liveuser 315538 May  9 13:31 S5NkSFH_WXY.jpg
[liveuser@administrator-300e5c Downloads]$
```

Изменение владельца файла

Иногда нужно изменить владельца файла. Например, я это часто делал при компиляции LFS. Особенно при выполнении некоторых тестов сборки пакетов, которые выполнять от имени `root` опасно, а от имени менее привилегированного пользователя очень даже кстати.

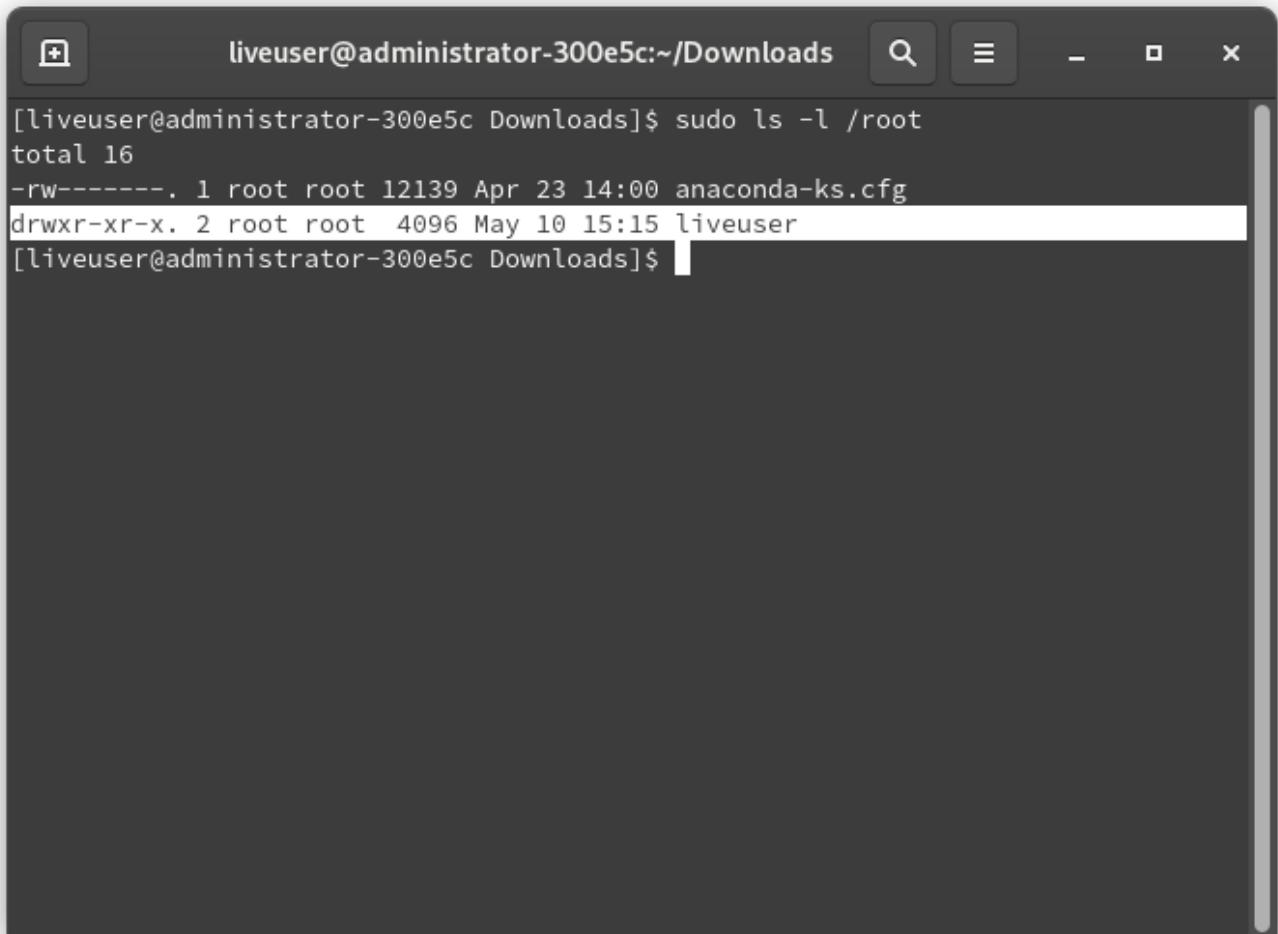
Для смены владельца используется `chown` - change owner. Синтаксис этой команды очень прост:

```
1 chown пользователь опции /путь/к/файлу
```

Ключи и опции CHOWN

- `-c` `--changes` - подробный вывод всех выполняемых операций
- `-v` `--verbose` - описание действий и вывод подробных данных о каждом обработанном файле
- `-R` `--recursive` - рекурсивная обработка всех подкаталогов
- `-f` `--silent` `--quiet` - минимум информации, выводимой на экран, даже сообщения об ошибках
- `--dereference` - изменять права для файла, к которому ведёт символьическая ссылка вместо самой ссылки (*по умолчанию*)
- `--no-dereference` `-h` - изменять права симлинков, но оставить неизменными файлы, на которые указывают эти симлинки
- `-L` - переходить по всем симлинкам на каталоги
- `-H` - если передан симлинк на каталог, перейти по нему
- `-P` - не переходить по символическим ссылкам на каталоги (*по умолчанию*)

Примеры использования chown



```
liveuser@administrator-300e5c:~/Downloads
[liveuser@administrator-300e5c Downloads]$ sudo ls -l /root
total 16
-rw-----. 1 root root 12139 Apr 23 14:00 anaconda-ks.cfg
drwxr-xr-x. 2 root root 4096 May 10 15:15 liveuser
[liveuser@administrator-300e5c Downloads]$
```

Допустим, есть два пользователя: `root` и `liveuser`. В директории `/root` создана поддиректория `/root/liveuser`. Там ещё несколько файлов. Эта поддиректория принадлежит пользователю `root`, как и все файлы в ней. А надо сделать так, чтобы принадлежала пользователю `liveuser`. Выполнить в терминале:

```
1 sudo chown liveuser /root/liveuser
```

Если вы хотите видеть подробную информацию о проделанном действии, выполните:

```
1 sudo chown -v liveuser /root/liveuser
```

```
[root@administrator-300e5c liveuser]# chown liveuser /root/liveuser
[root@administrator-300e5c liveuser]# ls /root
anaconda-ks.cfg  liveuser
[root@administrator-300e5c liveuser]# ls -l /root
total 16
-rw-----. 1 root      root 12139 Apr 23 14:00 anaconda-ks.cfg
drwxr-xr-x. 11 liveuser root  4096 May 10 17:25 liveuser
[root@administrator-300e5c liveuser]#
```

Но вы сменили владельца только для каталога `/root/liveuser`. А все подпапки и другие файлы, которые находятся в нём, всё так же принадлежат пользователю `root`. В тех подкаталогах ещё какие-то каталоги находятся. Можно, конечно, для каждого файла/каталога изменить владельца персонально, но это утомительно, а в некоторых случаях невозможно вообще. На этот случай у есть ключ `-R`, задающий утилите `chown` рекурсивно обойти все файлы и каталоги, изменив их права:

```
1 sudo chown -R liveuser /root/liveuser/
```

Как вы видите, все файлы из `/root/liveuser` теперь принадлежат пользователю `liveuser`, а не `root`. Однако, мы сменили только владельца, а не группу файла/ов.

Опять возвращаем всё в то состояние, которое было до экспериментов с `chown`:

```
1 sudo chown -Rv root /root/liveuser
```

Если вы хотите поменять не только владельца, но и группу файла, то запишите имя пользователя и имя группы через двоеточие:

```
1 sudo chown -v liveuser:liveuser /root/liveuser
```

Теперь измените группу и владельца на `liveuser` только для тех файлов, у которых владелец и группа `root` в каталоге `/root/liveuser/`:

```
1 sudo chown --from=root:root liveuser:liveuser ./
```

Изменение группы файла

Для изменения группы файла используется команда `chgrp` (*change group*). В отличии от предыдущей описанной команды `chown`, для `chgrp` требуется только имя группы, имя пользователя не нужно.

Синтаксис этой команды очень прост:

```
1 chgrp опции имя_группы /путь/к/директории
```

Note

Вместо имени группы можно указать её `GID` (идентификатор группы)

Ключи и опции CHGRP

- `-c --changes` - подробно описывать действия для каждого файла, чья группа изменяется
- `-f --silent --quiet` - не выдавать сообщения об ошибке для файлов, чья группа не может быть изменена
- `-h --no-dereference` - работать с символьными ссылками, а не файлами, на которые они указывают. Данная опция доступна, только если используется `lchown`
- `-v --verbose` - подробно описывать действие или отсутствие для **каждого** файла
- `-R` - рекурсивно изменить группы для каталогов и их содержимого, а возникающие ошибки не прекратят работу программы
- **-L (используется вместе с -R)** - для каждого файла, указанного или пользователем, или встреченного при обходе дерева каталогов, если этот файл является симлинком на каталог, изменить группу самого этого каталога и всех файлов в его иерархии
- `-h` - для каждого файла, являющегося символьической ссылкой, изменить группу самой этой ссылки, а не объекта, на который она указывает, а если система не поддерживает группы для симлинков, то ничего не делать.
- `--` - завершение списка опций

Примеры использования chgrp

Опять же, перед изменением группы-владельца файла, проверьте с помощью `ls` текущую группу:

```
1 ls -l 'ИМЯ_ФАЙЛА'
```

Теперь добавьте некую группу `mygroup`:

```
1 sudo groupadd mygroup
```

И сделайте группу `mygroup` владельцем директории `/root/liveuser`:

```
1 sudo chgrp mygroup /root/liveuser
```

Как и в случае с `chown`, группа стала владельцем только каталога `/root/liveuser`, но не файлов и подкаталогов этой директории. Чтобы изменить группу-владельца файла, можно, конечно, сделать вручную, а можно воспользоваться рекурсивным способом. За это отвечает ключ `-R`:

```
1 sudo chgrp -R mygroup /root/liveuser/
```

Однако, рекурсивный метод не назначает прав на симлинки, поэтому у них сохраняется прежняя группа. Чтобы установить новую группу и на символические ссылки, добавьте ключ `-h`:

```
1 sudo chgrp -Rh mygroup /root/liveuser
```

Для того, чтобы скопировать группу владельцев директории, воспользуйтесь ключом `--reference`:

```
1 sudo chgrp --reference /home/liveuser /root/liveuser
```

Специальные права доступа к файлам

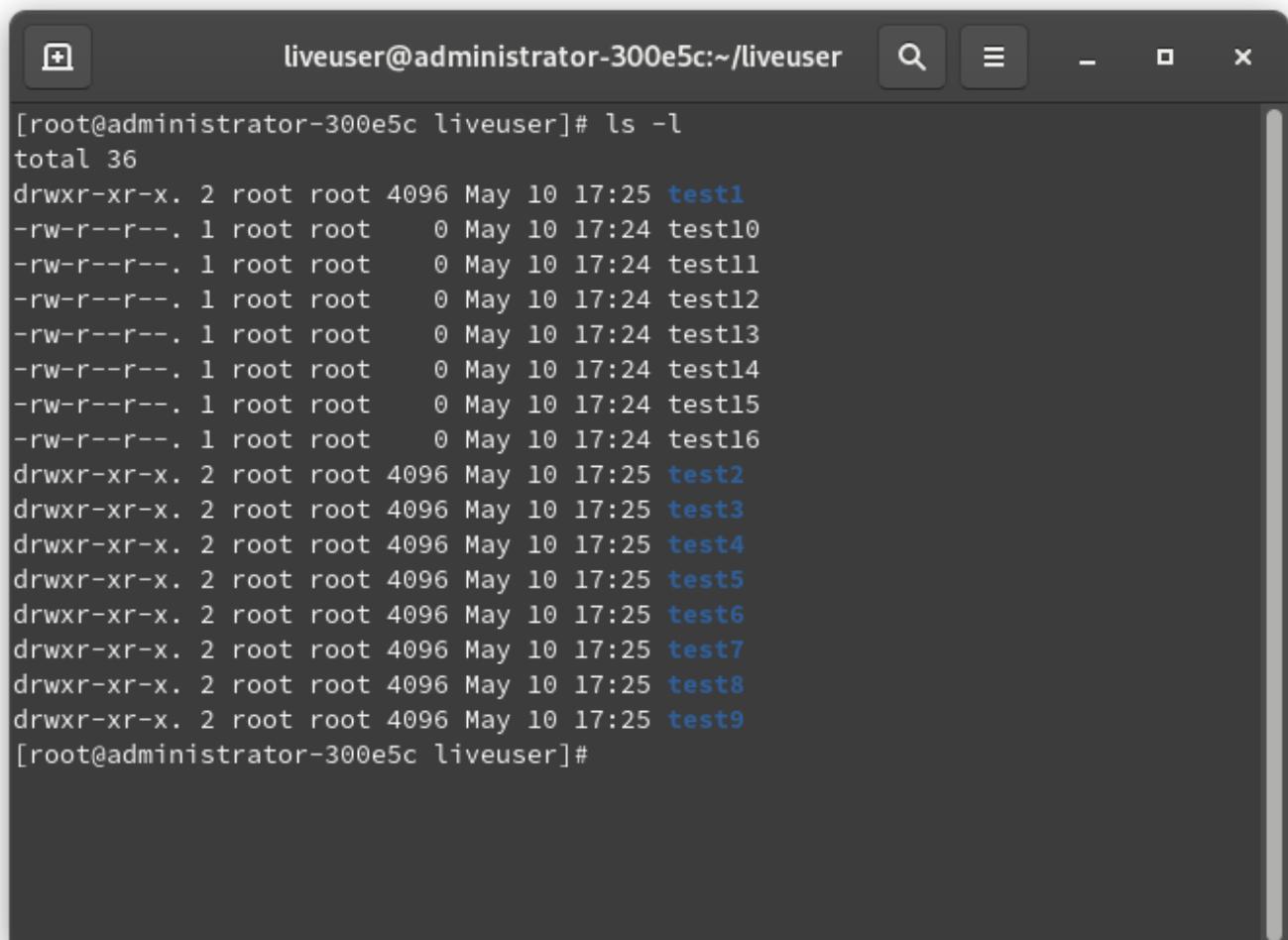
Для того, чтобы позволить *обычным* пользователям выполнять программы от имени суперпользователя, не зная его пароль, были созданы биты **SUID** и **SGID**. * Если установлен бит **SUID**, то при выполнении программы, ID пользователя меняется на ID владельца файла. *Фактически*, это позволяет обычным пользователям запускать программы от имени суперюзера. * **SGID** работает аналогичным способом, но разница в том, что юзер считается членом группы, с которой связан этот файл, а не групп, которым он действительно принадлежит. Если флаг **SGID** установлен на каталог, все файлы, созданные в нём, будут связаны с группой каталога, а не пользователя. Такое поведение используется для организации общих папок. * Бит **Sticky-bit** так же используется для создания общих директорий. Когда он установлен, пользователь может только создавать, читать и выполнять файлы, но не может удалять их, если они принадлежат другим пользователям.

Так, например, с помощью специальных прав доступа к файлам вы можете выполнять некоторые команды из **/sbin** и **/usr/sbin** от имени обычного пользователя.

Права на файлы - условные обозначения.

Опять же, чтобы узнать права на каждый файл, выполните:

```
1 ls -l
```



```
liveuser@administrator-300e5c:~/liveuser
ls -l
total 36
drwxr-xr-x. 2 root root 4096 May 10 17:25 test1
-rw-r--r--. 1 root root 0 May 10 17:24 test10
-rw-r--r--. 1 root root 0 May 10 17:24 test11
-rw-r--r--. 1 root root 0 May 10 17:24 test12
-rw-r--r--. 1 root root 0 May 10 17:24 test13
-rw-r--r--. 1 root root 0 May 10 17:24 test14
-rw-r--r--. 1 root root 0 May 10 17:24 test15
-rw-r--r--. 1 root root 0 May 10 17:24 test16
drwxr-xr-x. 2 root root 4096 May 10 17:25 test2
drwxr-xr-x. 2 root root 4096 May 10 17:25 test3
drwxr-xr-x. 2 root root 4096 May 10 17:25 test4
drwxr-xr-x. 2 root root 4096 May 10 17:25 test5
drwxr-xr-x. 2 root root 4096 May 10 17:25 test6
drwxr-xr-x. 2 root root 4096 May 10 17:25 test7
drwxr-xr-x. 2 root root 4096 May 10 17:25 test8
drwxr-xr-x. 2 root root 4096 May 10 17:25 test9
```

В первой колонке отображены права на файл. Вот условное обозначение каждого элемента:

Обозначение	Расшифровка
---	совсем нет прав
--x	разрешено только выполнение файла как программы, но не изменение и чтение
-w-	разрешена только запись и изменение файла
-wx	разрешено изменение и выполнение, но если это каталог, ещё и просмотр его содержимого
r--	права только на чтение
r-x	только чтение и выполнение, но не запись
rw-	чтение и запись, но не выполнение
rwx	все права
--s	установлен SUID или SGID бит, первый отображается в поле для владельца, второй для группы
--t	установлен Sticky-bit, из-за чего пользователи не могут удалить этот файл

Примеры использования chmod

Для изменения прав на файл используется `chmod`. Работа с ней такая же простая, как и с предыдущими утилитами. Вот её синтаксис:

```
1 chmod опции <категория><действие><флаг> файл
```

ПРАВА ДОСТУПА

В предыдущем разделе написал про права. Продублирую это и здесь, но покороче. Подобная таблица показывает "сокращения", которые используются для `chmod`, отображаются в выводе `ls` или `exa`, etc.

Обозначение	Расшифровка
r	чтение (Read)
w	запись (Write)
x	выполнение (eXecute)
s	выполнение от имени суперпользователя (Superuser) - дополнительный

И категории пользователей:

Обозначение	Расшифровка
u	владелец файла (User)
g	группа файла (Group)
o	все остальные пользователи (Other)

В качестве действий могут использоваться знаки + (включить) и - (отключить). Вот несколько примеров:

Обозначение	Расшифровка
u+x	разрешить выполнение для владельца
ugo+x	разрешить выполнение для всех
ug+w	разрешить запись для владельца и группы
o-x	запретить выполнение для остальных пользователей
ugo+rwx	разрешить все права для всех пользователей

Действия так же можно записывать и с помощью цифр. Первая цифра используется для указания прав для пользователя, вторая для группы и третья для всех остальных.

Числа	Действие	Обозначение в буквенном варианте
0	разрешения отсутствуют	---
1	x - запуск (выполнение, исполнение)	--x
2	w - изменение (запись)	-w-
3	x+w - исполнение и запись	-wx
4	r - чтение	r--
5	r+x - чтение и исполнение	r-x
6	r+w - чтение и изменение	rw-
7	r+w+x - чтение, изменение и запуск	rwx

Примеры цифровых действий:

Обозначение	Расшифровка
744	разрешить всё для владельца, а остальным только чтение
755	всё для владельца, остальным только чтение и запуск (исполнение)
764	всё для владельца, чтение и запись для группы, только чтение для остальных
777	всем разрешено всё

!> И тут очень большая опасность. Будьте осторожны при выполнении `chmod` и `chown`. Например, при выполнении `sudo chmod 777 /` вы, по сути, сломаете систему. Ошибка в вводе аргументов команды может привести к очень неожиданным последствиям.

ОПЦИИ И КЛЮЧИ CHMOD

- `-c` - выводить информацию обо всех изменениях
- `-f` - не выводить сообщения об ошибках
- `-v` - выводить максимум сообщений о ходе работы `chmod`
- `-R` - рекурсивный метод
- `--reference` - взять маску прав из указанного файла
- `--preserve-root` - не выполнять рекурсивные операции для корня файловой системы

Примеры работы с chmod

Разрешить выполнение определённого скрипта или бинарника пользователю. Допустим, у нас есть некий ELF файл, который называется `binary`. И его надо сделать исполняемым. Для начала выполните `ls -l |grep binary`, дабы просмотреть его текущие права.

```
1 chmod u+x binary
```

Теперь запускаю его:

```
1 ./binary
```

Тоже самое, но с помощью цифр:

```
1 chmod 766 binary
```

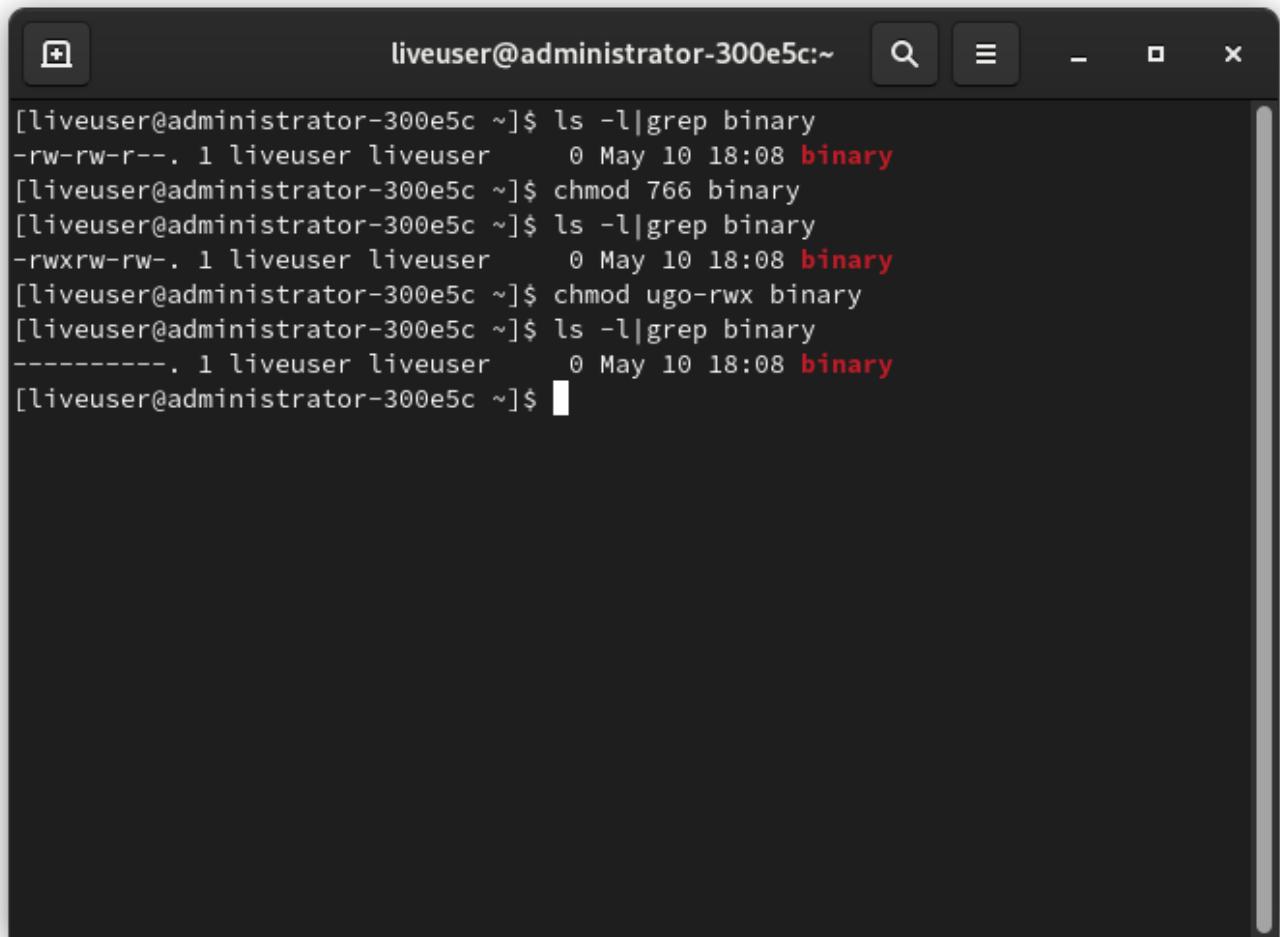
А теперь просмотрите, как изменились права на файл:

```
1 ls -l binary
```

```
liveuser@administrator-300e5c:~$ ls -l|grep binary
-rwx-rw-r--. 1 liveuser liveuser 0 May 10 18:08 binary
liveuser@administrator-300e5c:~$ chmod 766 binary
liveuser@administrator-300e5c:~$ ls -l|grep binary
-rwxrwx-rw-. 1 liveuser liveuser 0 May 10 18:08 binary
liveuser@administrator-300e5c:~$
```

А теперь отберите у `binary` все права:

```
1 chmod ugo-rwx binary
```



liveuser@administrator-300e5c:~

```
[liveuser@administrator-300e5c ~]$ ls -l|grep binary
-rw-rw-r--. 1 liveuser liveuser 0 May 10 18:08 binary
[liveuser@administrator-300e5c ~]$ chmod 766 binary
[liveuser@administrator-300e5c ~]$ ls -l|grep binary
-rwxrw-rw-. 1 liveuser liveuser 0 May 10 18:08 binary
[liveuser@administrator-300e5c ~]$ chmod ugo-rwx binary
[liveuser@administrator-300e5c ~]$ ls -l|grep binary
-----. 1 liveuser liveuser 0 May 10 18:08 binary
[liveuser@administrator-300e5c ~]$
```

Файлы с правами 000 недоступны никаким пользователям, кроме суперпользователя и владельца. Вернем права обратно:

```
1 chmod 755 binary
```

Для применения расширенных прав так же используется `chmod`. Нужно указать четырёхзначный аргумент в `chmod`, первая цифра относится к специальному разрешению, например:

```
1 sudo mkdir binary.d
2 sudo chmod 2755 binary.d
```

Эта команда добавит разрешение SGID на каталог `binary.d`.

ЧИСЛОВЫЕ ЗНАЧЕНИЯ SUID, SGID И STICKY BIT

- SUID - 4
- SGID - 2
- Sticky bit - 1

Ну и самое важное: * SUID - `chmod u+s` * SGID - `chmod g+s` * Sticky bit - `chmod +t`

Ещё немного про SUID, GUID. Понимание важного.

SUID

Рассмотрим простой пример. Нужно поменять пароль своей учётной записи. Для этого нужно отредактировать файл `/etc/shadow`. Но он доступен только суперпользователю. Однако, в утилите `/usr/bin/passwd` разрешение SUID применяется по умолчанию. В этом можно убедиться, просмотрев на этот самый `/usr/bin/passwd`:

```
1  ls -l /usr/bin/passwd
```

При смене пароля пользователь **ВРЕМЕННО** получает права root, что позволяет ему редактировать `/etc/shadow`, ибо есть разрешение SUID. Так ведь, если `passwd` именно это и делает? - редактирует нужный файл. Вот в этом и опасность разрешения SUID: с одной стороны, оно, однозначно, полезно и удобно, но довольно опасно. Поэтому пользуйтесь им с **осторожностью**.

SGID

А теперь второе.

`SGID` - идентификатор группы. SGID даёт пользователю, который исполняет определённый файл, разрешения владельца группы этого файла, что означает, что SGID позволяет выполнить примерно тоже самое, что и SUID. Но, как ни странно, SGID для этой цели если используется, то очень редко, но как в случае с SUID, он применяется к некоторым системным файлам по умолчанию.

Однако, SGID может быть полезен тогда, когда он применяется к каталогу: вы можете использовать его для установки владельца группы по умолчанию для файлов и подкаталогов, созданных в этом каталоге. По умолчанию, когда пользователь создает файл, его эффективная первичная группа устанавливается как владелец группы для этого файла.

STICKY-BIT

И, наконец, третье - `Sticky-bit`. Это разрешение полезно для защиты файлов от *случайного удаления* в среде, где несколько пользователей имеют права на запись в одну и ту же директорию; если применяется закреплённый sticky-bit, пользователь может удалить файл, только если он является пользователем-владельцем файла/каталога, в котором содержится файл. Именно поэтому он применяется, скажем, в `/tmp`.

Без sticky bit, если пользователь может создавать файлы в каталоге, он также может удалять файлы из этого каталога. В общедоступной групповой среде это может раздражать. Представьте себе пользователей `linda` и `lori`, которые оба имеют права на запись в каталог `/data/account` и получают эти разрешения благодаря участию в группе `account`. Поэтому `linda` может удалять файлы, созданные `lori`, и наоборот.

Когда вы применяете sticky bit, пользователь может удалять файлы, только если выполняется одно из следующих условий:
 * пользователь является владельцем файла * пользователь является владельцем директории с этим файлом

Увидеть sticky-bit можно, выполнив:

```
1  ls -ld
```

Увидите букву `t` в той позиции, где вы обычно видите разрешение на выполнение для других.

Практика

Введите в терминале:

```
1  sudo chmod -x /usr/bin/chmod
```

Т.е., вы сняли бит исполнения у `chmod`. И... Всё. А как вернуть? Чтобы можно было запускать `chmod`? С помощью него же самого не возможно, поэтому рядовой пользователь Linux переустановил бы пакет `coreutils`. Но можно и проще:

```
1  ldd /usr/bin/chmod
2  sudo /usr/lib64/ld-linux-x86-64.so.2 /usr/bin/chmod ugo+x /usr/bin/chmod
3
4  ls -l $(which chmod)
```

Так вы вернули бит исполнения программе `chmod`.

!> Настоятельно не рекомендуем производить подобные эксперименты с другими системными файлами!

Смотрите также:

- `man chmod`
- `man chown`
- `man ls`
- `man ldd`

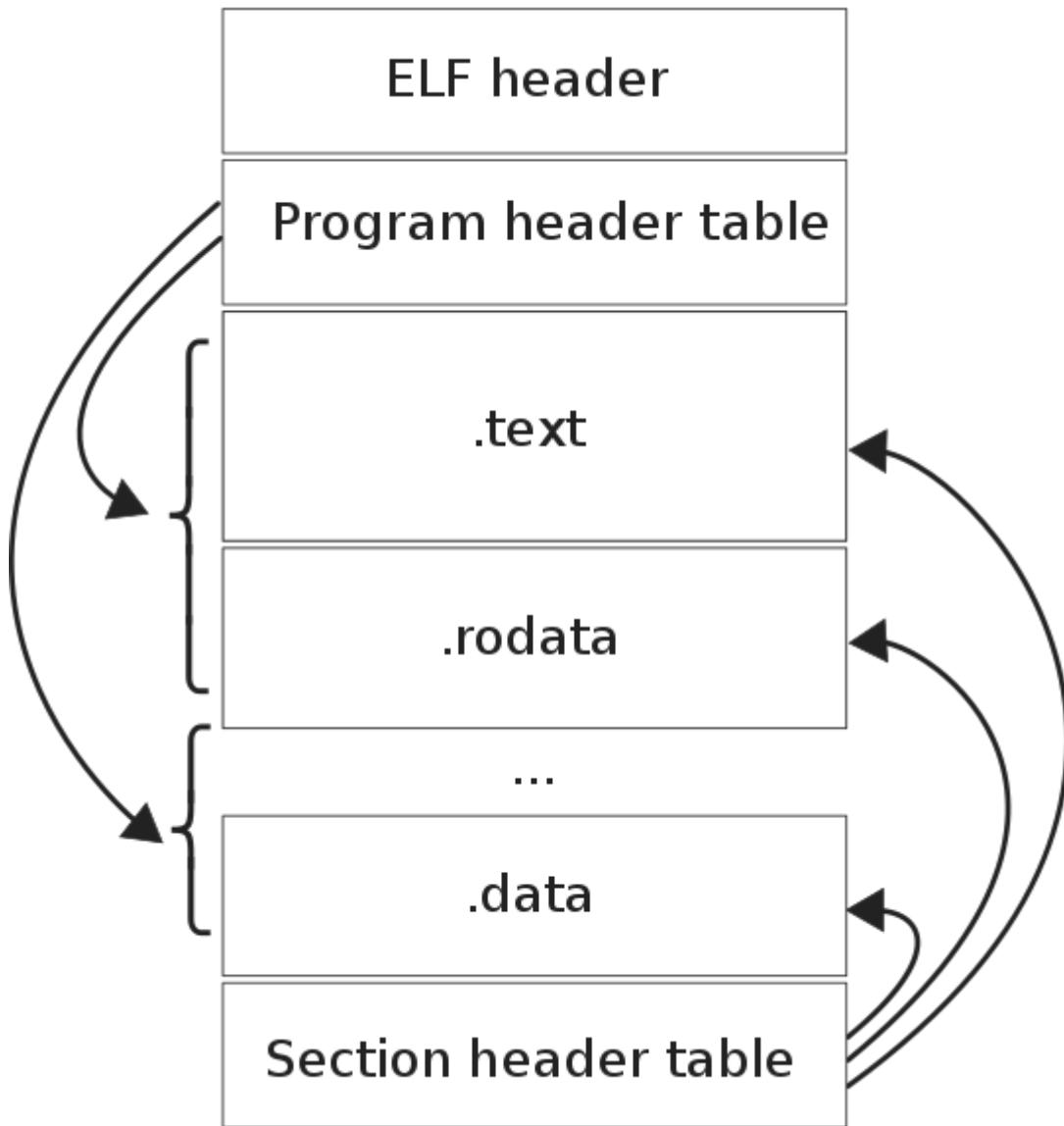
13.2.4 Строение ELF-файлов

Теория

ELF - сокращение от "Executable and Linkable Format" - формат исполняемых и связываемых файлов. ELF определяет их структуру. Данная спецификация позволяет UNIX-подобным(/образным) системам правильно интерпретировать содержащиеся в файле машинные команды. Используется во многих операционных системах: GNU/Linux, FreeBSD, Solaris, etc.

Понимание строения ELF файла может редко пригодиться, но, тем не менее, оно будет полезно для понимания процесса разработки программного обеспечения, поиска дыр в безопасности и обнаружения подозрительных программ или файлов.

Начальное строение



Для начала создадим директорию, в которой будут расположены тестовые программы, на которых будем "упражняться":

```
1  mkdir ~/LinuxPrograms
2  cd ~/LinuxPrograms
```

типы

Есть несколько типов ELF файлов (см. таблицы в конце статьи): * **Перемещаемый файл** - хранит инструкции (и данные), которые могут быть связаны с другими объектными файлами. Результатом может быть объектный или исполняемый файл. Так же к этому типу относятся объектные файлы статических библиотек. * **Разделяемый объектный файл** - также как и первый тип, содержит инструкции и данные, может быть связан с другими перемещаемыми и разделяемыми объектными файлами, в результате чего будет создан новый объектный файл, либо же при запуске программы ОС может динамически связывать его с исполняемым файлом программы, в результате чего будет создан исполняемый образ программы (в посл. случае речь идёт о разделяемых библиотеках). * **Исполняемый файл** - содержит полное описание, позволяющее ОС создать образ процесса. В т.ч.: *инструкции, данные, описания необходимых разделяемых объектных файлов* и др.

Для того, чтобы вывод всех команд, приведённых ниже, был краток, прост и понятен, напишите какую-нибудь простейшую программу, в которой нет **ничего** лишнего, что затруднит чтение:

```
1  vim simple.c
2
3  int main() {
4      return(0);
5 }
```

И скомпилируйте её:

```
1  gcc -o simple simple.c
```

Убедитесь в том, что это ELF файл:

```
1  file simple
```

Структура у каждого файла может различаться. Грубо говоря, ELF файл состоит из: * Заголовка * Данных

Подробнее: * **Таблица заголовков программы**: 0 или более сегментов памяти (только в исполняемом файле). Сообщает, как исполняемый файл должен быть помещён в виртуальную память процесса. Это необходимо для образа процесса, исполняемых файлов и общих объектов. Для перемещаемых объектных файлов это не требуется. * **Таблица заголовков разделов**: 0 или более разделов. Сообщает, как и куда нужно загрузить раздел. Каждая запись раздела в таблице содержит название и размер раздела. Таблица заголовков раздела должна использоваться для файлов, используемых при редактировании ссылок. * **Данные**: таблицы заголовка программы или раздела * **Заголовок ELF** (54/64 байта для 32/64 бит): определяет использование 32/64 бит (смотреть `struct Elf32_Ehdr` / `struct Elf64_Ehdr` в `/usr/include/elf.h`) * **Заголовок программы**: как создать образ процесса. Используются во время выполнения. Сообщают ядру или компоновщику время выполнения `ld.so`, что загружать в память и как найти информацию о динамической компоновке. * **Заголовок разделов**: используются во время компоновки или компиляции. Сообщают редактору ссылок `ld`, как разрешать символы и как группировать похожие потоки байтов из разных двоичных объектов ELF.

-*-*--

- **Разделы** - самые мелкие неделимые единицы в ELF файле, которые могут быть обработаны. Разделы содержат основную часть информации об объектных файлах для представления связывания. Эти данные включают инструкции, таблицу символов и информацию о перемещении. (просмотр ссылок)
- **Сегменты** - наименьшие отдельные единицы, которые могут быть отображены в памяти с помощью `exec` или компоновщика. (исполнимые)

Разделы и сегменты не имеют определённого порядка в ELF. Только заголовок имеет фиксированную позицию.

При помощи утилиты `readelf` можно просмотреть основную информацию о файле.

Эта утилита входит в состав пакета `binutils`, поэтому ничего доустанавливать не надо.

Основные возможности `readelf`:

- Просмотр заголовка файла:

```
1  readelf -h simple
```

- Просмотр информации о сегментах и сейкциях:

```
1  readelf -S -W simple
```

- Чтение информации о символах:

```
1  readelf -s -W simple
```

ЗАГОЛОВОК

Введите:

```
1  readelf -h simple
```

```

[administrator@fedora ~]$ readelf -h simple
Заголовок ELF:
Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Класс: ELF64
Данные: дополнение до 2, от младшего к старшему
Version: 1 (current)
OS/ABI: UNIX - System V
Версия ABI: 0
Тип: EXEC (Исполняемый файл)
Машина: Advanced Micro Devices X86-64
Версия: 0x1
Адрес точки входа: 0x401020
Начало заголовков программы: 64 (байт в файле)
Начало заголовков раздела: 23192 (байт в файле)
Флаги: 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: 29
Section header string table index: 28
[administrator@fedora ~]$

```

Заголовок является обязательным - он служит для того, чтобы данные корректно интерпретировались при линковке и исполнении.

Из вывода утилиты `readelf` следует, что заголовок начинается с т.н. *магического числа* (magic number). Это число содержит информацию о файле. Первые 4 байта определяют, что это ELF: 45 4c 46 .

После типа файла следует поле класса (архитектура, для которой предназначен бинарник).

Значения: * 0 - некорректный класс * 1 - 32 бит * 2 - 64 бит

Ниже находится поле данных. Это зависиткий от процессора метод кодирования данных.

Значения: * 0 - некорректный тип * 1 - LittleEndian (LSB) * 2 - BigEndian (MSB)

Разные типы процессоров по разному обрабатывают структуры данных, а эти значения помогают правильно интерпретировать объекты в файле.

Эффект LSB становится видимым при использовании утилиты `hexdump` на бинарном файле. Просмотрите заголовок ELF у нашего файла `simple`:

```
1 hexdump -n 16 simple
```

Получите такой вывод:

```
1 00000000 457f 464c 0102 0001 0000 0000 0000 0000
2 00000010
```

Пары значений другие из-за интерпретации порядка данных.

Затем следует поле "Версия". На данный момент, используется только версия 01.

Каждая ОС имеет свой способ вызова функций. Что-то похоже, а что-то различается. В поле OS/ABI описываются специфичные для операционной системы или ABI расширения, используемые в файле. В некоторых других структурах ELF файла имеются флаги и поля, значения которых зависят от ОС или ABI, интерпретация этих полей определяется значением данного байта. В таблице ниже представлена таблица значений:

Значение	Описание
0	UNIX System V
1	HP-UX
2	NetBSD
3	GNU ELF (GNU/Linux)
6	Solaris
7	AIX
8	IRIX
9	FreeBSD
10	Tru64 UNIX
11	Modesto
12	OpenBSD
13	OpenVMS
15	Amiga Research OS
18	OpenVOS

При необходимости, так же может быть указана версия ABI.

В поле "Машина" указывается архитектура аппаратной платформы, для которой предназначен файл. В таблице ниже представлены некоторые из них:

Значение	Описание
0	Не определено
3	Intel 80386
20	PowerPC
21	PowerPC (64 бит)
62	x86_64

В поле "Тип" указывается предназначение файла. В таблице ниже они приведены:

Значение	Описание	Значение поля
0	Некорректный тип	---
1	Перемещаемый файл (файл до линковки)	REL
2	Исполняемый файл	EXEC
3	Разделяемый объектный файл (библиотека)	DYN
4	Core file	CORE

(смотрите блок "Типы" этой статьи, чтобы узнать больше информации).

Продолжение

Помимо заголовка, есть данные. Оно, в свою очередь, подразделяется ещё на несколько: * программные заголовки (сегменты) * заголовки секций/секции * данные

О первых двух пунктах было уже говорено в пункте "Типы".

Для начала. Файл ELF имеет два различных «вида». Один из них предназначен для линкера и разрешает исполнение кода (сегменты). Другой предназначен для команд и данных (секции). В зависимости от цели, используется соответствующий тип заголовка. Начнём с заголовка программы, который находится в исполняемых файлах ELF.

Продолжение про заголовки программы

ELF файл состоит из нуля и более сегментов. И описывает, как создать процесс, образ памяти для исполнения в рантайме. Когда ядро видит эти сегменты, оно размещает их в виртуальном адресном пространстве, используя системный вызов `mmap ...`

Смотрите также:

`man 2 mmap`

... Т.е., конвертирует заранее подготовленные инструкции в образ памяти. Для разделяемых библиотек (*shared libs*) процесс схож.

И примеры заголовков:

GNU_EH_FRAME

Сортированная очередь, используемая компилятором GCC. В неё хранятся обработчики исключений. Они используются для того, чтобы корректно обработать ситуацию, если что-то пошло не так.

GNU_STACK

Используется для сохранения информации о стеке. Он не должен быть исполняемым, так как это может быть очень небезопасным.

Если сегмент `GNU_STACK` отсутствует, то используется исполняемый стек. Для того, чтобы показать детали устр-ва стека, используйте утилиты `scanelf` и `execstack`.

Смотрите также:

`man scanelf`

`man execstack`

Секции

Секции появляются в файле после преобразования компилятором GNU C кода C в ассемблер (и ассемблер GNU создаёт объекты).

```

1:administrator@fedora:~/Work/LinuxSoviet/stats/LFS/LinuxStr4/programs
0 .interp 0000001c 000000000400318 000000000400318 00000318 2**0
    CONTENTS, ALLOC, LOAD, READONLY, DATA
1 .note.gnu.property 00000020 000000000400338 000000000400338 00000338 2**2
    CONTENTS, ALLOC, LOAD, READONLY, DATA
2 .note.gnu.build-id 00000024 000000000400358 000000000400358 00000358 2**2
    CONTENTS, ALLOC, LOAD, READONLY, DATA
3 .note.ABI-tag 00000020 00000000040037c 00000000040037c 0000037c 2**2
    CONTENTS, ALLOC, LOAD, READONLY, DATA
4 .gnu.hash 0000001c 0000000004003a0 0000000004003a0 000003a0 2**3
    CONTENTS, ALLOC, LOAD, READONLY, DATA
5 .dynsym 00000048 0000000004003c0 0000000004003c0 000003c0 2**3
    CONTENTS, ALLOC, LOAD, READONLY, DATA
6 .dynstr 00000038 000000000400408 000000000400408 00000408 2**0
    CONTENTS, ALLOC, LOAD, READONLY, DATA
7 .gnu.version 00000006 000000000400440 000000000400440 00000440 2**1
    CONTENTS, ALLOC, LOAD, READONLY, DATA
8 .gnu.version_r 00000020 000000000400448 000000000400448 00000448 2**3
    CONTENTS, ALLOC, LOAD, READONLY, DATA
9 .rela.dyn 00000030 000000000400468 000000000400468 00000468 2**3
    CONTENTS, ALLOC, LOAD, READONLY, DATA
10 .init 0000001b 000000000401000 000000000401000 00001000 2**2
    CONTENTS, ALLOC, LOAD, READONLY, CODE
11 .text 00000175 000000000401020 000000000401020 00001020 2**4
    CONTENTS, ALLOC, LOAD, READONLY, CODE
12 .fini 0000000d 000000000401198 000000000401198 00001198 2**2
    CONTENTS, ALLOC, LOAD, READONLY, CODE
13 .rodata 00000010 000000000402000 000000000402000 00002000 2**3
    CONTENTS, ALLOC, LOAD, READONLY, DATA
14 .eh_frame_hdr 00000034 000000000402010 000000000402010 00002010 2**2
    CONTENTS, ALLOC, LOAD, READONLY, DATA
15 .eh_frame 000000c0 000000000402048 000000000402048 00002048 2**3
    CONTENTS, ALLOC, LOAD, READONLY, DATA
16 .init_array 00000008 000000000403e50 000000000403e50 00002e50 2**3
    CONTENTS, ALLOC, LOAD, DATA
17 .fini_array 00000008 000000000403e58 000000000403e58 00002e58 2**3
    CONTENTS, ALLOC, LOAD, DATA
18 .dynamic 00000019 000000000403e60 000000000403e60 00002e60 2**3
    CONTENTS, ALLOC, LOAD, DATA
19 .got 00000010 000000000403e60 000000000403e60 00002e60 2**3
    CONTENTS, ALLOC, LOAD, DATA

```

.TEXT

Содержит исполняемый код, упакованный в сегмент с правами на чтение и исполнение (но не редактирование).

.DATA

Инициализированные данные с правами на чтение и запись.

.BSS

Неинициализированные данные с правами на чтение/запись

Ещё немного о секциях

Также, некоторую информацию о секциях можно просмотреть, написав скрипт на Python с применением библиотеки `lief`:

```

1 #!/bin/python
2 import lief
3
4 binary = lief.parse("simple")
5 header = binary.header
6 print("Entry point: %08x" % header.entrypoint)
7 print("Architecture: ", header.machine_type)
8
9 for section in binary.sections:
10     print("Section %s - size: %s bytes" % (section.name, section.size))

```

Запуск:

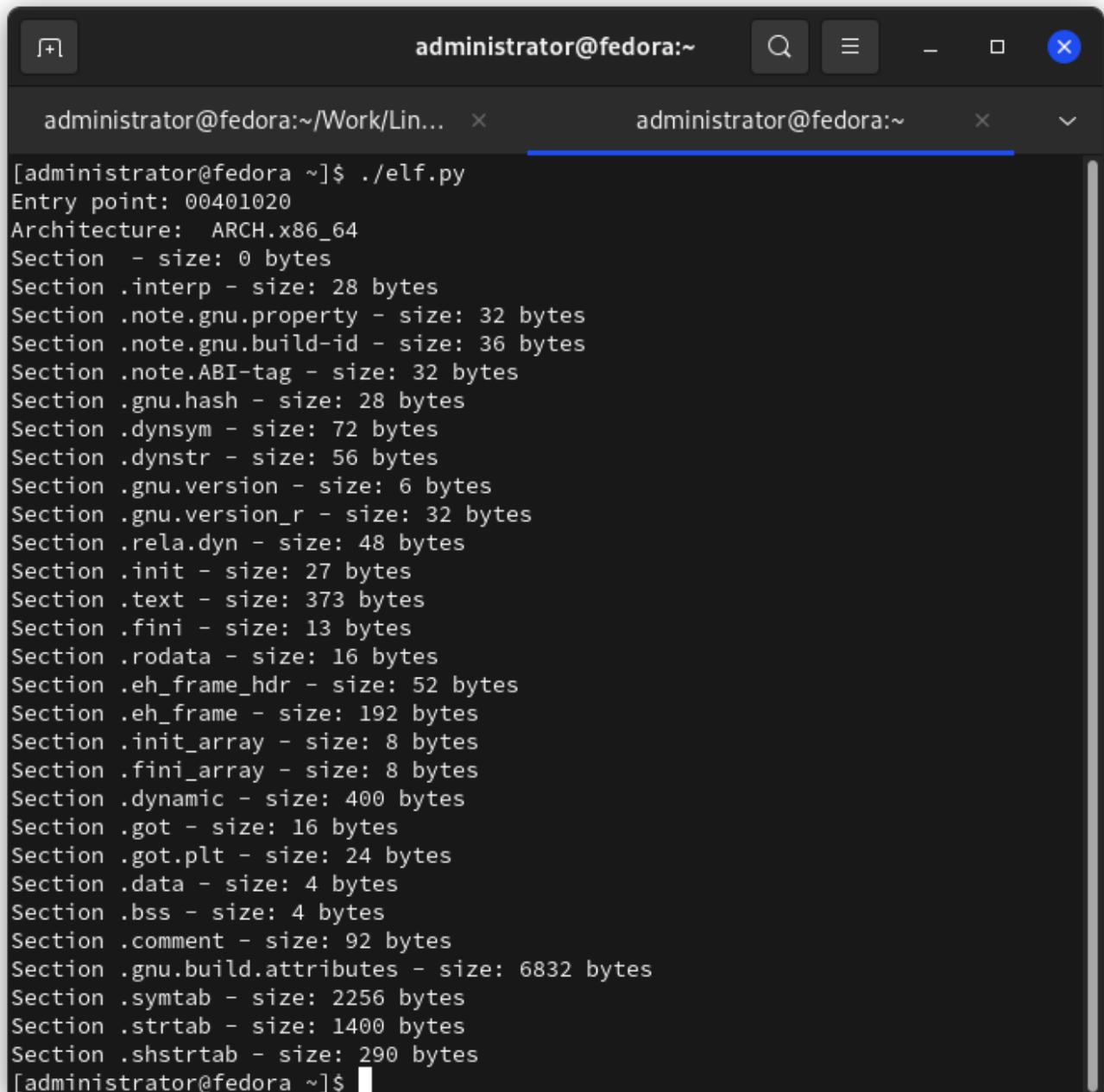
```

1 chmod +x elf.py
2 ./elf.py

```

Название `elf.py` замените на нужное.

Вывод такой:



```
[administrator@fedora ~]$ ./elf.py
Entry point: 00401020
Architecture: ARCH.x86_64
Section - size: 0 bytes
Section .interp - size: 28 bytes
Section .note.gnu.property - size: 32 bytes
Section .note.gnu.build-id - size: 36 bytes
Section .note.ABI-tag - size: 32 bytes
Section .gnu.hash - size: 28 bytes
Section .dynsym - size: 72 bytes
Section .dynstr - size: 56 bytes
Section .gnu.version - size: 6 bytes
Section .gnu.version_r - size: 32 bytes
Section .rela.dyn - size: 48 bytes
Section .init - size: 27 bytes
Section .text - size: 373 bytes
Section .fini - size: 13 bytes
Section .rodata - size: 16 bytes
Section .eh_frame_hdr - size: 52 bytes
Section .eh_frame - size: 192 bytes
Section .init_array - size: 8 bytes
Section .fini_array - size: 8 bytes
Section .dynamic - size: 400 bytes
Section .got - size: 16 bytes
Section .got.plt - size: 24 bytes
Section .data - size: 4 bytes
Section .bss - size: 4 bytes
Section .comment - size: 92 bytes
Section .gnu.build.attributes - size: 6832 bytes
Section .symtab - size: 2256 bytes
Section .strtab - size: 1400 bytes
Section .shstrtab - size: 290 bytes
[administrator@fedora ~]$
```

Смотрите также:

```
1 man readelf
2 man objdump
3 man mmap
4 man hexdump
5 man file
6 man gcc
```

13.3 Установка программ из исходного кода в Linux

Linux-системы неразрывно связаны с концепцией GNU – проекта, поддерживающего и развивающего философию свободно распространяемого программного обеспечения (ПО), в том числе и в виде исходного кода. А поскольку систем на базе ядра Linux существует великое множество и разработчики дистрибутивов всегда для своих систем используют исходный код ПО при сборке комплектов утилит, пакетов, да и самого ядра, то, очевидно, что использование исходных кодов ПО — это неотъемлемый аспект в эксплуатации Linux-систем. По крайней мере, любому пользователю, достаточно хорошо освоившему UNIX-подобные системы, рано или поздно приходится сталкиваться со сборкой ПО из исходного кода.

Но поскольку системы Linux, как правило, снабжены хранилищами пакетов (репозиториями), из которых происходит загрузка, установка и обновление ПО, то часто бывает так, что разработчики дистрибутива, которые и поддерживают репозитории, ещё не успели сформировать новые пакеты ПО, для которых уже выпущено обновление. В этом случае можно прибегнуть к самостоятельной сборке требуемых пакетов.

13.3.1 Общий порядок сборки пакетов — утилита make

Для облегчения сборки ПО из исходных кодов существует свободная утилита `make`. Она применяется во всех UNIX-подобных системах для подавляющего большинства утилит. При сборке пакета очень полезно изучать информацию, содержащуюся, как правило, в файлах `README` или `INSTALL`, входящих в пакет. В этих файлах разработчики ПО указывают инструкции и специфические мероприятия для успешной сборки пакетов. Здесь также можно найти и системные требования для работы ПО и описания необходимых зависимостей, без которых собрать пакет будет невозможно.

Порядок сборки выглядит так:

- Распаковка архива, содержащего файлы исходного кода (обычно именно так «исходники» и распространяются);
- Переход в директорию с распакованными исходными текстами;
- Подготовка (конфигурирование) предстоящей сборки (указание директорий установки, сторонних библиотек, архитектуры, дополнительных компонентов и т.д.). Для этого обычно используются служебные скрипты;
- Непосредственно, сама сборка — команда `make`;
- Установка (распространение) построенного ПО — например, командой `make install`.

Ниже будет приведена эта процедура на примере пакета Zlib.

Скачаем архив пакета:

```
1 wget https://zlib.net/zlib-1.2.11.tar.xz
```

Распакуем архив:

```
1 tar -xvf zlib-1.2.11.tar.xz
```

В результате в текущем каталоге появится еще один каталог, с распакованным пакетом. Перейдём в него:

```
1 cd zlib-1.2.11
```

Для успешной сборки и работы пакета необходимо проверить существующую конфигурацию системы на наличие требуемых зависимостей, библиотек и настроек, а также сконфигурировать сборку, запустив соответствующий скрипт `configure`:

```
1 ./configure
```

Подобные скрипты создаются разработчиками для облегчения процесса сборки/установки.

Вывод этого скрипта показывает, готов ли данный пакет к сборке:

```

1 Checking for gcc...
2 Checking for shared library support...
3 Building shared library libz.so.1.2.11 with gcc.
4 Checking for size_t... Yes.
5 Checking for off64_t... Yes.
6 Checking for fseeko... Yes.
7 Checking for strterror... Yes.
8 Checking for unistd.h... Yes.
9 Checking for stdarg.h... Yes.
10 Checking whether to use vs[n]printf() or s[n]printf()... using vs[n]printf().
11 Checking for vsnprintf() in stdio.h... Yes.
12 Checking for return value of vsnprintf()... Yes.
13 Checking for attribute(visibility) support... Yes.

```

Если вы хотите изменить место, куда должен впоследствии установиться пакет, то используйте ключ `--prefix=ДИРЕКТОРИЯ`:

```
1 ./configure --prefix=/usr
```

Это означает, что пакет впоследствии будет установлен в `/usr`.

Также по мере компиляции пакетов ключи у `configure` будут меняться. Для новых опций будет краткое описание.

Для просмотра всех доступных ключей и опций, выполните:

```
1 ./configure --help
```

Изучив вывод скрипта `configure`, можно сделать вывод о том, стоит ли далее приступать к сборке пакета. Обычно о критических ошибках сообщается фразами «`configure: error`». Убедившись, что всё нормально, можно приступать к построению:

```
1 make
```

Далее в консоль будет направлен вывод, отображающий ход сборки:

```

1 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -I. -c -o example.o test/example.c
2 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o adler32.o adler32.c
3 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o crc32.o crc32.c
4 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o deflate.o deflate.c
5 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o infback.o infback.c
6 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o inffast.o inffast.c
7 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o inflate.o inflate.c
8 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o inftrees.o inftrees.c
9 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o trees.o trees.c
10 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o zutil.o zutil.c
11 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o compress.o compress.c
12 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o uncompr.o uncompr.c
13 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o gzclose.o gzclose.c
14 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o zlib.o zlib.c
15 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o gzread.o gzread.c
16 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o gzwrite.o gzwrite.c
17 ar rc libz.a adler32.o crc32.o deflate.o infback.o inffast.o inflate.o inftrees.o trees.o zutil.o compress.o uncompr.o gzclose.o zlib.o gzread.o gzwrite.o
18 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o example example.o -L libz.a
19 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -I. -c -o minizip.o test/minizip.c
20 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o minizip minizip.o -L libz.a
21 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/adler32.adler32.c
22 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/crc32.crc32.c
23 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/deflate.deflate.c
24 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/infback.infback.c
25 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/inffast.inffast.c
26 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/inflate.inflate.c
27 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/inftrees.inftrees.c
28 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/trees.trees.c
29 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/zutil.zutil.c
30 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/compress.compress.c
31 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/uncompr.uncompr.c
32 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/gzclose.gzclose.c
33 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/zlib.zlib.c
34 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/gzread.gzread.c
35 gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o objs/gzwrite.gzwrite.c
36 gcc -shared -Wl,-soname,libz.so.1,-version-script,zlib.map -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o libz.so.1.2.11 adler32.o crc32.o deflate.o infback.o inffast.o
inflate.o inftrees.o trees.o zutil.o compress.o uncompr.o gzclose.o zlib.o gzread.o gzwrite.o -lc
37 rm -f libz.so libz.so.1
38 ln -s libz.so.1.2.11 libz.so
39 ln -s libz.so.1.2.11 libz.so.1
40 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o exampleh example.o -L libz.so.1.2.11
41 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o minizipsh minizip.o -L libz.so.1.2.11
42 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -I. -D_FILE_OFFSET_BITS=64 -c -o example64.o test/example.c
43 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o example64 example64.o -L libz.a
44 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -I. -D_FILE_OFFSET_BITS=64 -c -o minizip64.o test/minizip.c
45 gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o minizip64 minizip64.o -L libz.a

```

После успешного окончания которого можно произвести установку пакета (от пользователя root):

```
1 make install
```

Или вместе с командой sudo (если этот пакет установлен; выполняется эта команда от имени обычного пользователя):

```
1 sudo make install
```

В том случае, если вы собираете бинарный пакет для какого-либо пакетного менеджера (например, если вы написали его сами), то пакет нужно установить в отдельную директорию, а не в тот путь, который указан скриптом `configure`. Тогда укажите `make` переменную `DESTDIR`:

```
1 make DESTDIR=/путь/до/места/установки install
```

Пакет будет установлен в `$DESTDIR` (где `DESTDIR` — путь до нужной папки). Если в `configure` был указан, например, `--prefix=/usr`, то пакет будет установлен в `$DESTDIR/usr`. К примеру, создадим в директории сборки папку `PKG` и установим пакет туда:

```
1 mkdir PKG
2 make DESTDIR=$PWD/PKG install
```

Note

Указание переменной `$PWD` в таком случае желательно, если директория для установки находится в папке с исходным кодом, в которой выполняется сборка.

Некоторые системы сборки не поддерживают переменную `DESTDIR`, но поддерживают что-то аналогичное, например `INSTALL_DIR`. Либо `prefix`. Программы, собранные через `qmake`, устанавливаются через переменную `INSTALL_ROOT`:

```
1 make install INSTALL_ROOT="/путь/до/места/установки"
```

Некоторые системы сборки вообще не поддерживают такие переменные окружения, в этом случае файлы придётся копировать самому. Помните, что в той отдельной директории должна располагаться зеркальная иерархия корня системы, то есть так, как эти файлы должны лежать в системе со всеми подкаталогами. Например:

```
1 $ find PKG
2
3 PKG
4 PKG/usr
5 PKG/usr/include
6 PKG/usr/include/zconf.h
7 PKG/usr/include/zlib.h
8 PKG/usr/Lib
9 PKG/usr/Lib/libz.so
10 PKG/usr/Lib/libz.so.1.2.11
11 PKG/usr/Lib/libz.so.1
12 PKG/usr/Lib/pkgconfig
13 PKG/usr/Lib/pkgconfig/zlib.pc
14 PKG/usr/Lib/libz.a
15 PKG/usr/share
16 PKG/usr/share/man
17 PKG/usr/share/man/man3
18 PKG/usr/share/man/man3/zlib.3
```

На этом сборка из исходных кодов и установка пакета `zlib` успешно завершена.

13.4 Решение ошибок сборки

При сборке пакетов иногда происходят ошибки. В данном разделе будет описано решение наиболее распространённых ошибок.

13.4.1 Стадии, на которых может произойти ошибка

Ошибка может произойти на любой стадии, однако чаще всего это случается после ввода `make`. При этом в первую очередь определите действие, которое завершилось ошибкой - сделать это можно, просмотрев команду, завершившуюся с ошибкой. В частности, если команда даётся компилятору (`cc`, `gcc` или `clang`), то произошла ошибка компиляции. Эти ошибки, обычно, наиболее трудны в решении. Если команда даётся `ld`, то ошибка произошла при линковке. Также ошибка может произойти, например, при построении документации. В этом случае самым простым вариантом будет отключение выполнения этого шага.

13.4.2 Общие принципы решения ошибок

Убедитесь что ошибка воспроизводима - выполните `make clean`, а потом повторите `make`. Если ошибка не исчезла, то прочтайте лог (хотя бы последние 30 строк). Практически всегда там будет сказано о том, что за ошибка произошла.

Попробуйте поискать в интернете по частям лога, возможно, решение этой ошибки уже было где-либо описано.

13.4.3 Ошибки компиляции

Ошибки компиляции - наиболее сложные в своём решении. `gcc` всегда сообщает строку, в которой произошла ошибка - проверьте её.

Не найден заголовок

Весьма простая ошибка.

Вывод

```
1 dummy.c:1:10: fatal error: blablabla: No such file or directory
2      | #include <blablabla>
3      |
4 compilation terminated.
```

Имя заголовка и файла может быть другим.

Решение

Поиските этот заголовок в папке `/usr/include` и директории с исходным кодом пакета. Если он существует, то добавьте в переменную `CPPFLAGS` параметр `-I/путь/к/директории/с/этим/заголовком`. Если он не существует - установите пакет, который его предоставляет.

13.4.4 Ошибки линковки

В процессе линковки несколько объектных файлов соединяются в один, и к ним подключаются библиотеки.

`undefined reference to ...`

Данная ошибка вызвана тем, что необходимая библиотека не была подключена.

Решение

Попытайтесь определить, исходя из лога, какая библиотека не была подключена. Добавьте в переменную `CFLAGS` параметр `-lsomeplib` (не надо указывать название файла библиотеки), например, `-lcurses`.

13.4.5 Ошибки configure

Обычно они происходят из-за отсутствия зависимостей или их неработоспособности.

13.5 Кросс-компилятор

Кросс-компилятор (англ. `cross compiler`) - компилятор, производящий исполняемый код для платформы, отличной от той, на которой исполняется сам кросс-компилятор. Такой инструмент бывает полезен, когда нужно получить код для платформы, экземпляров которой нет в наличии, или в случаях когда компиляция на целевой платформе невозможна или нецелесообразна (например, это касается мобильных систем или микроконтроллеров с минимальным объёмом памяти). Например, компилятор, который работает на компьютере Windows 7, но и генерирует код, который работает на Android.

13.5.1 Применение кросс-компиляторов

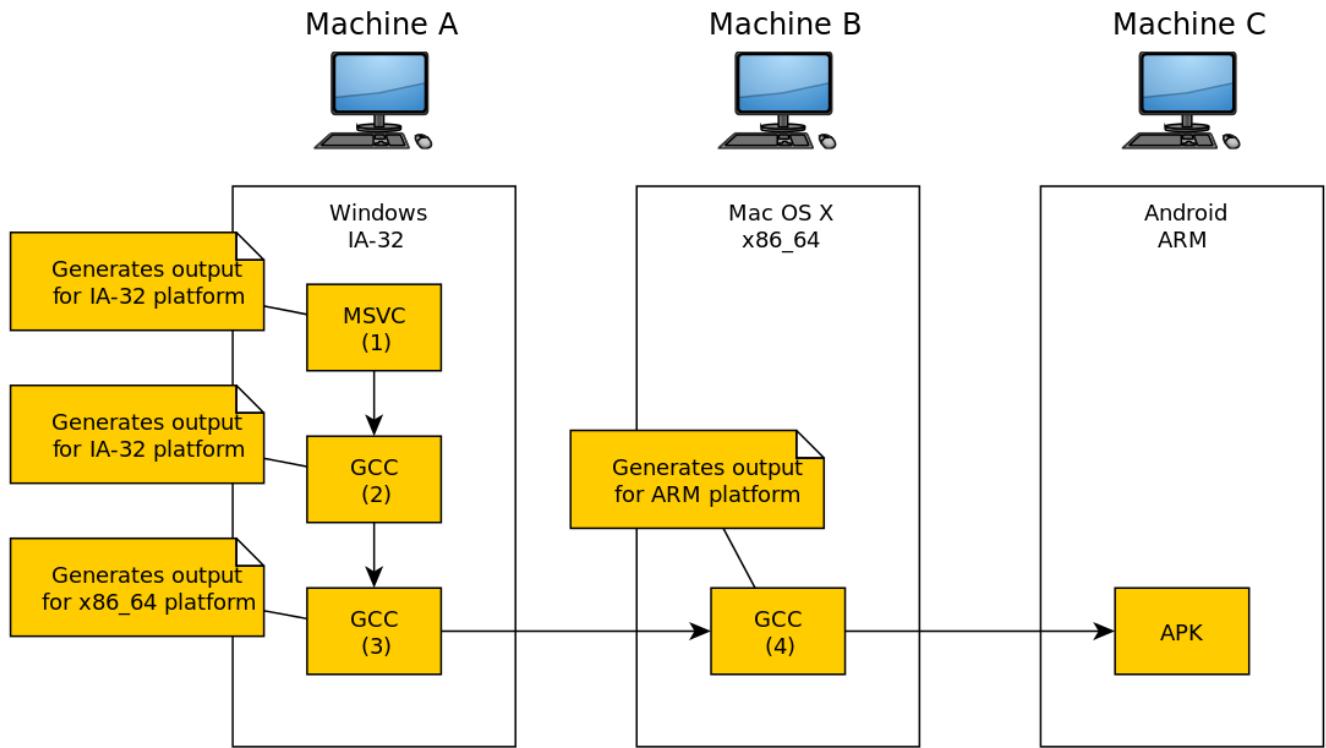
- Встроенные компьютеры, на которых ресурсы устройства крайне ограничены.
- Компиляция для нескольких машин. Например, компания может пожелать поддерживать несколько разных версий операционной системы или поддерживать несколько разных операционных систем. Используя кросс-компилятор, можно настроить единую среду сборки для компиляции для каждой из этих целей.
- Компиляции на ферме серверов. Подобно компиляции для нескольких машин, сложная сборка, которая включает в себя множество операций компиляции, может быть выполнена на любой свободной машине.
- Самонастройка на новую платформу. При разработке программного обеспечения для новой платформы или эмулятора будущей платформы используется кросс-компилятор, чтобы скомпилировать необходимые инструменты.
- Компиляция машинного кода для эмуляторов для уже устаревших платформ, таких как Commodore 64 или Apple II.

13.5.2 Канадский крест

Канадский крест - это техника построения кросс-компиляторов для других машин. Система конфигурирования и сборки GNU позволяет собирать программы, которые запускаются на системе, отличной от той, на которой собирались необходимые средства. Иными словами, она поддерживает сборку программ с помощью кросс-компилятора.

При использовании канадского креста с GCC могут быть четыре вида:

- Фирменный родной компилятор для машины (1) (например, компилятор из Visual Studio) используется для построения собственного компилятора GCC для машины (2).
- Родной компилятор GCC для машины (2) используется для сборки кросс-компилятора GCC с компьютера А на компьютер Б (3)
- GCC кросс-компилятор с компьютера А на компьютер Б (3) используется для построения компилятора GCC кросс-компилятор из машины в машину С (4)



Схема

Конечный кросс-компилятор (4) не сможет работать на машине сборки А; вместо этого он будет работать на машине В, чтобы скомпилировать приложение в исполняемый код, который затем будет скопирован на машину С и выполнен на машине С.

Термин «канадский крест» возник потому, что в то время, когда эти проблемы обсуждались, в Канаде было три национальные политические партии.

13.5.3 Кросс-компиляция с GCC

GCC, коллекция свободно распространяемых компиляторов, может быть настроена для кросс-компиляции. Она поддерживает множество платформ и языков.

Для кросс-компиляции с GCC необходимо, чтобы была доступна скомпилированная для целевой платформы версия binutils. Особенно важно наличие GNU Assembler. Поэтому binutils должны быть предварительно скомпилированы с ключом `--target=some-target`, указанным скрипту конфигурирования (англ.). GCC также должна быть указана опция `--target` с аналогичным содержанием. После этого, чтобы GCC могла использовать полученные binutils, надо поместить путь к ним в переменную окружения `path`, например:

```
1 PATH=/path/to/binutils/bin:${PATH} make
```

Кросс-компиляция GCC требует, чтобы часть стандартной библиотеки целевой платформы была доступна на хост-платформе. Программист может решить скомпилировать полную библиотеку С, но этот выбор может быть ненадежным. Альтернативой является использование файла, который представляет собой небольшую библиотеку С, содержащий только самые важные компоненты, необходимые для компиляции исходного кода С.

13.6 Выбор размера файла подкачки

Если оперативной памяти Вашего компьютера недостаточно (3 Гб и менее), то наиболее простым и быстрым решением проблемы является использование файла/раздела подкачки. В данной инструкции речь пойдёт о файле, так как это наиболее хорошее решение для сборки системы: размер файла очень быстро регулируется, быстро удаляется.

В целом, при компиляции базовой системы было занято примерно 3,5-4 Гб памяти.

Из этого и рассчитывайте размер swap.

Чтобы узнать, существует ли уже подкачка или нет, выполните:

```
1  swapon --show
```

А чтобы просмотреть использование ОЗУ и Swap:

```
1  free -m
```

13.6.1 Рассчёт размера подкачки (swap)

Обычно объём подкачки равен половине объёма ОЗУ/объёму ОЗУ, умноженному на 2, но не всегда этого может хватить, особенно на слабых ПК. Поэтому рассчитайте размер файла или раздела так, чтобы обеспечить минимум 4 Гб в общей сложности (ОЗУ+Swap). Для сборки базовой системы этого хватит, а для сборки таких программ, как, например, Pale Moon, этого не хватит - нужно мощное железо. Не проще ли, в таком случае, не компилировать такое "тяжёлое" ПО, а найти бинарные пакеты?

13.6.2 Создание файла подкачки

Чтобы создать файл подкачки, выполните:

```
1  sudo fallocate -l 1G /swapfile &&
2  sudo chmod 600 /swapfile &&
3  sudo mkswap /swapfile &&
4  sudo swapon /swapfile
```

Значения новых команд

- `sudo fallocate -l 1G /swapfile` - создать файл `/swapfile`, размером 1 Гб. Чтобы выбрать другой размер, замените "1G" на желаемое значение.
- `sudo chmod 600 /swapfile` - в целях безопасности, выставить нужные права на файл. О правах на файлы читать в интернете.
- `sudo mkswap /swapfile` - создать файловую систему `swap`.
- `sudo swapon /swapfile` - включение подкачки.

13.6.3 Настройка vm.swappiness

Теперь настройка свопа. Есть параметр, сообщающий ядру, как часто использовать подкачку.

Для того чтобы проверить, какой параметр используется, выполните:

```
1  cat /proc/sys/vm/swappiness
```

Если хотите изменить это значение, выполните:

```
1  sudo sysctl vm.swappiness=X
```

Либо же:

```
1  sudo vim /etc/sysctl.conf
```

```
1  vm.swappiness=X
```

Где `X` - нужное значение. `Swappiness` может иметь значение от 0 до 100, значение по умолчанию = 60. Низкое значение заставляет ядро избегать подкачки, высокое значение позволяет ядру использовать подкачку активнее. Использование низкого значения на достаточном количестве памяти улучшает отзывчивость системы. Ну и жёсткий диск будет использоваться не так часто.

Сохранение изменений после перезагрузки

Как только система перезагрузится, придётся опять включать подкачку и выставлять `vm.swappiness`, что, конечно, неудобно. Чтобы этого избежать, нужно сделать соответствующую запись в `/etc/fstab`:

```
1  echo '/swapfile none swap sw 0 0' |sudo tee -a /etc/fstab
```

И записать в `/etc/sysctl.conf` нужное значение `swappiness`:

```
1  echo 'vm.swappiness=X' |sudo tee -a /etc/sysctl.conf
```

Где `X` - нужное значение.

13.6.4 Удаление файла подкачки

После сборки и настройки системы, вероятно, `swap` вам больше не понадобится. Поэтому лучше его удалить. Напоминаю, что все действия из этой инструкции выполняются ТОЛЬКО на хост-системе.

Выполните:

```
1  sudo swapoff /swapfile
2  sudo rm /swapfile
```

И удалите записи в `/etc/fstab` и `/etc/sysctl.conf`.

Значение новых команд

- `sudo swapoff...` - отключить подкачку
- `sudo rm /swapfile` - удалить подкачку

ВАЖНО!!!

Danger

Если полностью отключить подкачку, то ОС будет использовать только ОЗУ и в случае его нехватки система может просто зависнуть.

Ну и пару нужных вещей:

- Подкачка нужна для ровного и эффективного высвобождения оперативной памяти, и использовать `swap` в качестве "экстренной памяти" не рекомендуется в принципе;
- Отключение `swap` не спасает от проблемы дискового ввода/вывода при конкуренции за память - дисковый I/O перемещается с анонимных страниц на файловые, что не только может быть менее эффективным, поскольку остаётся меньший пул страниц, доступных для высвобождения, но и само по себе может способствовать появлению этой высокой конкуренции.

Также вместо `swap` можно использовать `zram/zswap`.

Больше о подкачке смотреть [здесь](#).

О работе с `Zram` смотреть [здесь](#).

13.7 Настройка zram

Если оперативной памяти вашего компьютера недостаточно (3 Гб и меньше), то наиболее быстрым способом решения проблемы является использование подкачки, так как компиляция некоторых пакетов требует мощного ПК, в том числе, большого объёма ОЗУ. Для этих целей можно использовать файл или раздел подкачки, но можно и zram.

Note

Действия производятся на хост-системе

Для начала нужно загрузить модуль. В системе его может не оказаться, поэтому можно его скомпилировать. Такие дистрибутивы как Ubuntu и прочие предоставляют пакет `zram-config`, в Fedora и Calculate Linux zram активирован по умолчанию.

Выполните:

```
1 modprobe zram num_devices=4
```

В `num_devices` задаётся количество сжатых блочных устройств, которое будет создано. Для наиболее оптимального использования CPU стоит учесть: сжатие каждого устройства `zram` однопоточное. Потому создавайте их по количеству ядер. При настройке модуля задается фиксированный размер НЕ сжатых данных в байтах.

```
1 SIZE=1536
2 echo $((SIZE*1024*1024)) > /sys/block/zram0/disksize
3 echo $((SIZE*1024*1024)) > /sys/block/zram1/disksize
4 echo $((SIZE*1024*1024)) > /sys/block/zram2/disksize
5 echo $((SIZE*1024*1024)) > /sys/block/zram3/disksize
```

В итоге будет создано устройство `/dev/zram0` заданного размера - замените значение 1536 переменной `SIZE` на необходимое вам. Как было описано в предыдущей инструкции о выборе размера подкачки, в среднем, при компиляции ПО используется около 3-4 Гб ОЗУ. Из этого и рассчитывайте размер сжатого блочного устройства `zram`. В большинстве популярных дистрибутивов Linux уже настроен `zram`. Но, как правило, его объём равен половине объёма ОЗУ, что на слабых ПК может быть недостаточным.

```
1 Disk /dev/zram0: 1610 MB, 1610612736 bytes, 393216 sectors
2   Units = sectors of 1 * 4096 = 4096 bytes
3   Sector size (logical/physical): 4096 bytes / 4096 bytes
4   I/O size (minimum/optimal): 4096 bytes / 4096 bytes
```

Данные, записанные на него, будут прозрачно сжаты в памяти. Что делать с ним далее — уже ваш выбор, я на этих устройствах создаю разделы подкачки.

```
1 mkswap /dev/zram0
2 mkswap /dev/zram1
3 mkswap /dev/zram2
4 mkswap /dev/zram3
5
6 swapon /dev/zram0 -p 10
7 swapon /dev/zram1 -p 10
8 swapon /dev/zram2 -p 10
9 swapon /dev/zram3 -p 10
```

Для того чтобы проверить результат, выполните:

```
1 swapon -s
```

Далее уже ядро рассчитывает само, какие данные туда перемещать в зависимости от того, как часто вы к ним обращаетесь и как много памяти свободно.

В любом случае, для регулирования отправления данных в swap, настройте `vm.swappiness`, который предоставляет очень быструю и простую конфигурацию. О настройке читайте [здесь](#).

Так как с помощью zram подкачка находится в ОЗУ, т.е. данные в нём просто сжимаются, то такая подкачка (swap) даже быстрее обычного файла или раздела на жёстком диске.

[Источник.](#)

13.8 О шебангах в скриптах Linux

Все скрипты, которые создаёт пользователь, исполняемыми не являются. Для того чтобы дать право исполнения определённому файлу, нужно выполнить:

```
1 chmod +x $FILE
```

Где `$FILE` - имя файла

Конечно, скрипт можно запустить и без права исполнения, для этого нужно запустить его в определённом интерпретаторе (bash, perl, python, et cetera):

```
1 bash $FILE
```

Или

```
1 python $FILE
```

Но это не очень удобно, да и замедляет пользователя. А как быть, если скрипт нужно поместить, скажем, в `/usr/bin` или в любом другом каталоге из переменной `PATH`? Например, пакетный менеджер `slackpkg` из состава Slackware Linux или `urpmi` из Mandriva/Rosa/Mageia - те же самые скрипты на Bash и Perl соответственно.

Поэтому, лучшим решением будет вставка определённой строки с указанием нужного интерпретатора. Эта строка называется *sha-bang* (*шебанг*). И потом сделать скрипт исполняемым.

Например, для Bash-скриптов шебанг выглядит так:

```
1 #!/bin/bash
```

Или так:

```
1 #!/bin/env bash
```

`env` - UNIX-утилита, позволяющая модифицировать список переменных окружения перед исполнением пользовательской команды с изменением окружения.

Но также эту команду используют и для улучшения переносимости скриптов, так как в разных дистрибутивах нужные интерпретаторы могут находиться в разных местах, а вот путь к `env` одинаковый везде.

Например, в одних дистрибутивах путь к bash `/bin/bash`, а в других: `/usr/bin/bash`. И при компиляции этого интерпретатора он ставится именно в `/usr`, а уже позже его можно перенести в `/bin` (что и реализовано в этом руководстве, поэтому в вашем дистрибутиве путь классический: `/bin/bash`).

А, например, во FreeBSD bash находится по пути `/usr/local/bin/bash`, поэтому использование `env` будет таким:

```
1 #!/usr/bin/env bash
```

Но тут ещё одна загвоздка: иногда различаются ещё и имена интерпретаторов. Например, Python3 в системе Windows (через MinGW, например) называется `python`, а не `python3`, как, например, в Linux.

Поэтому, перед созданием скриптов (или при переносе таковых из Linux for yourself в другие системы) будьте внимательны в правильности шебанга. А лучше, проверьте его или тестовым скриптом (самый простейший "Hello, world!"), либо же воспользуйтесь утилитой `which` (или её аналогом), для того чтобы узнать нахождение нужного интерпретатора.