Министерство науки и высшего образования
Российской Федерации
КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ОСНОВЫ РАБОТЫ В LINUX

Учебное пособие

Краснодар
2025

УДК 004.451(075.8)
ББК 32.972.111я73
О 753

Рецензенты:
Доктор технических наук, профессор,
Донецкий национальный технический университет
В.Н. Павлыш
Кандидат технических наук, доцент,
Вятский государственный университет
В.Б. Чернявский

О 753 Основы работы в Linux: учебное пособие / авторы: Е. Р. Алексеев, Ю. С. Коваленко, К. В. Дога, Ю. Н. Обухова; Министерство науки и высшего образования Российской Федерации, Кубанский государственный университет. – Краснодар: Кубанский гос. ун-т, 2025. – 88 с. – 500 экз.

ISBN 978-5-8209-2625-9

Содержит основные сведения, необходимые для начального освоения операционной системы семейства Linux. Авторы знакомят читателя с базовыми понятиями операционных систем, построенных на ядре Linux. В издании описаны основные этапы установки ОС семейства Linux на компьютер, а также приведены особенности настройки российских дистрибутивов «Альт» после установки. Пособие содержит тексты одиннадцати лабораторных работ для практического освоения дисциплины.

Адресуется студентам математических, физико-технических, ИТ-специальностей.

УДК 004.451(075.8)
ББК 32.972.111я73

ISBN 978-5-8209-2625-9

ВВЕДЕНИЕ

Цель пособия – познакомить читателя с фундаментальными понятиями в области операционных систем (ОС), построенных на базе ядра *Linux*. Эти операционные системы считаются альтернативой проприетарной ОС, доминирующей на рынке персональных компьютеров.

В пособии нет привязки к конкретному дистрибутиву и подробного описания графического интерфейса. Большое внимание уделяется базовым навыкам, которыми должен овладеть пользователь для дальнейшего самостоятельного изучения операционных систем. В доступной форме разъясняются такие понятия, как операционная система, дистрибутив, права доступа к файлу, жёсткая ссылка и т. д.

Читатель познакомится с базовыми понятиями операционных систем, построенных на базе ядра *Linux*. Для отработки практических навыков приводятся тексты одиннадцати лабораторных работ.

В первой главе описаны две модели разработки программного обеспечения: свободная и проприетарная. Особое внимание уделено принципам свободного программного обеспечения. Вторая глава содержит информацию об операционных системах, дистрибутивах. В последующих нескольких главах (3–6) раскрыты такие важные для освоения *Linux* на уровне продвинутого пользователя понятия, как права доступа к файлам, дерево каталогов, роль администратора в системе, основные команды терминала, необходимые для управления файловой системой компьютера, работающего под управлением unix-подобных ОС. В седьмой описаны жёсткие и символьные ссылки. Отдельная глава посвящена особенностям работы в операционных системах семейства «Альт». Их настройка отличается и от дистрибутивов на базе *Debian*, и от дистрибутивов на базе rpm-пакетов. В девятой главе излагаются общие принципы установки операционных систем семейства *Linux* на компьютер пользователя. Десятая глава содержит задания лабораторного практикума, который выполняют студенты для практического освоения операционных систем семейства *Linux*.

Лабораторные работы, приведенные в пособии, разрабатывались и апробировались более десяти лет на базе Донецкого национального технического университета (г. Донецк, ДНР), Вятского государственного университета (г. Киров), Кубанского государственного университета (г. Краснодар).

1. СВОБОДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Современное программное обеспечение (ПО) по методам разработки и идеологии использования можно разделить на два больших класса: *свободное* и *проприетарное*.

Программное обеспечение называется **свободным,** если оно удовлетворяет следующим четырём условиям свободы.

Свобода 0. Вы можете запускать свободное программное обеспечение на своём (и не только на своём) компьютере. Возможность запускать программу не зависит от того, как вы её получили.

Свобода 1. Вы можете распространять свободное программное обеспечение (делиться программами с друзьями, коллегами, знакомыми).

Свобода 2. Вы можете изменять свободное программное обеспечение.

Свобода 3. Вы можете распространять модифицированное вами свободное программное обеспечение.

Чтобы изменять свободное программное обеспечение, должен быть доступен исходный код программ. Доступность исходного кода – самое важное свойство свободного программного обеспечения.

Обратите внимание, критерии свободы программного обеспечения никаким образом не затрагивают вопросы стоимости компьютерных программ или авторских прав. Зачастую свободные программы являются бесплатными, но это не обязательное требование.

Программное обеспечение, которое не удовлетворяет всем четырём свободам, называется **проприетарным**.

Свобода программного обеспечения открывает новые возможности для его разработки, модификации и тестирования.

Можно выделить следующие известные свободные программы: ОС семейства *Linux*, *Mozilla Firefox*, *Python 3*, *LibreOffice*, *Scilab, GNU Octave, Maxima*.

2. ЗНАКОМСТВО С ОПЕРАЦИОННОЙ СИСТЕМОЙ LINUX

Операционная система (ОС) – комплекс взаимосвязанных программ, предназначенных для работы с вычислительным устройством и организации взаимодействия с пользователем.

В основе понятия ОС лежит понятие ядра. **Ядро** – это низкоуровневая программа, которая управляет физическими ресурсами вычислительного устройства и другими программами.

В отличие от *Windows*, *Linux* – это не конкретная операционная система, которую можно установить на устройство. *Linux* – это ядро операционной системы, на базе которого создаются различные дистрибутивы.

Дистрибутив Linux – это система комплектации ядра ОС с комплексом утилит и прикладных программ.

Современные дистрибутивы:

– *Debian, Ubuntu, Mint;*

– *RedHat, Fedora, OpenSUSE, CentOS;*

– *Arch, Manjaro, Gentoo;*

– русские дистрибутивы *Linux* (*ROSA*, *Calculate*, *Astra*, *ALT Linux*).

Кроме разделения дистрибутивов по производителям, существует деление по типам использования:

1. *LiveCD* – дистрибутив, при использовании которого загрузка операционной системы (рабочего стола) происходит с CD (или USB). После этого с операционной системой можно работать, как с ОС, запущенной с жесткого диска.

2. *Установочные дистрибутивы* – дистрибутивы, с помощью которых осуществляется установка ОС на жесткий диск.

Термин «дистрибутив» применим не только к операционным системам, но и к прикладному программному обеспечению (ПО).

Дистрибутив – это набор файлов, который включает вспомогательные инструменты для автоматической или автоматизированной начальной настройки программного обеспечения.

При использовании дистрибутива все необходимые файлы устанавливаются таким образом, чтобы их правильно видела

операционная система. Кроме того, конфигурируются начальные параметры, язык, способ подключения к Интернету и т. д.

В ОС *Linux* дистрибутивы программ называются пакетами, которые бывают нескольких видов. Самые популярные из них:

1) *deb* (в *Debian*, *Ubuntu*, *Linux Mint* и др.);

2) *rpm* (*OpenSUSE*, *CentOS*, *Mageia, Alt Linux* и др.).

Desktop enviroment (DE) – это окружение рабочего стола, в которое входят:

– графическая подсистема;

– оконный менеджер;

– файловый менеджер;

– эмулятор терминала;

– меню запуска приложений;

– браузер и т. д.

В *Windows* существует всего одно окружение рабочего стола. Оно стандартно, пользователь видит его сразу после установки ОС и изменить не может. Однако один и тот же дистрибутив *Linux* может иметь несколько вариаций с разным графическим окружением, например:

1. *KDE (Plasma)*.
2. *GNOME (MATE, CINAMON)*.
3. *XFCE*.
4. *LXDE*.
5. *LXQT*.
6. *Openbox*.

Сейчас есть различные дистрибутивы *Linux*, между которыми много общего. Чаще всего поправки (в основном графические) вносятся в инсталляцию, изменяется список включаемого ПО. При этом ядро, как правило, остается без изменений. В качестве графической оболочки почти везде используется *KDE* или *GNOME*. В случае отсутствия их всегда можно установить, и тогда вне зависимости от основного дистрибутива у всех будет одинаковый графический интерфейс.

3. ФАЙЛОВАЯ СИСТЕМА ОС LINUX

Основные понятия в файловой системе ОС *Linux* – файл и каталог.

Файл – поименованная область данных, расположенная на внешнем носителе. Исходя из определения, файл характеризуется содержимым, именем и месторасположением.

Имя файла может иметь расширение. Большинство графических интерфейсов связывают программу обработки файла данного типа с его расширением.

Каталог – поименованный перечень файлов и подкаталогов (вложенных каталогов). Он позволяет облегчить поиск данных и систематизировать их на устройстве. В большинстве современных файловых систем используется иерархическая структура. В любой файловой системе *Linux* существует единственный общий для всех файлов каталог – корневой. Обозначается / (слеш).

В нём могут содержаться не только файлы, но и подкаталоги первого уровня вложенности. В каталогах первого уровня вложенности тоже могут существовать файлы и подкаталоги (по отношению к корневому это будут каталоги второго уровня вложенности) и т. д. Эта система называется деревом каталогов.

Положение любого файла в этой системе описывается при помощи полного пути, который начинается от / (корневого каталога), а затем через разделитель / перечисляются все подкаталоги согласно порядку вложенности. Заканчивается полный путь названием искомого файла. Например, `/home/user/pas/abc.pas` – файл *abc.pas*[1], который находится в каталоге `/home/user/pas`[2].

Древо устроено так, что среди подкаталогов конкретного каталога не может находиться он сам. Придумано это для того, чтобы избежать зацикливания. Благодаря такому принципу путь к файлу или каталогу определяется однозначно.

Текущий каталог – это каталог, с которым пользователь (или программа) работает в настоящий момент времени. С текущего

---

[1] *.pas* – расширение файлов программ на языке Pascal.

[2] `/home/user/pas/abc.pas` – путь к файлу, его полное имя.

каталога начинается относительный путь к файлу, и в этом существенное его отличие от полного пути.

Кроме того, в *Linux* у каждого пользователя обязательно есть свой каталог, который становится текущим сразу после регистрации в системе. Он называется домашним и предназначен для хранения пользовательских данных.

Родительский каталог – такой каталог, в котором содержится данный. Для корневого каталога родительским является он сам.

Обязательные подкаталоги корневого (см. рисунок):

`/bin` и **`/sbin`** – каталог для хранения исполняемых файлов системных программ;

`/boot` («*boot*» – загрузка системы) – файлы, необходимые для самого первого этапа загрузки: загрузки ядра и само ядро;

`/etc` – каталог для конфигурационных файлов;

`/dev` – каталог для файлов внешних устройств;

`/lib` (*libraries* – библиотеки) – каталог для хранения разделяемых библиотек;

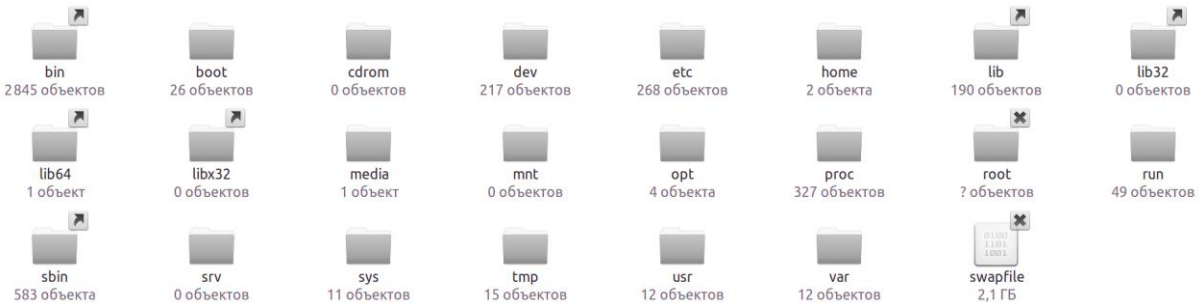
`/var` и **`/tmp`** – каталоги для всякого рода регулярно изменяемых или временных данных;

`/usr` – каталог для пользовательских программ со всеми их файлами, например с документацией;

`/home` – место пользовательских каталогов с данными;

`/mnt` (**`/media`**) – каталог для монтирования (от англ. mount) для временного подключения файловых систем;

`/root` – домашний каталог администратора системы – пользователя *root*.



Содержимое корневого каталога

4. ТЕРМИНАЛ LINUX И ОСНОВНЫЕ КОМАНДЫ

Некоторые действия в *Linux* могут осуществляться только в режиме терминала, иногда работать в нем проще, чем в графическом режиме.

Терминал – это программа, которая используется для взаимодействия пользователя с операционной системой.

Терминал необходим для решения административных задач, таких как установка пакетов, управление пользователями, действия с файлами и т. д.

Запуск осуществляется командой *Системные – Терминал*[1]. После чего в окне терминала можно вводить команды вида

```
команда параметры
```

Однако ряд действий можно осуществить только с правами администратора, для чего вводится команда:

```
sudo команда параметры
```

После этого система спросит пароль администратора.

Текстовый редактор nano

Для редактирования небольших текстов можно использовать текстовый редактор *nano*, который входит в набор программ GNU и является стандартным консольным редактором для многих дистрибутивов *Linux*. Справка по возможностям *nano* доступна после его запуска с помощью комбинации клавиш *Ctrl+G* (F1). Рассмотрим некоторые комбинации клавиш в *nano*:

^G – справка;

^O – сохранить данные в файле;

^X – закрыть nano;

^W – поиск текста;

^ – замена текста;

^6 – пометка начала и конца выделяемого фрагмента текста, для выделения текста можно использовать клавиши управления курсором;

[1] Во многих дистрибутивах Linux терминал можно вызвать комбинацией клавиш *Ctrl+Alt+T*

^K – вырезать текст в буфер;

M^ – копировать текст в буфер;

^U – вставить текст из буфера;

M U – отмена последнего действия;

^Y – переход к первой строке;

^V – переход к последней строке;

^R – прочитать файл.

Здесь **^** – клавиша *Ctrl*, **M** – клавиша *Alt*. Более подробное описание возможностей редактора *nano* можно найти на следующих сайтах:

– https://losst.ru/tekstovyj-redaktor-nano-v-linux-dlya-ovichkov;

– https://darksf.ru/2018/07/25/rukovodstvo-po-ispolzovaniyu-teksto.

Рассмотрим основные команды Linux, которые могут понадобиться рядовому пользователю.

Команды получения справочной информации

ОС *Linux* позволяет получить справочную информацию по самой системе. Современные дистрибутивы содержат тысячи страниц документации, представленной в электронном виде. Выводить эту информацию можно с помощью страниц интерактивного руководства `man`.

Команда man

Команда `man` имеет вид

man name

Здесь `name` – название команды или другого объекта, о котором нужна информация. Например, команда

```
man pwd
```

выведет на экран информацию о команде `pwd`. Для управления процессом вывода используются следующие клавиши:

Q – выход из программы;

Enter – просмотр строка за строкой;

Space – вывод следующего экрана информации;

B – вернуться к предыдущему экрану;

/, за которым следует строка символов, и **Enter** – поиск введенной строки символов;

N – повторение предыдущего поиска.

Если необходимо отпечатать копию страницы, то можно воспользоваться командой

```
man name | lpr
```

для postscript-принтера

```
man -t name | lpr
```

В русскоязычных дистрибутивах часть справочных страниц переведена на русский язык, однако в основном справочная информация англоязычная.

Команда info

Команда `info` является альтернативой команде `man`. Она имеет вид

info name

здесь *name* – имя команды.

Например,

```
info pwd
```

Информация, которую вы увидите, в большинстве случаев несколько отличается от той, которую дает команда `man`. Но самое существенное отличие заключается в том, что выдаваемая `info` информация представлена в гипертекстовом формате. В силу этого появляется возможность просматривать различные разделы помощи. Работая в тестовом режиме, вы можете запустить `info` в одной из альтернативных консолей. Переход от графического режима к текстовым альтернативным консолям осуществляется комбинациями клавиш: *Ctrl+Alt+F2, Ctrl+Alt+F3,* возврат в графический – *Ctrl+Alt+F7.*

Команда help

Если ввести в командной строке help без параметров, то будет выведен список всех встроенных команд. Если ввести команду

help name

где *name* – имя одной из этих команд, то выведется очень краткая справка о применении этой команды.

Команды для работы с файлами

Для работы с файлами и каталогами чаще всего применяются следующие команды:

`touch file` – создание пустого текстового файла с именем *file*;

`cat file` – просмотр файла с именем *file*;

`less file` – используется для просмотра файла с возможностью скроллинга;

`tac file` – просмотр текстового файла с именем *file* в обратном порядке, от последней строки к первой;

`cp file1 file2` – копирование *file1* в *file2*;

`mv file1 file2` – перемещение *file1* в *file2*;

`rm file` – удаление файла с именем *file*;

`rm –r dir` – рекурсивное удаление каталога *dir* (вместе с содержимым);

`rmdir dir` – удаление пустого каталога *dir*;

`locate file` – быстрый поиск файла;

`which file` – вывод каталога, в котором находится программа, если она установлена;

`mkdir dir` – в текущем каталоге создает каталог с именем *dir*;

`cd dir` – переход в каталог с именем *dir*;

`ls dir` – вывод содержимого каталога *dir*.

Вместо имен каталогов можно использовать:

`.` – текущий каталог;

`..` – родительский каталог (на уровень выше);

`~` – домашний каталог пользователя.

Рассмотрим примеры нескольких связанных между собой команд работы с файлами и каталогами.

`touch ./my.txt` – в текущем каталоге создаем пустой текстовый файл *my.txt*;

`mkdir dir1` – в текущем каталоге создаем каталог *dir1*;

`nano ./my.txt` – вызываем текстовый редактор *nano* и редактируем в нем файл *my.txt*;

`cp ./my.txt ./dir1/my2.txt` – копируем файл *my.txt* из текущего каталога в созданный в нем каталог *dir1* под именем *my2.txt*;

`cp ./my.txt ./dir1/my3.txt` – копируем файл *my.txt* из текущего каталога в подкаталог *dir1* под именем *my3.txt*.

`sudo mv ./dir1/my3.txt /tmp/3.txt` – переносим файл *./dir1/my3.txt* в каталог */tmp* под именем *3.txt* (для выполнения команды нужны права суперпользователя);

`rm ./my.txt` – удаляем файл *my.txt* из текущего каталога;

`cd ./dir1` – переходим в каталог *./dir1*, который теперь будет текущим;

`cp ./my2.txt ./my4.txt` – копируем файл *./my2.txt* из текущего каталога (теперь это каталог *dir1*) под именем *my4.txt* в этот же каталог;

`ls .` – вывод на экран содержимого текущего каталога;

`cd ..` – переход на уровень выше;

`rm -r dir1` – удаляем каталог *dir1* вместе с его содержимым.

Поиск файлов. Команда find

Команда `find` имеет вид

`find путь -опции`

Здесь путь – полный путь к каталогу, в котором следует произвести поиск. Опции могут быть следующие:

`-name` (поиск файлов по имени);

`-user` (поиск файлов, принадлежащих указанному пользователю);

`-group` (поиск файлов, принадлежащих указанной группе);

`-perm` (поиск файлов с указанным режимом доступа);

`-type` (поиск файлов определенного типа);

`-size ±n` (поиск файлов с размером `n` единиц, где `b` – блоки 512 байт; `c` – байт; `k` – килобайт; `M` – мегабайт; `G` – гигабайт);

`-mtime ±n` (поиск файлов, содержание которых изменялось менее чем «-» или более чем «+» дней назад).

Рассмотрим примеры.

`find / -name config -print` – искать файлы с именем *config* (`-name config`), начиная с каталога / (первый параметр команды `find`), и выводить имена файлов на экран с помощью аргумента `-print` (поиск рекурсивный);

`find . -type f -name "d*" -print` – найти в текущем каталоге обычные файлы (не каталоги), имя которых начинается с символа *d*;

`find . -newer pr1.cpp -type f -print` – найти в текущем каталоге файлы, измененные позже, чем файл *pr1.cpp*;

`find . -type f -empty` – найти в текущем каталоге пустые файлы.

Перенаправление информации в файл

Основные операторы перенаправления представлены в таблице.

Основные операторы перенаправления

Оператор	Назначение
>	Данные на выходе команды выводятся не в консоль, а в файл или на другое устройство (принтер).
>>	Выходные данные дописываются в конец файла, при этом сохраняется существующая информация.
<	Считывание данных не с клавиатуры, а из файла.
|	Выходные данные одной команды считываются и записываются, как входные данные другой команды.

Пример.

```
cat > file1
Hello world!
```

(для завершения ввода текста нажмите сочетание клавиш Ctrl+D)

В *file1*, который автоматически был создан в домашнем каталоге пользователя, записана фраза «*Hello world!*».

```
cat file1
```

В терминале отобразится строка из файла *file1* «*Hello world!*».

```
cat > file2
some other text
```

В *file2*, который также автоматически был создан в домашнем каталоге пользователя, записана фраза «*some other text*» **.**

```
cat< file1 | cat>> file2
```

Содержимое *file1* было считано и перенаправлено на ввод в *file2*, при этом данные дописываются в конец файла, сохраняя записанную ранее строку «*some other text*».

```
cat file2
```

На экран выводится содержимое *file2*, а именно строка «*some other text*», и «*Hello world!*», которая была скопирована из *file1*.

При перенаправлении информации несколько файлов можно объединить в один.

Запишем в *file1* текст «*Строка1*», в *file2* – «*Строка2*», в *file3* – «*Строка3*». Затем содержимое *file1* и *file2* допишем в конец файла *file3*.

```
cat > file1
Строка1
cat > file2
Строка2
cat > file3
Строка3
cat file1 file2 >> file3
cat file3
```

В результате получим три надписи в следующем порядке:

```
Строка3
Строка1
Строка2
```

5. СОЗДАНИЕ НОВОГО ПОЛЬЗОВАТЕЛЯ. ИЗМЕНЕНИЕ ПРАВ ДОСТУПА

Команды управления пользователями

Для создания нового пользователя можно воспользоваться одной из следующих команд:

sudo adduser nameuser

sudo useradd nameuser

useradd – команда, которая создает пользователя, не имеющего домашнего каталога (если не указан ключ m) и каких-либо дополнительных настроек;

adduser – более безопасная команда, которая создает пользователя вместе с его домашним каталогом и другими пользовательскими настройками.

Изменить или задать пароль пользователю *nameuser* можно с помощью команды

sudo passwd nameuser

Изменение параметров учетной записи пользователя *nameuser* осуществляется командой

sudo usermod nameuser

Параметры команды:

-d, --home homedir – новый домашний каталог учётной записи;

-e, --expiredate date – установить дату окончания действия учётной записи в *date*;

-f, --inactive inactivity – установить период неактивности пароля после устаревания учётной записи равным *inactivity*;

-h, --help – показать данное сообщение и закончить работу;

-l, -login newname – новое значение имени учётной записи;

-L, --lock – заблокировать учётную запись;

-m, --move-home – переместить содержимое домашнего каталога в новое место (использовать только вместе с `-d`);

-o, --non-unique – разрешить создание учётной записи с уже имеющимся UID;

-p, --password pass – задать новый шифрованный пароль для учётной записи.

Сменить пользователя можно несколькими способами.

1. После загрузки системы на экране появляется список пользователей. Требуется выбрать того пользователя, под которым нужно войти в систему, и ввести пароль.

2. После нажатия стандартного сочетания клавиш *Ctrl+Alt+L* блокируется экран устройства. При попытке входа в систему на экране появляется кнопка «переключить пользователя». Требуется выбрать того пользователя, под которым нужно войти в систему, и ввести пароль.

3. В терминале «смена пользователя» осуществляется командой **su – nameuser** (где **nameuser** – это имя пользователя).

Права доступа в ОС Linux

В Windows можно определить права доступа для каталога, и они автоматически распространяются на все файлы и подкаталоги. В ОС *Linux* права не наследуются, они у каждого файла свои. Чтобы увидеть права доступа к файлам текущего каталога, достаточно набрать в терминале **ls -l**. В первом столбце выведены права доступа. Первый символ – это тип файла, остальные девять – права. Результат работы команды преведен ниже.

```
$ ls -l
-rw-r--r-- 1 evg evg 212 2008-09-12 19:38 1.tex~
-rw-r--r-- 1 evg evg 195 2008-09-12 19:42 2~
-rw-r--r-- 1 evg evg 223 2008-09-12 20:00 2.tex~
drwxr-xr-x 2 evg evg 4096 2008-09-14 13:03 Example
-rw-r--r-- 1 evg evg 1647594 2008-09-12 22:59 Ex.zip
-rw-r--r-- 1 evg evg 1585850 2008-09-13 19:30 ll.pdf
-rw-r--r-- 1 evg evg 6229739 2008-09-13 19:33 lv.pdf
-rw-r--r-- 1 evg evg 878 2008-09-12 19:09 nf.lyx~
drwxr-xr-x 3 evg evg 4096 2008-09-14 12:40 primer
drwxr-xr-x 2 evg evg 4096 2008-09-14 13:03 proba
drwxr-xr-x 2 evg evg 4096 2008-09-10 13:42 Regul
-rw------- 1 evg evg 342861 2008-09-12 23:45 Reg.zip
```

Типы файлов:

b – специальный блочный файл;

d – каталог;

c – специальный символьный файл;

f – обычный файл, который может обозначаться через тире;

l – символическая ссылка;

p – именованный канал;

s – сокет.

Права доступа делятся на три группы:

1) *user* – права пользователя;

2) *group* – права группы, к которой принадлежит пользователь;

3) *other* – права всех остальных пользователей системы.

```
r w x | r w x | r w x
 user | group | other
```

Права доступа к файлам и каталогам имеют различный смысл. Для файлов:

r – право на чтение данных из файла;

w – право на изменение содержимого файла (запись);

x – право на исполнение файла.

Права доступа к каталогу интерпретируются по-другому:

r – право на чтение (просмотр) содержимого каталога (команда терминала `ls`);

w – право на изменение содержимого каталога – создание и удаление объектов в этом каталоге;

x – право на «вход» в каталог (команда терминала `cd`).

Для упрощения записи у каждого типа доступа есть числовое представление, используется двоичная система счисления. Единица – есть право, 0 – нет права. Так, запись прав `rwx r-x r-x` в бинарном виде будет выглядеть следующим образом: 111 101 101. Но двоичное представление не очень удобно, поэтому используют восьмеричное. 111 в двоичной системе – это 7 в

восьмеричной, а 101 – это 5, таким образом 111 101 101 – это 755. Итак, получаем:

```
 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111
 --- | --x | -w- | -wx | r-- | r-x | rw- | rwx
  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7
```

Основные команды для работы с правами доступа

Команда **chgrp** позволяет изменить группу владельца файла. Если эту команду использует простой пользователь, то он, во-первых, должен быть владельцем файла, а во-вторых, он должен быть членом группы, которой он хочет дать права. Пользователь не должен быть членом группы, которая владеет файлом. Синтаксис следующий:

```
chgrp newgroup files
```

можно использовать как имя, так и ID группы (GID).

```
$ ls -l
-rw-r-r- 1 adm root 0 2007-09-03 15:33 file.txt
$ chgrp users file.txt
$ ls -l
-rw-r-r- 1 adm users 0 2007-09-03 15:33 file.txt
```

Команду можно использовать с ключом **-R**, что приведет к рекурсивному обходу всех файлов в каталоге и его подкаталогах.

Команда **chown** используется для изменения как владельца, так и группы. В большинстве систем только суперпользователь имеет право пользоваться этой командой. У команды следующий синтаксис:

```
chown newuser : newgroup files
```

Как и в команде `chgrp`, вы можете использовать имена пользователей и групп, а также их идентификаторы GID и UID. Команда не проверяет пользователей и группы на существование, так что можно задать и несуществующие.

```
$ ls -l
-rw-r-r- 1 adam users 0 2007-09-03 15:33 file.txt
root@laptop:# chown root:root file.txt
adam@laptop:~$ ls -l
-rw-r-r- 1 root root 0 2007-09-03 15:33 file.txt
```

Например,

`chown student file.txt` – смена владельца файла *file.txt* на пользователя `student`, группа остается без изменений.

`chown : university file.txt` – смена группы файла *file.txt* на `university`, владелец без изменений.

Рассмотрим команду `chmod` для смены прав доступа:

```
chmod access rights files
```

Перед выполнением команды необходимо выбрать класс пользователей, для кого мы хотим изменить права: `u`, `g` или `o` (`User, Group, Others`). Есть также дополнительная группа `a` (`all`, все). Затем выбираем оператор: + (дать права), - (убрать права) и = (присвоить права), в конце – сами права (`rwx`, `r-x` и т.д.).

Например,

`chmod u+x skrypt.sh` – пользователю (`u`) добавляется (+) право на выполнение (`x`) файла `skrypt.sh`;

`chmod go-r raport.odt` – у группы и остальных (`go`) отбирается (-) право на чтение (`r`);

`chmod a=w finanse.ods` – всем (`a`) присваивается (=) право на запись (`w`), а остальные права стираются (=);

`chmod u+rwx , g+rwx , o+x trurl.py` – пользователю прибавляются права на все, группе тоже, а остальным прибавляется право на выполнение.

Существует и второй синтаксис команды **`chmod`**:

```
chmod –R ugo file
```

Здесь:

`u` – цифра в восьмеричной системе счисления, определяющая права доступа для владельца файла;

`g` – цифра в восьмеричной системе счисления, определяющая права доступа для группы, к которой принадлежит владелец файла;

`o` – цифра в восьмеричной системе счисления, определяющая права доступа для всех остальных пользователей;

`file` – имя файла или каталога;

`-R` – параметр, позволяющий выполнить команду рекурсивно, если речь идёт об изменении прав каталога (под

рекурсией тут понимается распространение действия команды на все файлы и каталоги, которые находятся внутри указанного в команде); без этого ключа права меняются только для указанного каталога.

Пример.

```
chmod 710 . /progr
```

После выполнения этой команды:

– владелец файла `progr` может просматривать, редактировать и исполнять файл `progr`, который находится в текущем каталоге (`7 – 111`);

– члены группы могут только запускать файл `progr` на выполнение (`1 – 001`);

– у всех остальных пользователей нет никаких прав для работы с этим файлом `0 – 000`.

```
chmod 744 . /abc –R
```

Каталогу `abc` из текущего каталога пользователя и всем файлам и каталогам, которые находятся внутри `abc` установить следующие права:

– у владельца каталога есть права чтения, модификации и запуска (`7 – 111`);

– у членов группы, которой принадлежит владелец каталога, есть право чтения (`4 – 100`);

– у всех остальных есть право чтения (`4 – 100`).

Рассмотрим, какая информация хранится в системе о файле. Для того чтобы увидеть всю информацию о файлах и каталогах, необходимо использовать команду **`ls`** с ключами **`-li`**.

Пример.

```
$ ls -li .
итого 32
61997811 drwxrwxrwx 2 aer aer 4096 апр 26 2020 bin
61997770 drwxrwxrwx 3 aer aer 4096 апр 26 2020 etc
61997773 drwx-wxrwx 3 aer aer 4096 апр 26 2020 include
61870058 drwxrwx-wx 3 aer aer 4096 апр 26 2020 lib
61869170 drwxrwxrwx 2 aer aer 4096 апр 26 2020 libexec
61870062 -rwxrwx--- 1 aer aer 5218 апр 14 2020 LICENSE.md
61869172 drw-rwxrwx 8 aer aer 4096 апр 26 2020 share
```

Первый столбец – значение дескриптора[1].

Второй столбец (десять символов) определяют тип файла (1 символ) и права доступа (9 символов).

Типы:

`d` – каталог;

`-` – обычный файл.

Права доступа: первые три символа – права владельца, затем права членов группы, последние три – права всех остальных.

Число в **третьем** столбце определяет:

– *для файлов* – количество жёстких ссылок, подробнее о них в седьмой главе;

– *для каталогов* – количество вложенных каталогов +2 (добавляются два каталога, текущий и родительский каталоги).

Четвертый столбец – владелец файла (каталога).

Пятый столбец – группа, в которой состоит владелец файла.

Шестой столбец – размер файла, для каталога размером является число 4096.

Седьмой столбец – дата создания (модификации) файла.

Восьмой столбец – имя файла.

Параметр **итого (total)** определяет размер каталога в блоках.

[1] Подробнее об этом в седьмой главе.

6. ПАКЕТЫ В ОС LINUX

В ОС *Linux* приложения распространяются в виде пакетов. В дистрибутивах на базе *Debian* (*Ubuntu*, *Linux Mint*, *Astra Linux*) – это *deb-пакеты*, в *Red Hat*, *OpenSuse*, *ALT Linux* в репозитории – это *rpm-пакеты*. И те и другие хранятся в репозиториях.

Репозиторий – место хранения и поддержки каких-либо данных. Эти данные чаще всего имеют вид файлов, доступных для дальнейшего распространения по сети.

Репозиторий дистрибутива – хранилище программ дистрибутива. Все описанные далее утилиты требуют прав суперпользователя.

Утилита установки deb-пакетов `dpkg`

Основные ключи утилиты **`dpkg`**:

`dpkg -i file.deb` – установка пакета (обновление, если пакет установлен);

`dpkg -r file.deb` – удаление пакета;

`dpkg -l file` – поиск установленных пакетов (*file* – маска поиска пакета, можно использовать *, *?*);

`dpkg -L file.deb` – вывод списка файлов из установленного пакета;

`dpkg -p file.deb` – вывести информацию об установленном пакете;

`dpkg –unpack file.deb` – распаковать, но не устанавливать пакет.

Утилита `rpm`

Универсальной командой для работы с rpm-пакетами является утилита **`rpm.`** Более подробное описание утилиты можно найти на сайте https://losst.ru/ustanovka-rpm-paketov-v-linux.

Синтаксис утилиты:

```
rpm -режим-опции пакет
```

Режимы:

-q, --query – запрос, получение информации (часто используется для проверки, установлен ли пакет);

-i, --install – установка;

-V, --verify – проверка пакетов;

-U, --upgrade – обновление;

-e, --erase – удаление.

Опции программы:

-v – показать подробную информацию;

-h – выводить статус-бар;

--percent – выводить информацию в процентах о процессе распаковки;

--force – выполнять действие принудительно;

-i – получить информацию о пакете;

-l – список файлов пакета;

-R – вывести список пакетов, от которых зависит текущий.

Простейшая команда установки:

```
rpm –i имяпакета.rpm
```

Для вывода более подробной информации о процессе установки можно воспользоваться опцией `-v`:

```
rpm –iv имяпакета.rpm
```

Для включения статус бара в процессе установки служит опция `-h`:

```
rpm –ihv имяпакета.rpm
```

Для установки пакетов из репозитория используются утилиты `apt` и `apt-get`[1].

Утилита apt

Основные команды утилиты **apt**(**apt-get**):

[1] Утилиты `apt` и `apt-get` используются в дистрибутивах на базе Debian, а также в российских дистрибутивах «Альт» https://www.basealt.ru. Приводимое описание утилит `apt` (`apt-get`) относится к дистрибутивам семейства Debian (Debian, Ubuntu, Mint и т. д.). Утилита `apt-get` российских дистрибутивов «Альт» описана в восьмой главе.

apt update (apt-get update) – обновление баз данных доступных программных пакетов;

apt upgrade (apt-get upgrade) – проверяет наличие обновлений для всех установленных пакетов и предлагает загрузить и установить их;

apt dist-upgrade (apt-get dist-upgrade) – полное обновление дистрибутива до новой версии;

apt install package (apt-get install package) – загрузка и установка программного пакета по заданному названию (если он найден в базах данных);

apt -f install (apt-get -f install) – поиск неработающих программных пакетов и попытка отладить те из них, которые отмечены сообщением "unmet dependency" (неудовлетворенные зависимости);

apt autoclean (apt-get autoclean) – удаляет не полностью загруженные или ещё не установленные пакеты;

apt autoremove (apt-get autoremove) – удаляет неиспользуемые пакеты в системе;

apt remove package (apt-get remove package) – удаление пакета, остаются конфигурационные файлы;

apt purge package (apt-get purge package) – удаление пакета вместе с конфигурационными файлами;

apt-cache search package – поиск пакета (по названию и краткому описанию);

apt show package (apt-get show package) – подробная информация о пакете;

apt deb package (apt-get deb package) – установка скачанного локального deb-пакета.

Отличие `apt` и `apt-get` можно найти на сайтах:

– https://www.alv.me/Upravlenie-deb-paketami-Utilita-apt-bez-izl/;

– https://losst.ru/apt-vs-apt-get-v-chem-raznitsa.

7. ЖЁСТКИЕ И СИМВОЛЬНЫЕ ССЫЛКИ

Понятие ссылки

Каждый файл представляет собой область данных на диске.

Жесткая ссылка на файл – это ссылка, с помощью которой связывается имя файла и область данных (на диске), где располагается его содержимое. Количество таких ссылок не ограничено.

При просмотре содержимого каталога с ключом **`-i`** (`ls -i`) можно увидеть не только имя файла, но и его дескриптор (уникальное числовое значение, которое присваивается файлу при его создании).

Жёсткая ссылка

Пользователь *Linux* может создать жёсткую ссылку на файл при помощи утилиты **`ln`** (от англ. link – «соединять, связывать»). Первый параметр – это имя файла, на который нужно создать ссылку, второй – имя новой ссылки (жёсткой).

С помощью редактора `nano` создадим текстовый файл *first.txt.*

```
nano first.txt
ls -li
итого 4
9190023 -rw-rw-r-- 1 aer aer 9 окт 25 10:48 first.txt
```

Здесь `9190023` – дескриптор – уникальный числовой идентификатор, связанный с физическим местом на диске. Создадим жёсткую ссылку на файл.

```
ln first.txt second.txt
```

Создадим каталог *3*, и в нём создадим ещё одну ссылку на файл *first.txt*.

```
mkdir 3
ln first.txt 3/third.txt
ls -li -R
9190023 -rw-rw-r-- 1 aer aer 9окт 25 10:48 first.txt
9190023 -rw-rw-r-- 1 aer aer 9окт 25 10:48 second.txt
3:
итого 4
9190023 -rw-rw-r-- 1 aer aer 9 окт 25 10:48 third.txt
ln first.txt second.txt
```

Таким образом, мы имеем три файла, указывающих на одну и ту же область данных. У всех трёх файлов один и тот же дескриптор (*9190023*). Если мы удалим первый файл (*first.txt*), то файлы *second.txt, third.txt* будут по-прежнему указывать на ту же область данных.

```
rm first.txt
ls -li *.* -R
9190023 -rw-rw-r-- 1 aer aer 9окт 25 10:48 second.txt
3:
итого 4
9190023 -rw-rw-r-- 1 aer aer 9 окт 25 10:48 third.txt
```

Как мы видим, одна и та же область памяти на диске может иметь несколько имён (жёстких ссылок).

У жёстких ссылок есть два существенных ограничения:

– жёсткая ссылка не может указывать на каталог; это может привести к бесконечной циклической ссылке на один и тот же каталог;

– жёсткая ссылка может указывать на файл только в рамках одной файловой системы.

Причина последнего ограничения в том, что номер дескриптора уникален только в рамках одной файловой системы. В разных файловых системах могут оказаться два разных файла с одинаковым номером дескриптора, в результате будет невозможно установить, на какой из них указывает жёсткая ссылка.

Символьная ссылка

Символьная ссылка – файл, в котором хранится полное имя другого файла или каталога. Аналогом символьной ссылки является понятие ярлыка. Дескриптор символьной ссылки отличается от дескриптора файла (каталога), на который она ссылается.

При создании символьной ссылки используется команда `ln` с ключом **`-s`**. Рассмотрим предыдущий пример, но изменим тип создаваемой ссылки. Будем создавать символьную ссылку.

Предполагается, что файл *first.txt* создается в домашнем каталоге *user*.

С помощью текстового редактора `nano` создадим файл *first.txt*.

```
nano first.txt
```

В файле *second.txt* создадим символьную ссылку на файл *first.txt*. Обратите внимание, что при создании необходимо указать полное имя файла *first.txt*.

```
ln -s /home/user/first.txt second.txt
```

Создаём каталог с именем «*3*». В нём в файле *third.txt* создадим символьную ссылку на файл *first.txt*.

```
mkdir 3
ln -s /home/user/first.txt 3/third.txt
Перед удалением файла first.txt.
ls -li -R
9190023 -rw-rw-r-- 2 aer aer 9 окт 25 10:48 first.txt
first.txt
9190026 lrwxrwxrwx 1 aer aer 9 окт 25 11:31 second.txt ->
first.txt
3:
итого 0
9190028 lrwxrwxrwx 1 aer aer 9 окт 25 11:34 third.txt ->
first.txt
```

Удаляем файл *first.txt*.

```
rm first.txt
```

После удаления файла *first.txt*.

```
ln -li -R
итого 4
9190026 lrwxrwxrwx 1 aer aer 9 окт 25 11:31 second.txt ->
first.txt
3:
итого 0
9190028 lrwxrwxrwx 1 aer aer 9 окт 25 11:34 third.txt ->
first.txt
```

После удаления файла *first.txt* символьные ссылки *second.txt* и *third.txt* указывают на несуществующий файл.

8. ОСОБЕННОСТИ НАСТРОЙКИ РОССИЙСКИХ ДИСТРИБУТИВОВ «АЛЬТ» ПОСЛЕ УСТАНОВКИ

Под первоначальной настройкой будем понимать нестандартную установку необходимого программного обеспечения, а не настройку визуальных эффектов. Мы рассмотрим все возможные способы установки программ в дистрибутивах семейства «Альт». Но эти способы должны работать в каждой системе, построенной на базе любого дистрибутива «Альт» десятой платформы. В коммерческих дистрибутивах (и в *Simply Linux*) часть из описанного далее уже выполнена автоматически.

Обновление системы

Обновление системы выполняется от имени администратора. Рассмотрим основные возможности управления пользователями в дистрибутивах семейства «Альт» десятой платформы.

Управление пользователями в «Альт». Администратор системы

Смена пользователя в простейшем случае осуществляется командой

`su [-] [user]`

Здесь *user* – имя пользователя, от имени которого осуществляется вход в систему. Если имя пользователя отсутствует, то речь идёт об администраторе (пользователь *root*).

После ввода команды необходимо ввести пароль пользователя *user*. Особую роль играет символ **-** (минус).

Если после команды **su** ввести символ минус (**`su - user`**), то вход в систему будет произведён от имени пользователя *user*, и домашним каталогом будет `/home/user`. Все настройки системы в таком случае хранятся в этом же каталоге `/home/user`.

Если же знак минус отсутствует в команде **`su user`**, то вход в систему будет произведён от имени пользователя *user*, но с домашним каталогом и настройками текущего пользователя.

После выполнения команды **`su`** и ввода пароля администратора вы получаете права администратора и можете выполнять любые действия. Однако следует помнить, что домашним каталогом остаётся каталог текущего пользователя системы и сохраняются *все настройки системы текущего пользователя*.

После выполнения команды **`su -`** и ввода пароля администратора ситуация становится другой. Вы входите в систему как администратор. Домашним каталогом становится каталог `/root`. Настройки системы – *настройки пользователя root*. Именно у пользователя *root* в настройках описаны пути ко всем системным программам. И поэтому рекомендуется перед выполнением команд обслуживания системы получить не просто права администратора (`su`), а именно стать администратором (`su -`).

Команды для обновление системы

Обновление системы осуществляется с помощью двух команд, которые должны быть выполнены от имени администратора (пользователь *root* (`su -`)):

`apt-get update` – обновление баз данных доступных программных пакетов;

`apt-get dist-upgrade` – команда проверяет наличие обновлений для всех установленных пакетов и предлагает загрузить и установить их.

Источники программного обеспечения

Программное обеспечение может быть установлено из *репозитория ALT Linux*, непосредственно из *rpm-пакета*, а также

с помощью особого механизма установки популярных приложений, в том числе и проприетарных.

Установка программного обеспечения из репозитория (Sisyphus)

Свободные приложения могут быть установлены из репозитория *ALT Linux* – `Sisyphus` («Сизиф»). В терминале можно воспользоваться командой `apt-get` (от администратора):

`apt-get install package` – загрузка и установка локального пакета или пакета из репозитория;

`apt-get -f install` – поиск неработающих программных пакетов и попытка отладить те из них, которые отмечены сообщением "unmet dependency" (неудовлетворенные зависимости);

`apt-get autoclean` – удаляет не полностью загруженные или ещё не установленные пакеты;

`apt-get autoremove` – удаляет неиспользуемые пакеты в системе;

`apt-get remove package` – удаление пакета, остаются конфигурационные файлы;

`apt-get purge package` – удаление пакета вместе с конфигурационными файлами;

`apt-cache search package` – поиск пакета (по названию и краткому описанию);

`apt-get show package` – подробная информация о пакете.

Установку, удаление и обновление приложений можно осуществлять с помощью приложения с графическим интерфейсом `Synaptic` (рис. 8.1).

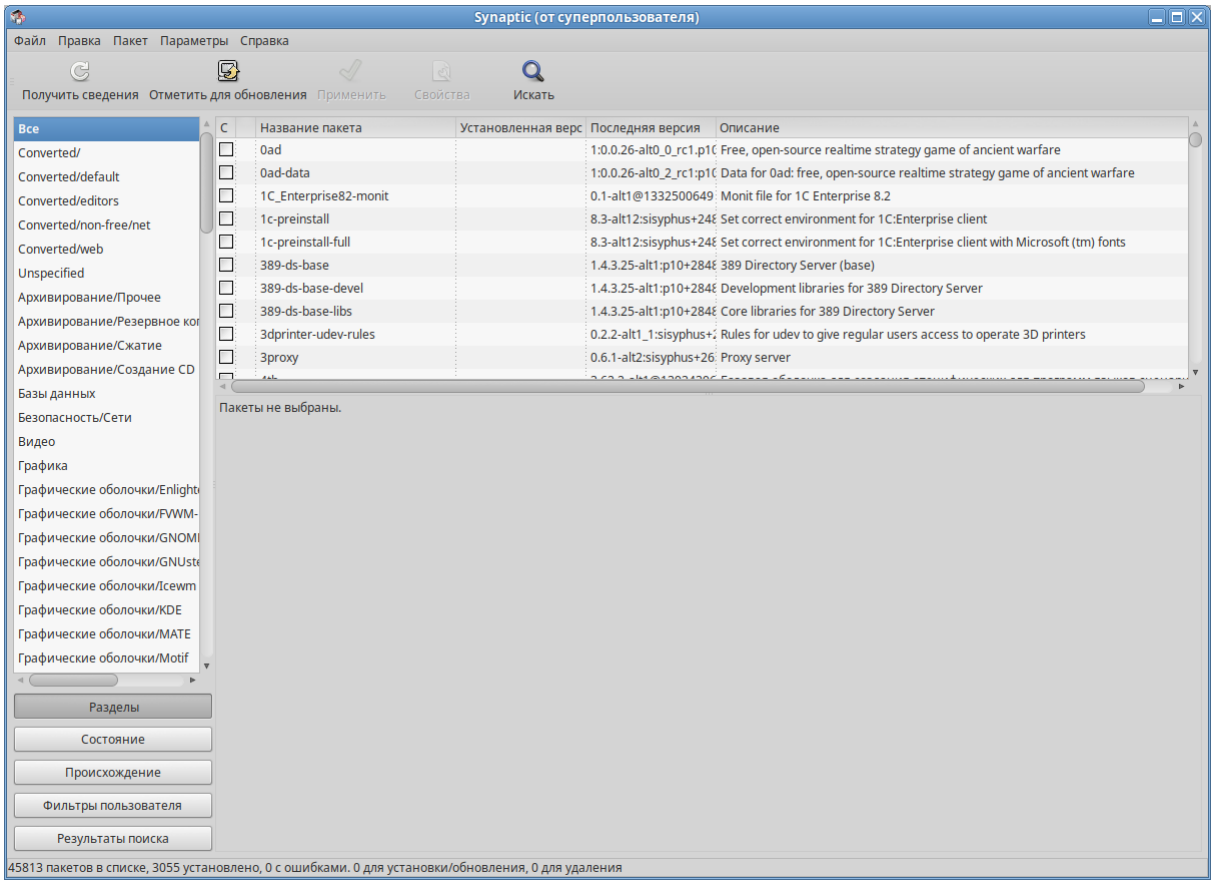


Рис. 8.1. Менеджер пакетов Synaptic

Отдельные локальные rpm-пакеты могут быть также установлены с помощью утилиты `rpm` (см. шестую главу).

Установка бесплатного и проприетарного ПО

Современный пользователь компьютера привык к популярным бесплатным кроссплатформенным приложениям (*Geogebra, Yandex Disk,* браузеры *Google Chrome, Yandex* и др.). В дистрибутивах семейства «Альт» десятой платформы автоматизирован процесс установки этих программ. Существует графическое приложение `appinstall` (рис. 8.2[1]).

[1] Если приложение не установлено, то его можно установить стандартным образом `apt-get install appinstall`.

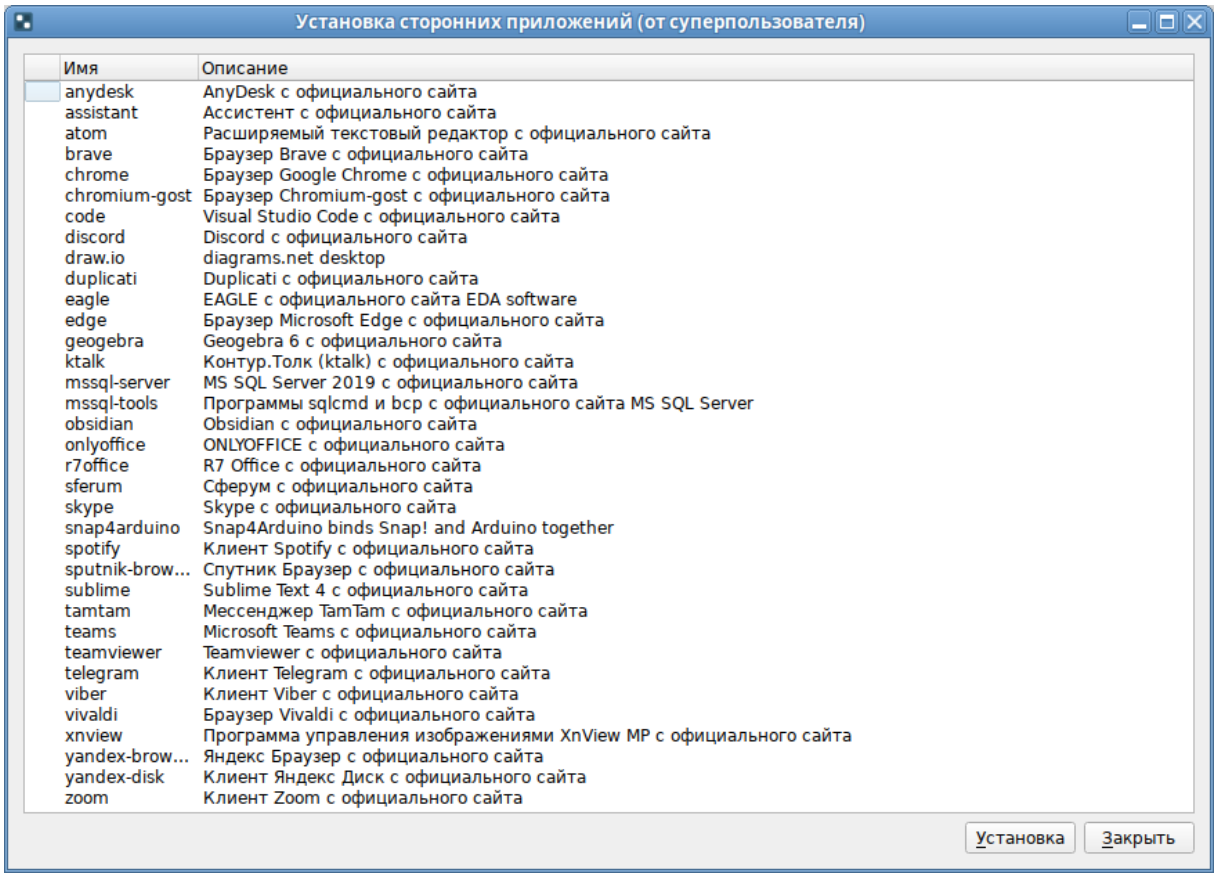


Рис. 8.2. Утилита appinstall

С помощью этой программы можно установить любое приложение из списка. Установка включает скачивание последней версии приложения с официального сайта и непосредственно установку.

Приложение `appinstall` является графическим интерфейсом над консольной утилитой **`epm`**.

Утилита `epm`[1]

Для установки, удаления и обновления программ с сайта разработчика необходимо выполнять команду **`epm play`** от администратора.

[1] Если приложение не установлено, то его можно установить стандартнымобразом `apt-get install epm`

```
epm p lay
Available applications:
anydesk - AnyDesk from the official site
assistant - Assistant (Ассистент) from the official site
atom - The hackable text editor from the official site
brave - Brave browser from the official site
chrome - The popular and trusted web browser by Google
(Stable Channel) from the official site
chromium-gost - Chromium with GOST support from the official
site
code - Visual Studio Code from the official site
discord - Discord from the official site
draw.io - diagrams.net desktop
duplicati - Duplicati from the official site
eagle - EAGLE (EDA software) from the official site
edge - Microsoft Edge browser(dev) from the official site
geogebra - Geogebra 6 from the official site
ktalk - Контур.Толк (ktalk) from the official site
mssql-server - MS SQL Server 2019 from the official site
mssql-tools - MS SQL Server sqlcmd and bcp from the official
site
obsidian - Obsidian from the official site
onlyoffice - ONLYOFFICE for Linux from the official site
r7office - R7 Office for Linux from the official site
sferum - Sferum for Linux from the official site
skype - Skype for Linux from the official site
snap4arduino - Snap4Arduino binds Snap! and Arduino
together
spotify - Spotify client for Linux from the official site
sputnik-browser - Sputnik browser from the official site
sublime - Sublime Text 4 from the official site
tamtam - TamTam messenger from the official site
teams - Microsoft Teams for Linux from the official site
teamviewer - Teamviewer from the official site
telegram - Telegram client from the official site
viber - Viber for Linux from the official site
vivaldi - Vivaldi browser from the official site
xnview -XnView MP:Image management from the official site
yandex-browser - Yandex browser from the official site
yandex-disk - Yandex Disk from the official site
zoom - Zoom client from the official site
```

Синтаксис утилиты `epm`:

```
epm play [options] [<app>]
```

Опции могут быть следующие:

<app> – установка приложения <app>;

--remove <app> – удаление приложения <app>;

-- update [<app>|all] – обновление приложения <app> (или всех установленных приложений), если доступны новые версии;

--list – вывод списка всех установленных приложений с помощью epm;

--list-all – вывод списка всех доступных приложений;

--list-scripts – вывод списка всех доступных установочных скриптов;

--short (with --list) – вывод списка только имён всех доступных приложений;

--installed <app> – проверка, установлено ли приложение с именем app.

Набор необходимого ПО для науки и образования, особенности установки и настройки

В репозиторий «Альт» десятой платформы включены скрипты для установки наборов образовательного ПО:

task-edu-gradeschool – образовательное программное обеспечение (начальная школа);

task-edu-highschool – образовательное программное обеспечение (средняя школа);

task-edu-kde5 – среда KDE5 для «Альт Образование»;

task-edu-lite – базовый набор образовательного ПО, облегчённый для RPi4;

task-edu-preschool – образовательное программное обеспечение (дошкольное образование);

task-edu-school – образовательное программное обеспечение для школ;

task-edu-secondary-vocational – образовательное программное обеспечение (среднее профессиональное образование);

task-edu-server-apps – образовательное программное обеспечение (серверные приложения);

task-edu-teacher – образовательное программное обеспечение (для учителей);

task-edu-tools – вспомогательные программы для «Альт Образование»;

task-edu-university – образовательное программное обеспечение (высшее образование);

task-edu-video-conferencing – образовательное программное обеспечение (сервер видеоконференций).

Пример установки программного обеспечения для университета – пакеты `task-edu`, `task-edu-university`

```
su -
apt-get install task-edu task-edu-university
```

Установка этих скриптов приводит к установке большого количества образовательных и научных приложений.

Создание установочных флешек

Для создания загрузочной (установочной) флешки можно воспользоваться графическим приложением *AltImageWriter* (установка из репозитория `apt-get install altmediawriter`) или консольной утилитой `dd`.

Следует учитывать, что при создании загрузочной флешки данные, которые находились на USB-устройстве, не сохранятся.

Для создания загрузочной флешки необходим файл образа (файл с расширением *iso*) дистрибутива *ALT Linux*, который можно скачать на сайте https://getalt.org/ru/. Для первого знакомства с *ALT Linux* рекомендуется использовать *ALT Workstation*, *ALT Образование* или *Simply Linux*.

Утилита `altmediwriter`

Рассмотрим, как создать загрузочную флешку с помощью приложения *AltImageWriter* (рис. 8.3).

Утилита *AltImageWriter* позволяет скачать файл образа одного из дистрибутивов семейства «Альт» (*«Альт Рабочая Станция», «Альт Образование», Simply Linux* и др.), а затем записать его на флешку. Следует учитывать, что современный дистрибутив занимает много места. Для скачивания файла образа должен быть стабильный Интернет. Если образ выкачан заранее и есть файл с расширением `iso`, следует выбрать режим «*Другой образ*». После этого необходимо выбрать файл с образом и USB-накопитель, на который будет записан образ (рис. 8.4). Утилита *AltImageWriter* является универсальной, она создает загрузочную флешку с любым дистрибутивом *Linux*.

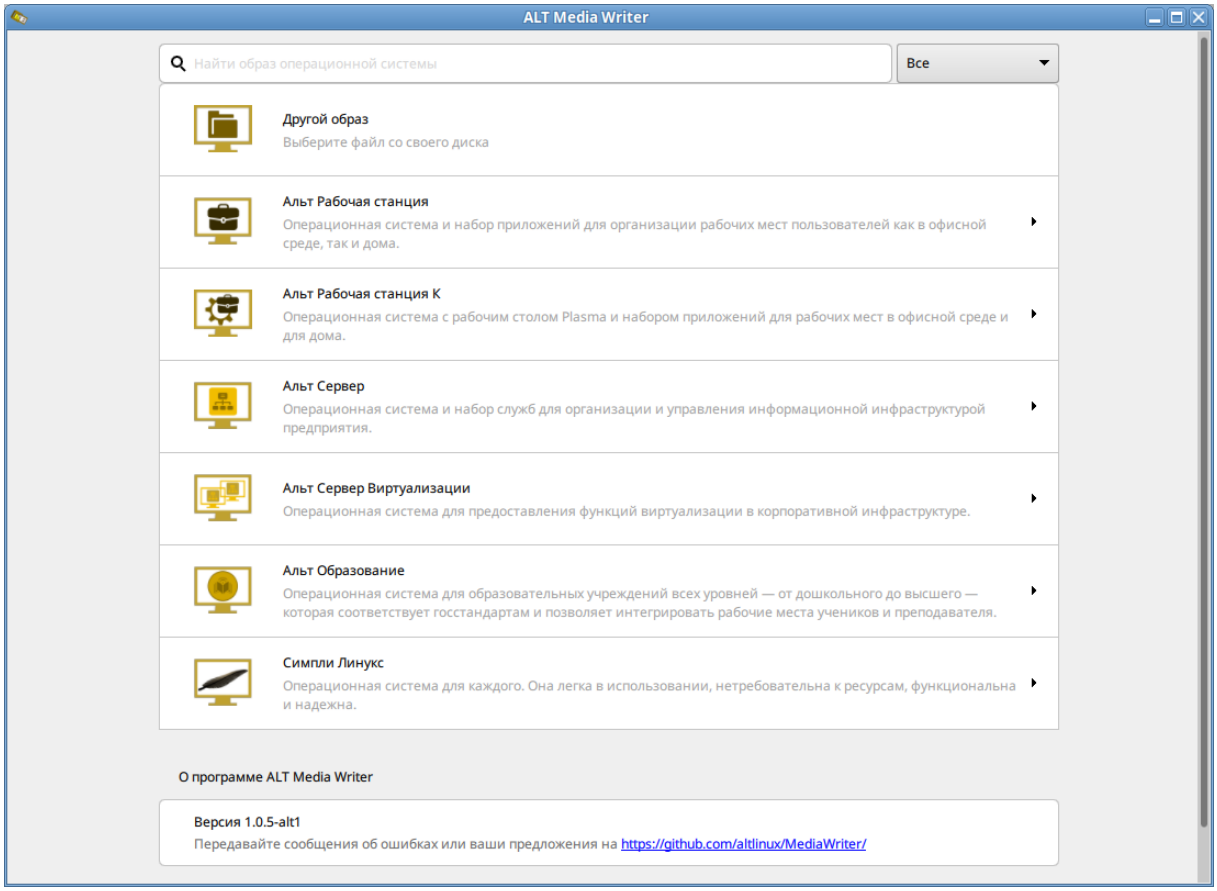


Рис. 8.3. Утилита altmediawriter

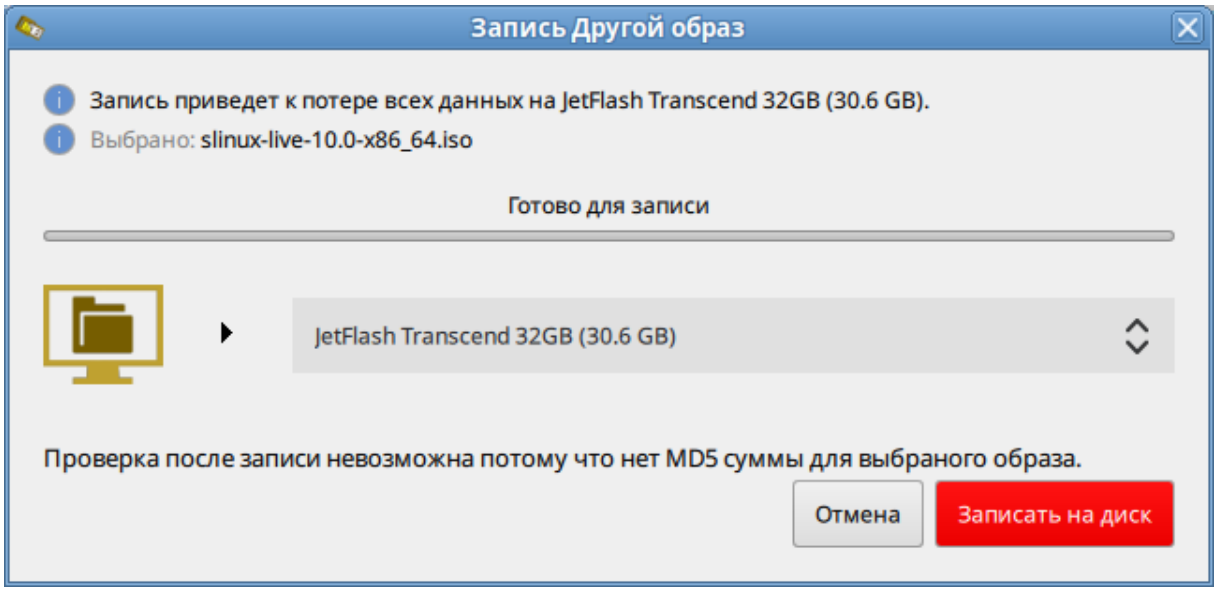


Рис. 8.4. Запись образа

Для создания загрузочной флешки в операционной системе *Windows* разработчики *ALT Linux* советуют использовать *AltImageWriter* (https://github.com/altlinux/ALTMediaWriter) или *Win32 Disk Image* (https://sourceforge.net/projects/win32diskimager). Альтернативой является известная в мире *Windows* утилита *Rufus* (https://rufus.ie/ru/).

9. УСТАНОВКА LINUX НА КОМПЬЮТЕР ПОЛЬЗОВАТЕЛЯ

Установку ОС *Linux* рассмотрим на примере дистрибутива *Linux Mint*, так как он прост в использовании и подходит для начинающих пользователей.

Для того чтобы установить дистрибутив *Linux* (*Linux Mint* или любой другой), необходимо скачать файл `iso` (образ дистрибутива), записать образ на флешку, запустить дистрибутив в *Live-режиме*, запустить в *Live-режиме* инсталятор ОС и следовать инструкциям на экране. Теперь рассмотрим процесс установки более подробно.

Получение образа ОС Linux

Ссылки для скачивания некоторых современных дистрибутивов представлены в таблице.

Ссылки для скачивания современных дистрибутивов

Дистрибутив	Ссылка
Debian	https://www.debian.org/distrib/
LinuxMint	https://linuxmint.com/download_all.php
Ubuntu	https://ubuntu.ru/get
Ubuntu Mate	https://ubuntu-mate.org/ru/download/
ALT Linux	https://getalt.org/ru/
Runtu	https://runtu.org/download/
Red OS	https://redos.red-soft.ru/product/downloads/
GreenLinux	https://greenlinux.ru/download/

После скачивания *iso-образа* актуальной версии дистрибутива необходимо создать загрузочную флешку.

Запись образа на USB

Для записи образа ОС на USB под управлением *Windows* необходима сторонняя программа *Rufus*. Скачать её можно с сайта https://rufus.ie/ru. При записи образа на USB, устройство форматируется. Все важные данные необходимо сохранить на другом носителе. После запуска программы необходимо (рис. 9.1):

1) выбрать носитель (в графе *«Устройство»*);

2) определить *iso*-файл (щелкнуть по кнопке *«Выбрать»*);

3) создать схему разделов: *GPT* или *MBR* (в современных устройствах чаще *GPT*);

4) нажать *«Старт»*.

Далее подтвердить форматирование устройства.

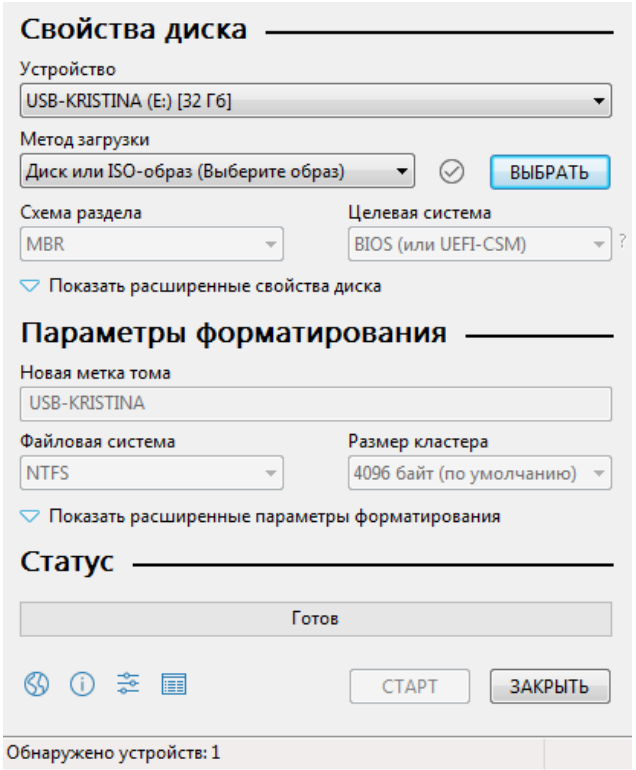


Рис. 9.1. Утилита rufus

Консольная утилита dd

Консольная утилита `dd` предназначена для создания загрузочной флешки путем прямой записи iso-файла на USB-устройства.

Команду `dd` нужно выполнять с правами администратора.

```
dd if=name .iso of=/dev/sdX status=progress
```

Здесь

`name.iso` – имя *iso*-файла дистрибутива вашей операционной системы.

`/dev/sdX` – файл устройства флешки.

Имя файла флешки можно узнать с помощью утилит `df` и `fdisk`.

Рассмотрим результат вывода утилиты `df`.

```
(base) aer@aer Загрузки $ df
Файловая система Размер Использовано  Дост Использовано% Смонтировано в
udevfs              3,8G          96K  3,8G             1% /dev
runfs               3,8G         1,7M  3,8G             1% /run
/dev/sda6           114G          58G   50G            54% /
tmpfs               3,8G          16K  3,8G             1% /dev/shm
tmpfs               3,8G          11M  3,8G             1% /tmp
/dev/loop0          172M         172M     0           100% /var/lib/snapd/snap/codium/262
/dev/loop1           47M          47M     0           100% /var/lib/snapd/snap/snapd/16292
/dev/loop2           56M          56M     0           100% /var/lib/snapd/snap/core18/2538
/dev/sda4           510M         422M   89M            83% /mnt/sdb4
/dev/sda3           107G          41G   67G            38% /mnt/sdb3
/dev/sda1            96M          35M   62M            37% /mnt/sdb1
/dev/sdb6           118G         8,0G  110G             7% /mnt/sda6
/dev/sdb5           801G         690G   71G            91% /DATA
tmpfs               776M          96K  776M             1% /run/user/500
/dev/sdc1   5,7G   5,7G     0    100% /run/media/aer/ALT Workstation 10.0 x86_64
```

Очевидно, что в данном случае файл устройства флешки – `/dev/sdc1`.

Консольная утилита `fdisk` предназначена для разметки дисков вашего компьютера, поэтому не следует использовать её без ключа `-l` (просмотр разделов дисков). Утилита выполняется от имени администратора. Рассмотрим результат её выполнения.

```
su -
fdisk -l
Диск /dev/sdb: 931,51 GiB, 1000204886016 байт, 1953525168 секторов
Disk model: TOSHIBA MQ04ABF1
Единицы: секторов по 1 * 512 = 512 байт
Размер сектора (логический/физический): 512 байт / 4096 байт
Размер I/O (минимальный/оптимальный): 4096 байт / 4096 байт
Тип метки диска: dos
Идентификатор диска: 0xa2bb2c02
Устр-во    Загрузочный    начало      Конец    Секторы Размер Идентификатор Тип
/dev/sdb4                  4094 1953523627 1953519534 931,5G             5 Расширенный
/dev/sdb5             246147072 1953521663 1707374592 814,1G            83 Linux
/dev/sdb6                  6144  246145023  246138880 117,4G      7 HPFS/NTFS/exFAT
Диск /dev/sda: 223,57 GiB, 240057409536 байт, 468862128 секторов
Disk model: TS240GMTS420S
Единицы: секторов по 1 * 512 = 512 байт
Размер сектора (логический/физический): 512 байт / 512 байт
Размер I/O (минимальный/оптимальный): 512 байт / 512 байт
Тип метки диска: gpt
Идентификатор диска: 32A24718-B0D7-4DB0-8D34-142FF84726E7
Устр-во       начало     Конец   Секторы Размер Тип
/dev/sda1       2048    206847    204800   100M EFI
/dev/sda2     206848    239615     32768    16M Зарезервированный раздел Microsoft
/dev/sda3     239616 224233672 223994057 106,8G Microsoft basic data
/dev/sda4  224235520 225279999   1044480   510M Среда для восстановления Microsoft
/dev/sda5  225280000 225492991    212992   104M EFI
/dev/sda6  225492992 468856831 243363840   116G Файловая система Linux
Диск /dev/sdc: 28,48 GiB, 30579621888 байт, 59725824 секторов
Disk model: Transcend 32GB
Единицы: секторов по 1 * 512 = 512 байт
Размер сектора (логический/физический): 512 байт / 512 байт
Размер I/O (минимальный/оптимальный): 512 байт / 512 байт
Тип метки диска: dos
Идентификатор диска: 0xbc616086
Устр-во    Загрузочный начало    Конец  Секторы Размер Идентификатор Тип
/dev/sdc1  *               64 11870207 11870144   5,7G             0 Пустой
/dev/sdc2                4308    16787    12480   6,1M            ef EFI (FAT-12/16/32)
```

В результатах вывода утилиты `fdisk -l` также можно найти информацию, что файл устройства флешки – `/dev/sdc`.

Установка Linux рядом с Windows

Чтобы установить *ОС семейства Linux* рядом с *Windows,* необходимо предварительно выделить на диске место, которое будет использоваться *ОС Linux*. Для этого нужно через меню *«Пуск»* найти утилиту *«Управление компьютером»*. В списке слева выбрать пункт *«Управление дисками»* (рис. 9.2), после чего появится таблица со списком разделов. От раздела, в котором есть свободное место, нужно отрезать *30–100 Gb* для будущей ОС. Для

этого правой кнопкой мышки щелкнуть по разделу и во всплывающем меню выбрать *«Сжать том»* (рис. 9.3).

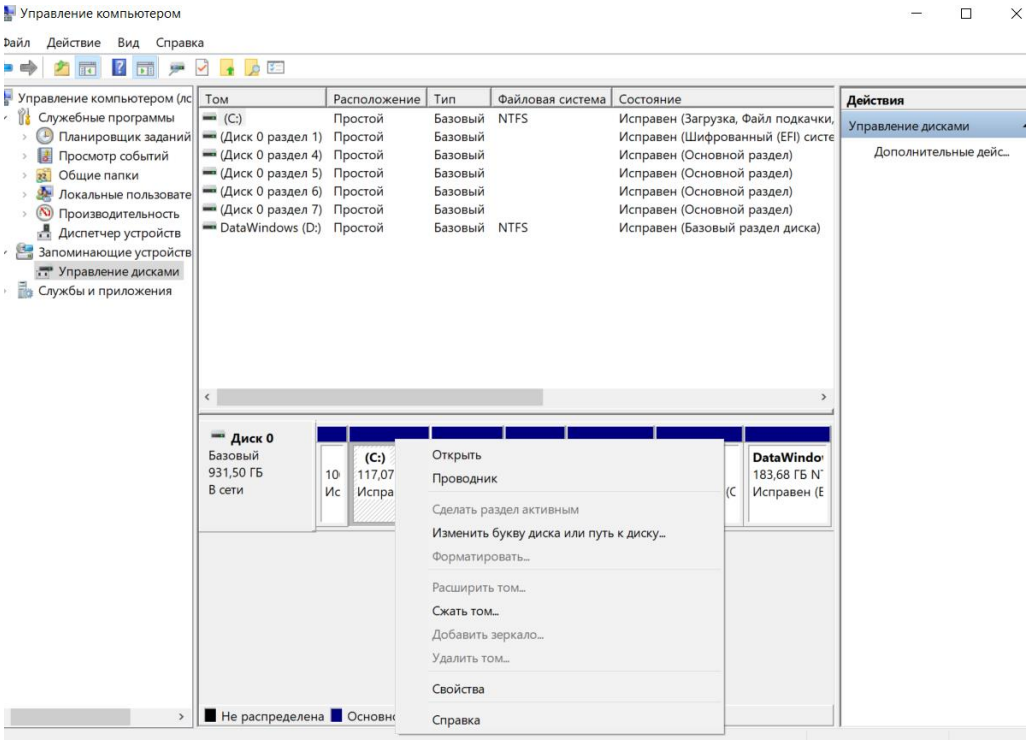


Рис. 9.2. Управление дисками

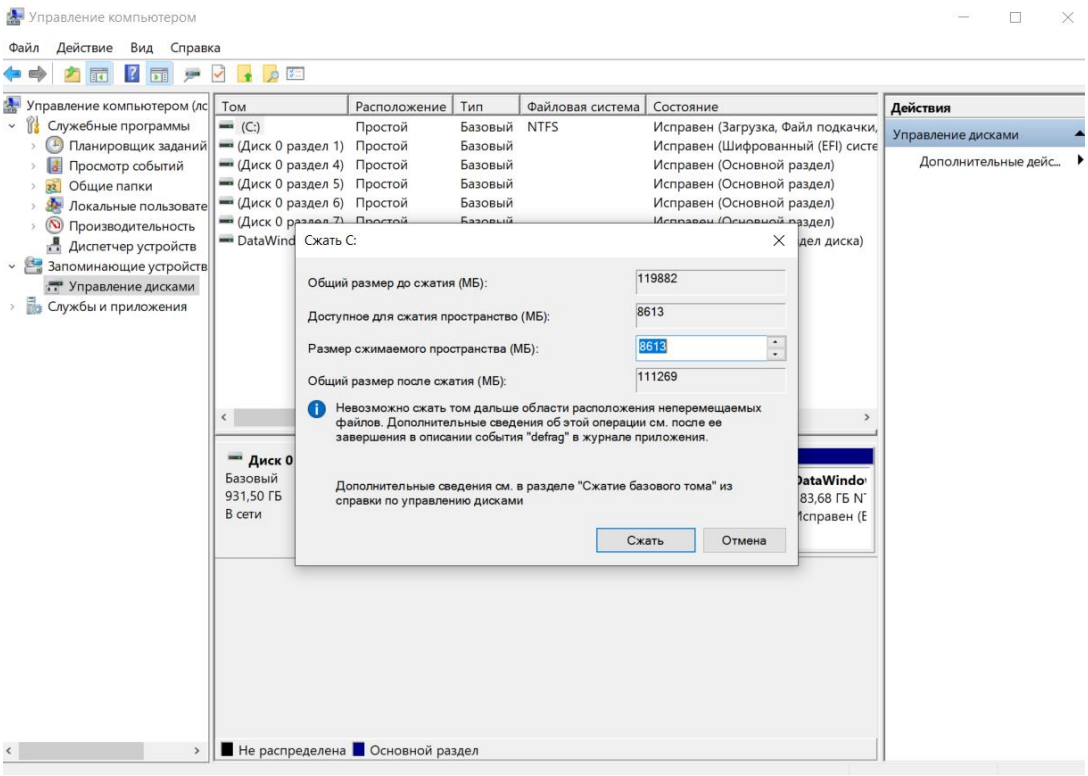


Рис. 9.3. Сжатие тома

В поле *«Размер сжимаемого пространства»* нужно указать, сколько места будет отрезано от раздела. Нажать кнопку *«Сжать»*.

В результате в таблице разделов появится пустая область указанного размера.

Модификацию размеров диска можно осуществить и в процессе установки *ОС семейства Linux*.

Загрузка LiveCD

Чтобы загрузить ОС в *Live-режиме*, нужно зайти в *BIOS/UEFI* и изменить порядок загрузки. На первое место поставить загрузку с USB.

Вход в *BIOS* на разных компьютерах осуществляется по-разному. Необходимо при включении компьютера удерживать определённую клавишу, но у разных производителей она отличается. Иногда это *F1, F2, F10, F11, Esc, Del*. После изменения порядка загрузки требуется сохранить изменения и перезагрузить компьютер.

Загрузочная флешка должна быть вставлена в USB-порт до перезагрузки. После перезапуска начнётся загрузка *Linux* в *Live-режиме*[1], появится загрузочное меню. Выбрать пункт *«Start Linux»*, после чего появится рабочий стол (рис. 9.4).

[1] Если Вами выбран дистрибутив без *Live-режима*, то процесс установки начнётся сразу.

Рис. 9.4. Рабочий стол Linux Mint

Установка Linux

Чтобы установить систему на компьютер, нужно дважды щелкнуть по иконке *Install Linux Mint*, которая расположена на рабочем столе.

После этого запускается инсталятор[1]. В открывшемся окне необходимо последовательно выбрать язык будущей системы, установить дополнительные компоненты (*драйвера видеокарты, WiFi, поддержка Flash, MP3* и др.), раскладку клавиатуры (рис. 9.5–9.7).

[1] Здесь и далее представлен процесс установки *Linux Mint*, но логика и основные принципы установки в других дистрибутивах похожи.

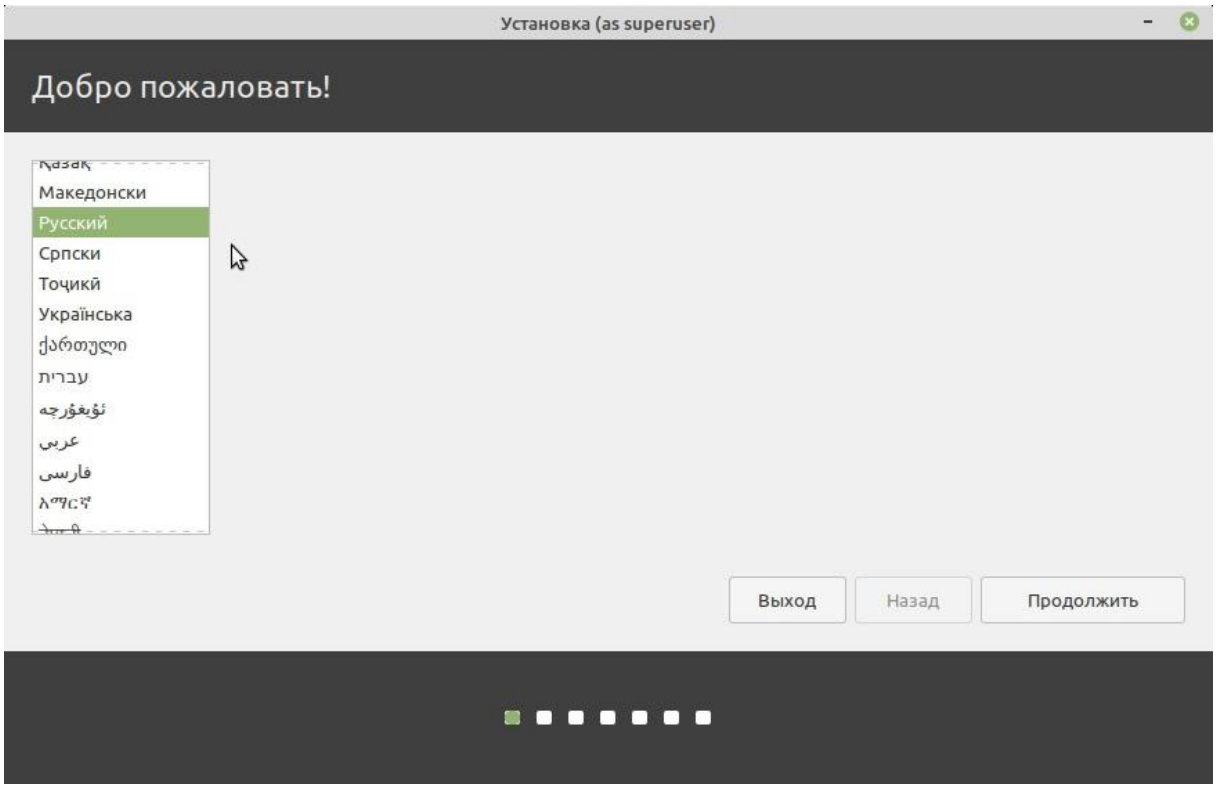


Рис. 9.5. Выбор языка установки

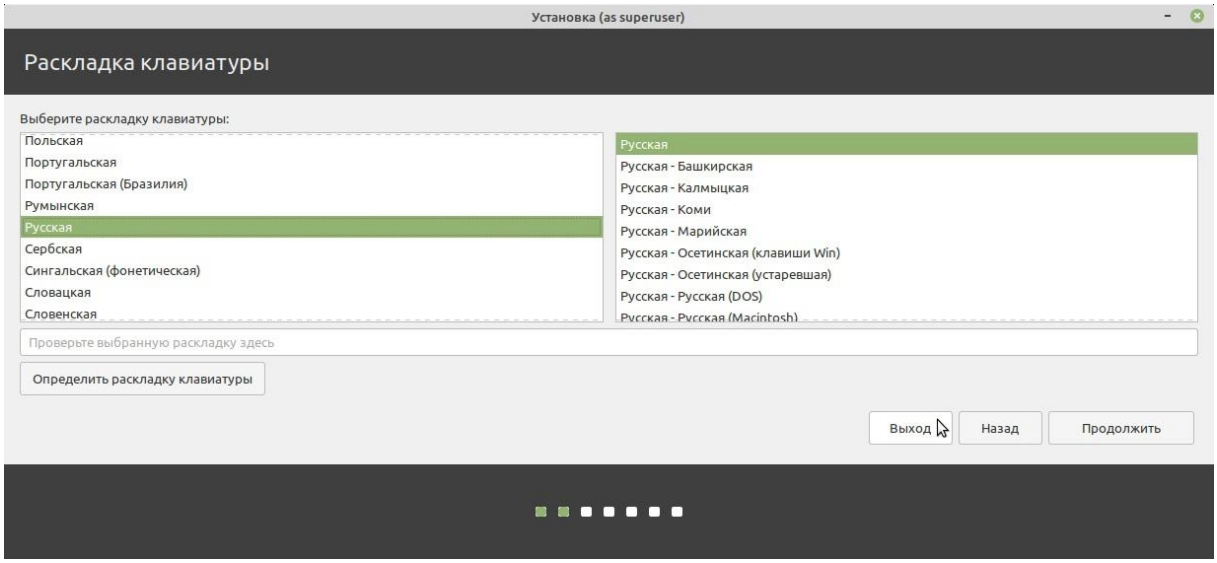


Рис. 9.6. Выбор раскладки клавиатуры

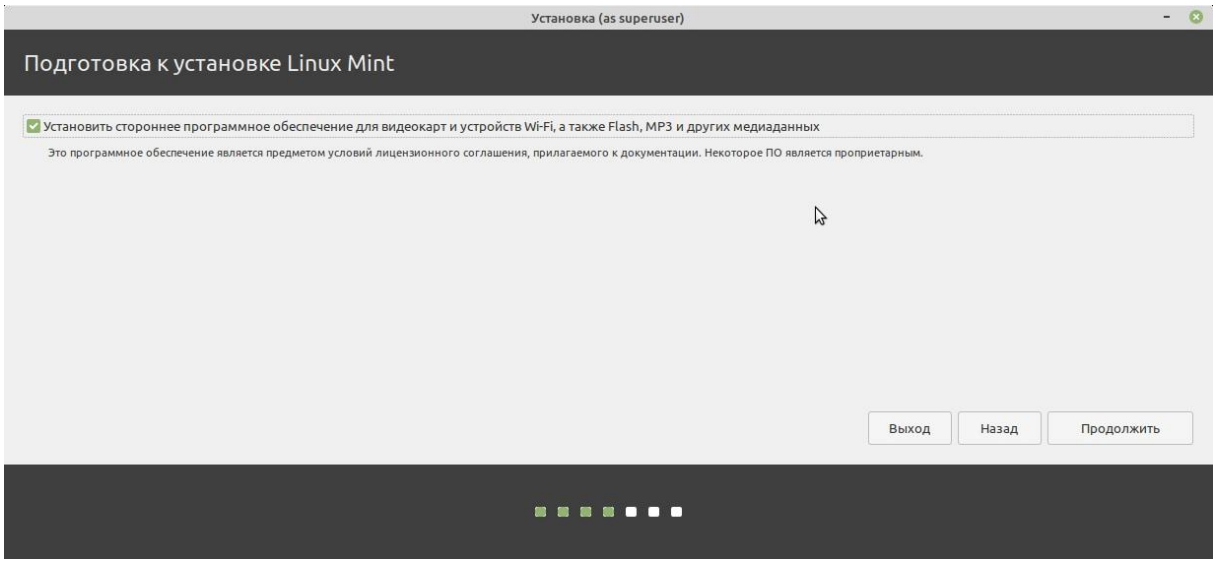


Рис. 9.7. Дополнительные компоненты при установке

Далее в окне появится список типов установки. Выбрать *«Другой вариант»* для того, чтобы можно было самостоятельно разметить диск.

Особенности разметки диска при установке Linux

Создаваемые на диске разделы могут отличаться размером, файловой системой, назначением. Linux может занимать несколько разделов, каждый из которых служит для своих целей:

1) / (*«корень»*) – основной (корневой) раздел, в котором размещаются все каталоги системы;

2) `/home` – домашний раздел, в котором размещаются пользовательские данные;

3) `swap` – раздел подкачки, используется системой, если не хватает оперативной памяти.

Жесткий диск обычно обозначается как `/dev/sda`. В случае, когда дисков несколько, названия будут `/dev/sdb`, `/dev/sdc` и т. д. В списке уже будет несколько разделов (разделы других операционных систем) (рис. 9.8).

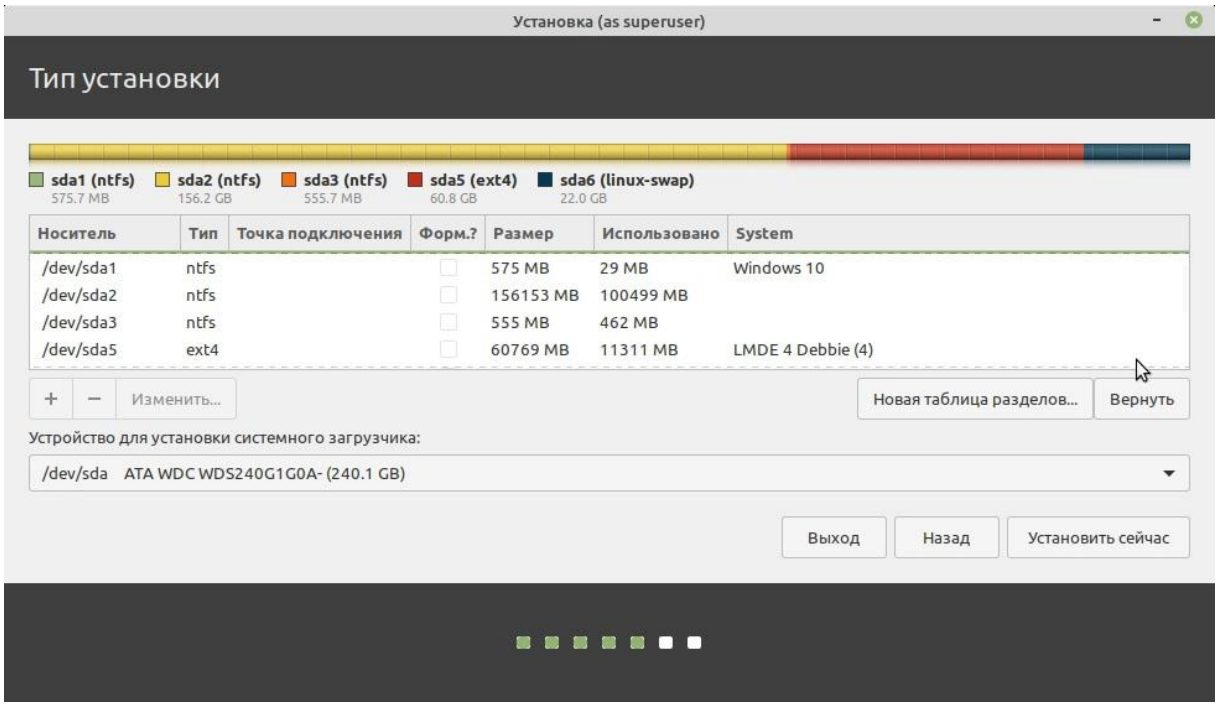


Рис. 9.8. Разделы диска

Если вы предварительно удалили ненужные разделы или уменьшали какой-либо раздел, то в карте диска будет строка *«свободное место»*. В этом случае вам можно будет создавать новые разделы, используя кнопку *«+»*.

Если же свободного места нет, то придётся освобождать его на каком-либо разделе прямо сейчас, используя кнопку «*Изменить»*. После чего на карте диска появится свободное место.

Рассмотрим более подробно процесс создания необходимых разделов. Начнём с раздела подкачки (*swap-раздела*).

В таблице разделов выделить поле *«свободное место»* и нажать кнопку *«+»*. В настройках необходимо:

1) указать размер раздела в мегабайтах. Размер раздела подкачки зависит от параметров компьютера. В среднем это *4–16 Gb*. Рекомендуется отводить для раздела подкачки 1–2 объёма оперативной памяти[1];

2) определить местоположение нового раздела (начало этого пространства);

3) использовать как раздел подкачки.

[1] Если у вас 8 Гб ОЗУ и более, то можно и не использовать раздел подкачки.

Теперь нужно создать корневой раздел / и, возможно, раздел /home. Алгоритм будет одинаковый для обоих разделов[1].

Для этого в таблице разделов снова выделить поле *«свободное место»* и нажать *«+»*. В настройках необходимо (рис. 9.9):

1) указать размер раздела в мегабайтах (минимум *20 Gb* – для / и оставшееся – для /home, если он планируется как отдельный раздел);

2) определить местоположение нового раздела – начало этого пространства;

3) использовать как *журналируемую файловую систему Ext4*;

4) указать точку монтирования / (/home).

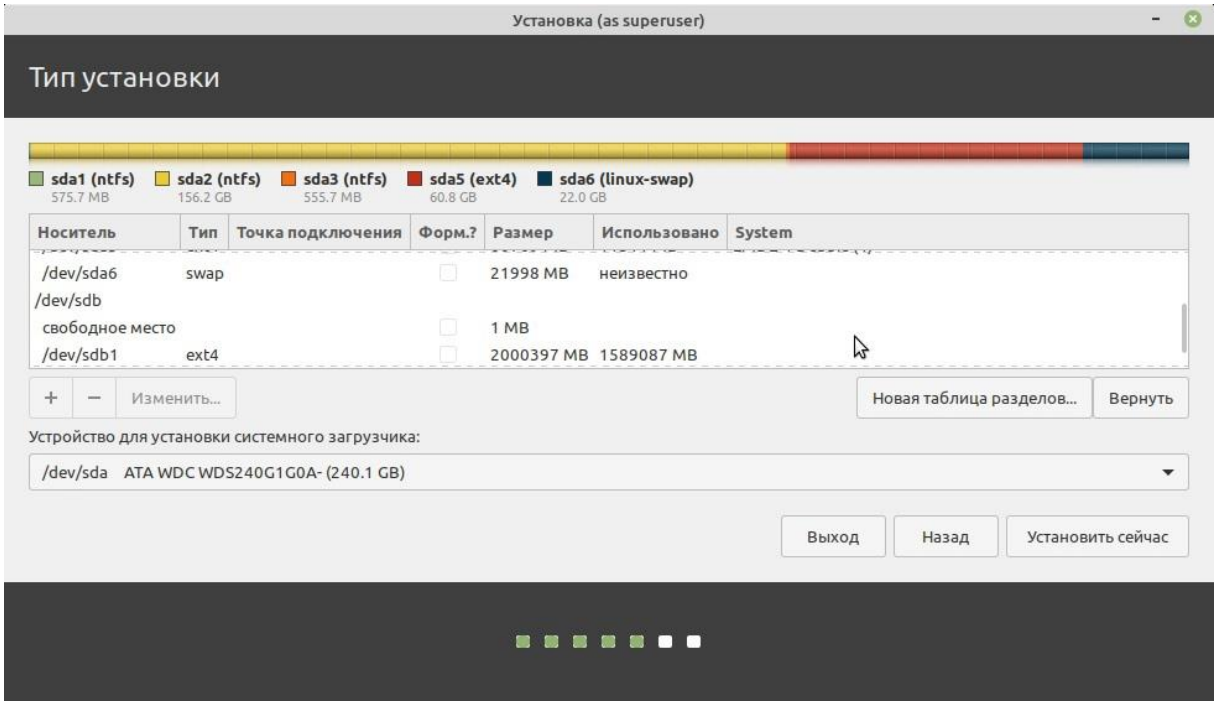


Рис. 9.9. Создание корневого раздела

[1] Если раздел /home не создавать, то каталог /home будет создан в корневом разделе.

На современных компьютерах должен существовать загрузочный раздел *EFI* (файловая система *FAT32*, размер *100–300 Мб*). Поэтому при разметке диска для современной версии *Linux* нужно либо создать его, либо проверить его наличие в другой операционной системе (рис. 9.10).

После этого можно завершать разметку диска.

Далее необходимо выбрать часовой пояс (рис. 9.11) и создать пользователя (рис. 9.12).

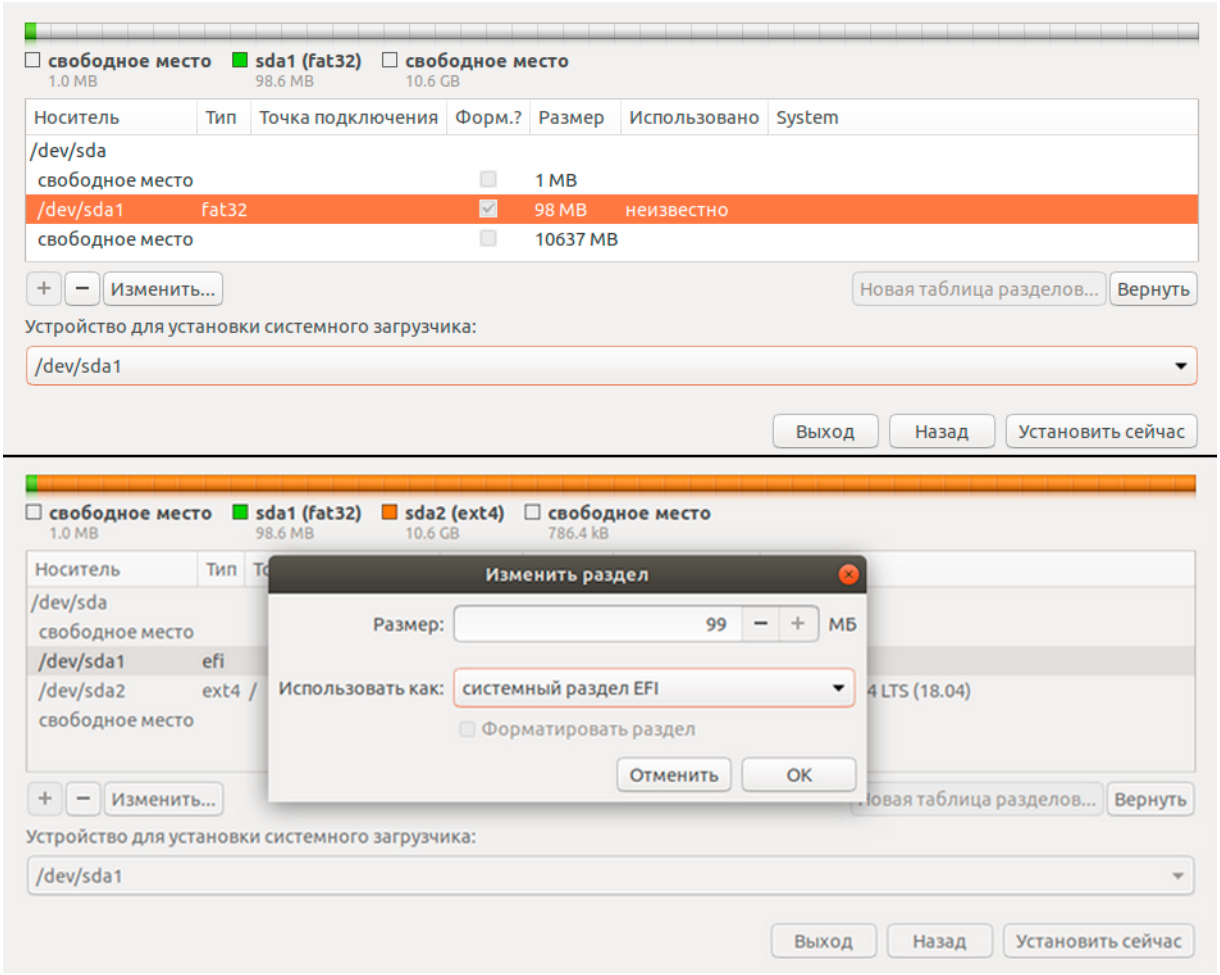


Рис. 9.10. Раздел EFI

Рис. 9.11. Выбор часового пояса

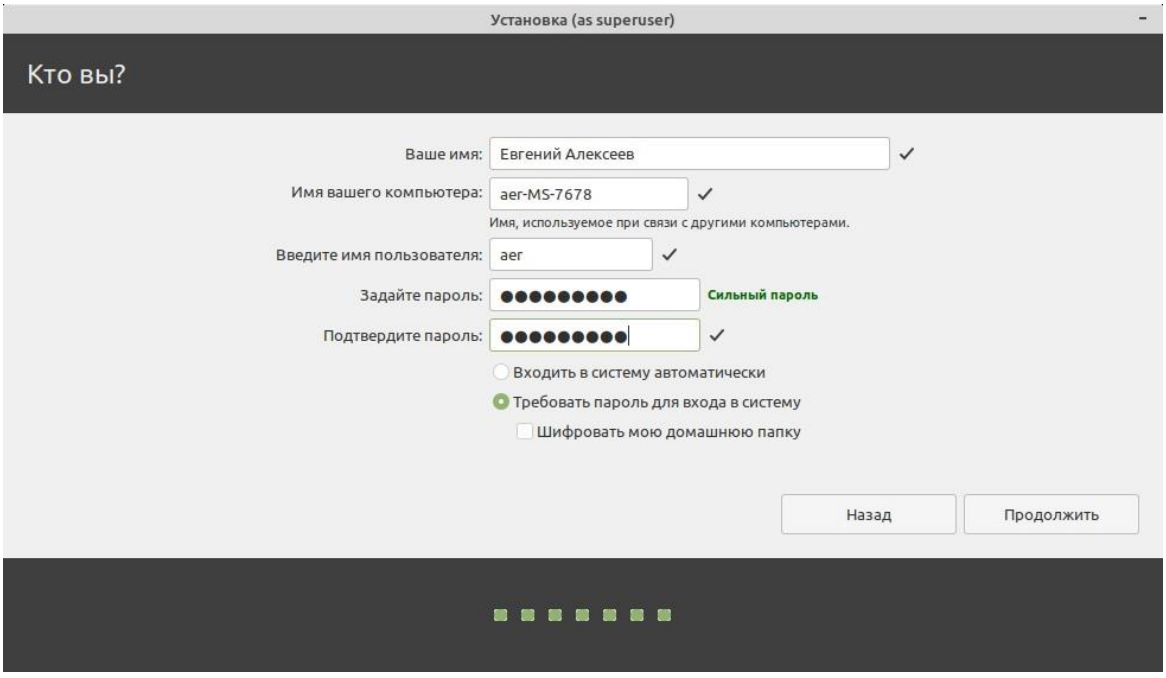


Рис. 9.12. Создание пользователя

При создании пользователя указать:

1) ваше имя;
2) имя вашего компьютера;
3) имя пользователя;
4) пароль и повтор пароля.

Далее выбрать *«Требовать пароль для входа в систему»* или *«Входить в систему автоматически»*, если при входе ввод пароля не нужен.

Существует возможность выбрать опцию *«Шифровать мою домашнюю папку»*. Начинающему пользователю этого делать не нужно.

После указания всех настроек нажать кнопку *«Продолжить»*. Начинается процесс установки (рис. 9.13).

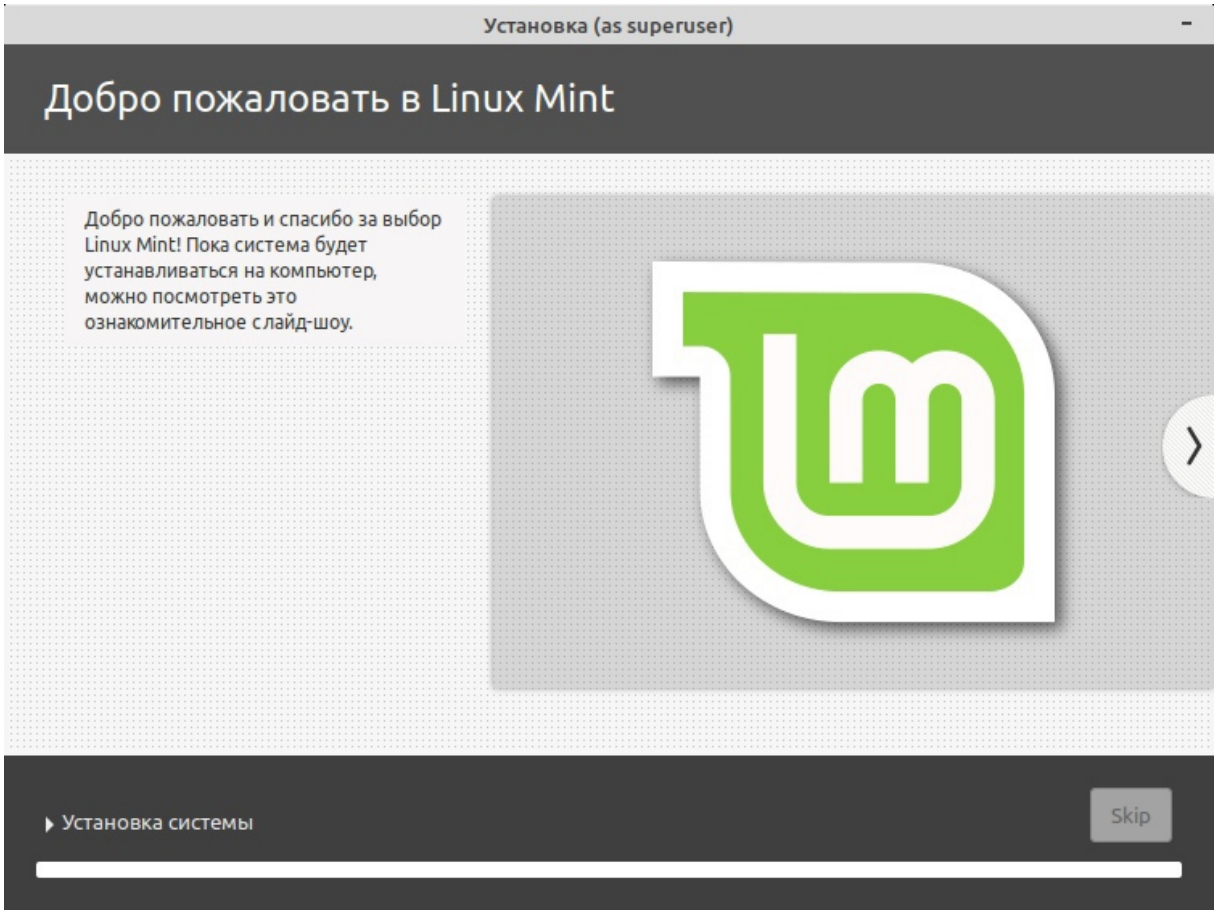


Рис. 9.13. Установка системы

После его завершения можно перезагрузить компьютер. При перезагрузке в меню запуска будет автоматически выбран *Linux*. Для входа в *Windows* нужно самостоятельно выбрать пункт *Windows Boot Manager* (название может быть другим).

Видео установки других дистрибутивов можно посмотреть по ссылкам:

– https://vk.com/video/@aer64?z=video7785282_456239349 – Linux Ubuntu;

– https://disk.yandex.ru/i/RMRUwc_SbbETPg – семейства Альт.

Настройка начального загрузчика

Одна из первых проблем, с которой сталкивается начинающий пользователь, – настройка начального загрузчика *Grub*.

После установки *Linux* на компьютер, с установленной ОС *Windows*, по умолчанию загружается *Linux*, и есть четыре варианта загрузки (количество вариантов может отличаться):

1. Загрузка ОС Linux.
2. Загрузка ОС Linux в безопасном режиме.
3. Тест оперативной памяти.
4. Загрузка Windows.

По умолчанию окно загрузчика видно 10 с, в течение которых нужно выбрать операционную систему для загрузки. Эти параметры можно изменить, они настраиваются в файле `/etc/default/grub`.

Файл *grub* относится к файлам, которые управляют работой операционной системы, поэтому редактировать его может только администратор.

Необходимо вызвать терминал командой *Пуск – Системные – Терминал*. В открывшемся окне вызвать следующие команды: `sudo nano /etc/default/grub` (вызван текстовый редактор для редактирования файла *grub*).

После ввода пароля откроется окно (рис. 9.14), в котором можно изменить следующие параметры:

– GRUB_DEFAUT – изменение порядка загрузки ОС (указывается номер строки в загрузчике). Важно заметить, что нумерация строк идёт с 0, поэтому если *Windows* в списке

загружаемых систем находится на 4 строке, то в параметре указывается 3.

– GRUB_TIMEOUT – изменение времени. По умолчанию пользователь видит загрузчик 10 с. Это можно изменить, указав здесь другое число (в секундах).

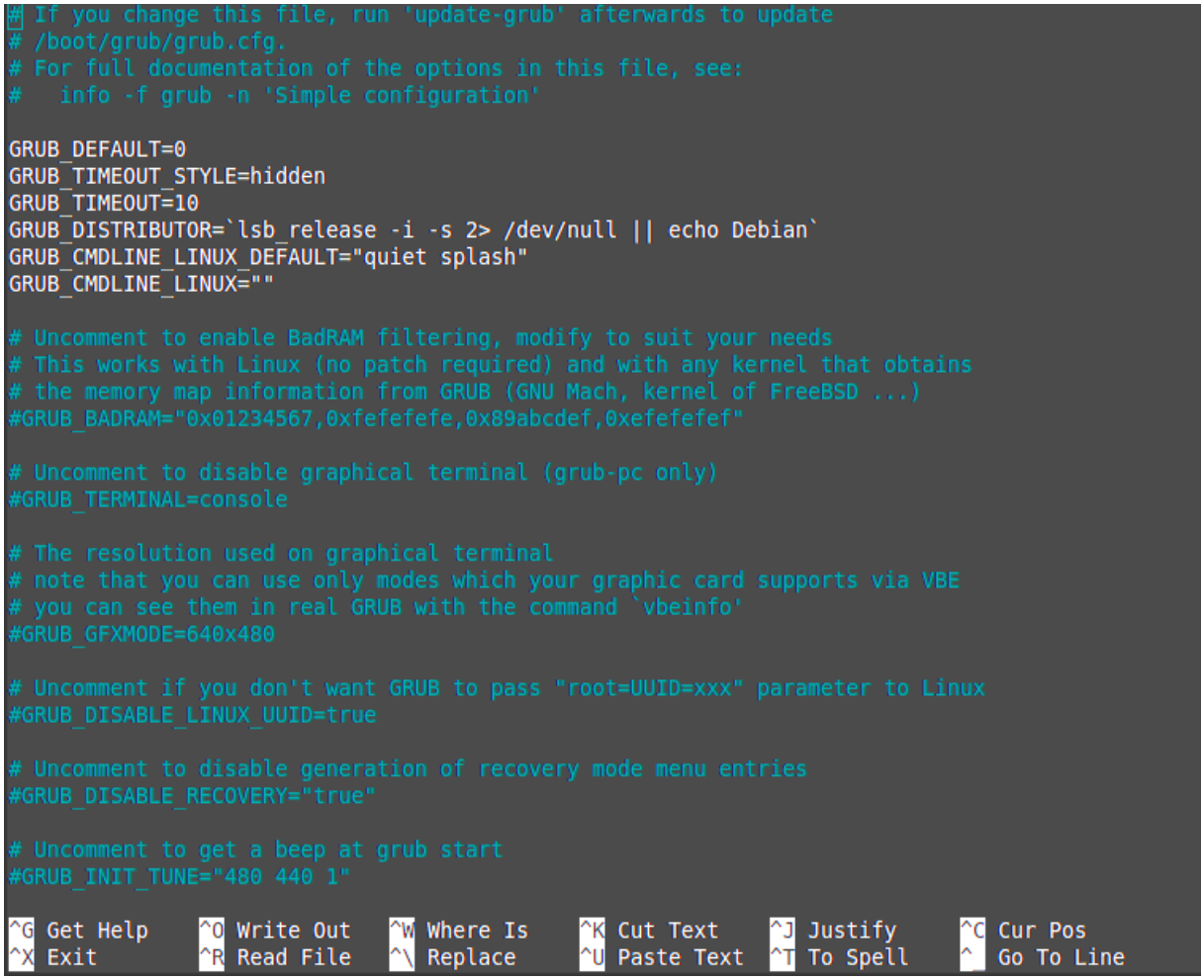


Рис. 9.14. Редактирование файла grub

После внесения изменений нажать *Ctrl+X* для выхода из редактора. После чего подтвердить сохранения, нажав клавиши *Y* и *Enter*. Затем обязательно вводим команду `sudo update-grub` для обновления загрузчика. После этого можно перезагружать компьютер.

10. ПРОГРАММИРОВАНИЕ ПОД УПРАВЛЕНИЕМ ОС СЕМЕЙСТВА LINUX

Приведём краткую справочную информацию об установке трансляторов с языков *Python, Julia, Free Pascal*.

Обычно интерпретатор с языка *Python3* устанавливается вместе с системой. Если вдруг по какой-то причине он не стоит, то установить его можно командой

```
# apt-get install python3
```

Установку *Julia* с официального сайта можно осуществить командой

```
$curl -fsSL https://install.julialang.org | sh
```

Компилятор с языка *Free Pascal* устанавливается командой

```
# apt-get install fpc
```

Для разработки на *Python* и *Julia* можно использовать *idle*[1] или *Jupyter Notebook (JupyterLab)*, для разработки на *Free Pascal – Geany*.

Рассмотрим установку приложений семейства *Jupyter* подробнее. При использовании *Python* версии до 3.9 включительно установка *Jupyter Notebook и JupyterLab* осуществляется командами

```
$ pip install notebook
$ pip install jupyterlab
```

Начиная с *Python 3.10* надо предварительно установить утилиту `pipx`, затем *Jupyter Notebook* и *JupyterLab* командами

```
# apt-get install pipx
$ pipx install notebook
$ pipx install jupyterlab
```

Далее рассмотрим возможности разработки программ под управлением *ОС семейства Linux* на языках программирования: *Fortran, C, C++*.

[1] Установка *idle* – `# apt-get install idle`

В качестве *IDE* для разработки на *Fortran, C/C++* авторы рекомендуют[1] использовать свободный текстовый редактор *Geany*[2] или свободную среду разработки *CodeBlocks*[3].

Компиляция программ на C, C++, Fortran

Установка компилятора языка *C* в различных операционных системах выполняется командой:

`#apt install gcc` – *Debian, Ubuntu;*

`#apt-get install gcc` – *ALT Linux.*

Установка компилятора языка *C++* осуществляется следующими командами:

`#apt install g++` – *Debian, Ubuntu;*

`#apt-get install gcc-c++` – *ALT Linux.*

Установка компилятора языка *Fortran* происходит так:

`#apt install gfortran` – *Debian, Ubuntu;*

`#apt-get install gcc-fortran` – *ALT Linux.*

Компиляция программ на *C* выполняется командой:

```
$ gcc file.c -o file -lm
```

здесь *file.c* – имя файла с кодом на языке *C*;

file – имя исполняемого файла (если ключ `-o` пропущен, то выходной файл имеет стандартно имя `a.out`);

`-lm` – ключ, необходимый для компиляции программ, использующих библиотеку `math.h`.

Компиляция программ на C++ выполняется командой:

```
$ g++ file.cpp -o file
```

здесь *file.cpp* – имя файла с кодом на языке *C++;*

file – имя исполняемого файла (если ключ `-o` пропущен, то выходной файл имеет стандартно имя `a.out`).

Компиляция программ на *Fortran* происходит так:

```
$ gfortran file.for -o file
```

---

[1] Рекомендация носит субъективный характер и не является обязательной для читателя.

[2] Установка *Geany* – `# apt-get install geany`

[3] Установка *CodeBlocks* – `# apt-get install codeblocks`

здесь *file.for* – имя файла с кодом на языке *Fortran file* – имя исполняемого файла (если ключ `-o` пропущен, то выходной файл имеет стандартно имя `a.out`).

При разработке программ с большим количеством вычислений компиляторы *gcc, g++, gfortran* позволяют оптимизировать программы по быстродействию. Для получения оптимизированных программ можно использовать ключи `-O0, -O1, -O2, -O3, -Os`:

– при использовании ключа **`-O0`** оптимизация отключена, достигается максимальная скорость компиляции;

– при использовании ключа «мягкой» оптимизации **`-O1`** происходит некоторое увеличение времени компиляции, этот ключ оптимизации позволяет одновременно уменьшать занимаемую программой память и уменьшить время выполнения программы;

– при использовании ключа **`-O2`** происходит существенное уменьшение времени работы программы, при этом не происходит увеличение памяти занимаемой программой, не происходит развертка циклов и автоматическое встраивание функций;

– ключ «агрессивной» оптимизации **`-O3`** нацелен в первую очередь на уменьшение времени выполнения программы, при этом может произойти увеличение объёма кода и времени компиляции, в этом случае происходит развертка циклов и автоматическое встраивание функций;

– ключ **`-Os`** ориентирован на оптимизацию размера программы, включаются те опции из набора **`-O2`**, которые обычно не увеличивают объём кода, применяются некоторые другие оптимизации, направленные на снижение его объёма.

Запуск программы на выполнение в терминале

```
$./file
```

здесь *file* – имя исполняемого файла.

Отладчик `gdb`

Большинство современных сред разработки имеют встроенный отладчик. В случае использования свободных компиляторов *gcc, g++, gfortran* отладчиком выступает `gdb`.

Для использования отладчика `gdb` программ код программы должен быть откомпилирован с ключом `-ggdb` (или `-g`).

```
$gcc file.c -g -o program
$g++ file.cpp -g -o program
$gfortran file.for -g -o program
```

После этого можно запустить команду

```
$gdb program
```

здесь *program* – имя исполняемого файла, откомпилированного с ключом `-ggdb`.

Далее перечислены основные команды отладчика:

`run` запускает программы в отладочном режиме;

`break` определяет точку останова в программе, синтаксис команды – `break n` или `break file:n`, где *n* – номер строки, *file* – имя файла;

`inspect var` выводит значение переменной *var*;

`next` выполняет следующую команду без захода в подпрограмму;

`step` выполняет следующую команду с заходом в подпрограмму;

`list` выводит несколько строк до и после текущей;

`until n` выполняет программу до строки с номером *n*;

`cont` продолжает прерванное выполнение программы;

`call` позволяет вызвать любую функцию;

`help` выводит справку по отладчику `gdb`;

`quit` или *Ctrl+D* завершает работу отладчика.

Рассмотрим отладку простой программы на C++[1] сортировки массива (файл *otladka_max.cpp*):

```
#include <iostream>
#include <math.h>
int main ( int argc , char ** argv )
{int i , n , j , k , nmax ;
double *a ;
cout<<"n=" ;
cin >>n ;
```

[1] Для кодов на *C* или *Fortran* отличия будут только в используемом компиляторе.

```
    a=new double [ n ] ;
    cout<<"Введите массив "<<endl;
    for ( i =0; i <n ; i ++)
        cin >>a[i];
    k=n ;
    for ( j =0; j<=k ; )
        {for (nmax=0, i =1; i <k ; i ++)
            if (a[ i ]>a[nmax]) nmax=i ;
            swap (a[nmax], a[k-1]);
            k--;}
    cout<<"Отсортированный массив "<<endl;
    for ( i =0; i <n ; i ++)
        cout<<a [ i ]<<" " ;
    cout<<endl;
    return 0 ;}
```

Произведем компиляцию программы с ключом -ggdb и запустим программу:

```
$ g++ otladka_max.cpp -o otladka_max -ggdb
$ gdb otladka_max
```

Далее приведён пример работы с отладчиком с комментариями

```
    GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
    Copyright (C) 2014 Free Software Foundation, Inc.
    License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
    This is free software: you are free to change and
redistribute it.
    There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
    This GDB was configured as "x86_64-linux-gnu".
    Type "show configuration" for configuration details.
    For bug reporting instructions, please see:
    <http://www.gnu.org/software/gdb/bugs/>.
    Find the GDB manual and other documentation resources
online at:
    <http://www.gnu.org/software/gdb/documentation/>.
    For help, type "help".
    Type "apropos word" to search for commands related to
"word"...
    Reading symbols from otladka_max...done.
    Reading symbols from otladka_max...done.
```

(gdb) break 10 //Устанавливаем точку останова в 10 строке

```
Breakpoint 1 at 0x4009fc: file otladka_max.cpp, line
10.
```
(gdb) break 15 //Устанавливаем точку останова в 15 строке
```
Breakpoint 2 at 0x400a81: file otladka_max.cpp, line
15.
```
(gdb) break 20 //Устанавливаем точку останова в 20 строке
```
Breakpoint 3 at 0x400b18: file otladka_max.cpp, line
20.
```
(gdb) break 21 //Устанавливаем точку останова в 21 строке
```
Breakpoint 4 at 0x400b28: file otladka_max.cpp, line
21.
```
(gdb) run //Запуск программы в отладочном режиме
```
Starting program: /home/user/cpp/otladka_max
```
n=10
```
Breakpoint 1, main (argc=1, argv=0x7fffffffe4a8) at
otladka_max.cpp:10
10 a=new double[n];
```
(gdb) inspect n //Вывод значения переменной n
```
$1 = 10
```
(gdb) cont //Продолжение выполнения программы (до точки остановки)
```
Continuing.
'Введите массив'
1 9 0 8 2 7 6 5 4 3
Breakpoint 2, main (argc=1, argv=0x7fffffffe4a8) at
otladka_max.cpp:15
15 for(j=0;j<=k;)
```
(gdb) inspect k //Вывод значения переменной k
```
$2 = 10
```
(gdb) cont // Продолжение выполнения программы (до точки остановки)
```
Continuing.
Breakpoint 3, main (argc=1, argv=0x7fffffffe4a8) at
otladka_max.cpp:20
20 k--;
(gdb) inspect k //Вывод значения переменной k
$3 = 10
(gdb) cont
Continuing.
Breakpoint 3, main (argc=1, argv=0x7fffffffe4a8) at
otladka_max.cpp:20
20 k--;
```
(gdb) inspect k //Вывод значения переменной k
```
$4 = 9
```
(gdb) inspect a[1] //Вывод значения элемента массива a – a_1

```
$5 = 3
(gdb) inspect a[9] //Вывод значения элемента массива a – a9
$6 = 9
(gdb) inspect a[2] //Вывод значения элемента массива a – a2
$7 = 0
(gdb)
```

Отладчики всех современных сред разработки представляют собой графический интерфейс для доступа к консольному отладчику `gdb`.

Разработка параллельных программ на C/C++ и Fortran

Установка библиотеки *MPI* производится с помощью команды

```
# apt-get install openmpi-bin libopenmpi-dev
```

Компиляция программ, разработанных с помощью библиотеки *MPI:*

`$ mpicc file.c -o file` – язык C;

`$ mpicxx file.cxx -o file` – язык C++;

`$ mpif90 file.for -o file` – язык Fortran.

Запуск скомпилированного параллельного *MPI*-приложения

```
mpirun -np K ./file
```

здесь *K* – количество процессов.

Компиляция программ, разработанных с помощью библиотеки *OpenMP* на языках C, C++, Fortran.

```
$ gcc -fopenmp file.c -o file
$ g++ -fopenmp file.cpp -o name
$ gfortran -fopenmp Name.cpp -o name
```

Сборка библиотек

Библиотека – набор функций и других объектов языка, используемых неоднократно другими программами. Библиотеки компилируются независимо от вызывающей их программы и хранятся в виде объектных файлов.

Существуют два вида библиотек – *статические* и *динамические*.

При компиляции программы код всех функций из статической библиотеки помещается в состав исполняемого файла.

Код динамических библиотек не входит в исполняемый файл, последний содержит лишь ссылку на библиотеку. Если динамическая библиотека будет удалена или перемещена в другое место, то программа работать не будет. Динамическая библиотека подгружается в оперативную память только в момент вызова функций из её состава.

Далее будет рассмотрено создание статических и динамических библиотек с помощью компиляторов *gcc, g++, gfortran*, а также три варианта разработки приложения на языке *C++*[1], исходные тексты программ которого хранятся в нескольких файлах.

Сборка программы из нескольких файлов

Создадим работающее приложение из файлов *integral1.h, integral1.cpp, integral2.h, integral2.cpp, integralmain.cpp.* Здесь основная программа хранится в файле *integralmain.cpp*, необходимые функции – в файлах *integral1.cpp* (заголовочный файл – *integral1.h*), *integral2.cpp* (заголовочный файл – *integral2.h*).

Классическим способом сборки проекта из нескольких файлов являются несколько шагов компиляции.

Шаг 1

`g++ -c integral1.cpp` – компиляция файла *integral1.cpp*[2].

Шаг 2

`g++ -c integral2.cpp` – компиляция файла *integral2.cpp*.

Шаг 3

`g++ integralmain.cpp integral1.o integral2.o -o primer` – компиляция основного приложения – файл *integralmain.cpp* (исполняемый файл имеет имя *primer*).

---

[1] Здесь и далее сборка на *C* или *Fortran* будет отличаться только именами файлов и именами компиляторов.

[2] Откомпилированные (объектные файлы) в unix-подобных системах имеют расширение `.o`.

Запуск исполняемого файла *primer* производится командой `./primer`

Вместо трех шагов сборки можно выполнить команду `g++ integralmain.cpp integral1.cpp integral2.cpp -o primer1`

Однако следует помнить, что в этом случае происходит автоматическая перекомпиляция всех файлов независимо от того, нужна она или нет.

Обратите внимание, что в именах файлов при использовании компиляторов *gcc, g++* и *gfortran* можно применить шаблоны имён (символы подстановки **, ?*). Например, написать

```
g++ integral*.cpp -o primer
```

Сборка статической библиотеки

Наличие механизмов раздельной компиляции делает возможным повторное использование кода разработанных функций. Скомпилированные файлы (файлы с объектным кодом) собираются в библиотеку. Библиотека является одним файлом, в котором содержится код всех откомпилированных файлов. В библиотеке также содержится некий индекс, позволяющий ускорять процесс сборки по сравнению с обычной сборкой из объектных файлов.

В качестве примера рассмотрим статическую библиотеку. Создадим небольшую библиотеку, в которую включим функции из файлов *integral1.h, integral1.cpp и integral2.h, integral2.cpp.*

Откомпилируем файлы, включаемые в библиотеку стандартным образом:

```
g++ -c integral1.cpp
g++ -c integral2.cpp
```

Воспользуемся командой `ar`, которая применяется для создания и изменения библиотечных файлов.

```
ar crs libintegral.a ./integral1.o ./integral2.o
```

Рассмотрим параметры этой команды подробно:

– ключи команды `ar: c` – создать библиотеку; `r` – заменить совпадающие объектные файлы внутри библиотеки новыми; `s` – создать индекс внутри библиотеки;

– libintegral.a – определяет имя создаваемой библиотеки, имя файла всегда должно начинаться с lib и иметь расширение .a, в нашем примере мы создаём библиотеку integral, которая будет храниться в файле libintegral.a;

– ./integral1.o ./integral2.o – имена объектных файлов, помещаемых в библиотеку.

Осталось собрать исполняемый файл, подключив в него функции из статической библиотеки. Это можно сделать с помощью команды:

```
g++ ./integralmain.cpp -L. -lintegral -o integralmain_static
```

Ключ -L определяет месторасположение библиотеки, а ключ -l – её имя. Порядок следования ключей важен.

Особенности разработки динамических библиотек

Создание динамических библиотек имеет ряд особенностей. На первом этапе необходимо откомпилировать файлы, которые служат основой для создаваемой библиотеки с ключом -fpic или -fPIC. Ключ -fpic создаёт более компактный объектный файл, но поддерживается не всеми компиляторами. Ключи -fpic и -fPIC позволяют создавать объектный код, который может использоваться разными программами во время выполнения. В нашем примере создания динамической библиотеки численного интегрирования команды компиляции могут быть такими:

```
g++ -c -fpic integral1.cpp
g++ -c -fpic integral2.cpp
```

Рассмотрим, как из откомпилированных с ключом -fpic файлов создать динамические библиотеки. Как и статические, файлы динамических библиотек должны иметь префикс lib. Однако расширение файлов динамических библиотек – .so (*shared object*).

Следует помнить, что для одной библиотеки будет использоваться четыре имени:

1) *«реальное имя»* – имя файла на диске, содержащего код библиотеки, должно иметь форму libимя.so.n.m.k, где n – номер версии библиотеки, m – номер промежуточной версии, k –

это номер релиза промежуточной версии библиотеки; а также последнюю часть реального имени – номер релиза с предшествующей точкой можно опускать;

2) «*so-имя*» (*soname*) имеет вид `libимя.so.n`. В одном каталоге с реальным именем должна существовать символьная ссылка на него, имеющая so-имя этой библиотеки;

3) *имя, используемое компоновщиком при создании исполняемого файла* – `libимя.so`. В каталоге с файлом библиотеки создаётся символьная ссылка с этим именем на so-имя либо на реальное имя;

4) *имя библиотеки для использования при сборке исполняемого файла*.

Рассмотрим пошагово все этапы создания динамической библиотеки численного интегрирования.

Шаг 1. Компилируем с ключом `-fpic` файлы *integral1.cpp* и *integral2.cpp.* В результате формируются файлы *integral1.o, integral2.o.*

```
$g++ -c -fpic integral1.cpp
$g++ -c -fpic integral2.cpp
$ ls
integral1.cpp integral1.o
integral1.h
integral2.cpp
integral2.h
integral2.o
```

Шаг 2. Создаём файл библиотеки *libintegral.so.1.0* из откомпилированных с ключом `-fpic` файлов *integral1.o* и *integral2.o.*

```
$ g++ -shared -Wl,-soname,libintegral.so.1 -o
libintegral.so.1.0 ./integral1.o ./integral2.o
$ ls
integral1.cpp integral1.o
integral2.h libintegral.so.1.0
integral1.h
integral2.cpp integral2.o
```

Шаг 3. Создаем каталог для хранения динамической библиотеки.

```
$ mkdir ~/mylib
```

Шаг 4. Скопируем файл *libintegral.so.1.0* в каталог, в котором будет храниться динамическая библиотека численного интегрирования.

```
$cp ./libintegral.so.1.0 ~/mylib/libintegral.so.1.0
$ls ~/mylib
libintegral.so.1.0
```

Шаг 5. Создадим so-файлы, для этого выполним следующую команду[1]:

```
$ ldconfig -n /home/aer/mylib
```

Шаг 6. Создадим файл *libintegral.so,* как символьную ссылку.

```
$ ln -s /home/aer/mylib/libintegral.so.1
/home/aer/mylib/libintegral.so
$ ls ~/mylib
libintegral.so libintegral.so.1 libintegral.so.1.0
```

Шаг 7. Компилируем программу с использованием динамической библиотеки

```
$ g++ ./integralmain.cpp -L/home/aer/mylib
-lintegral -o integralmain-dinamic
```

Созданный исполняемый файл сразу запустить не удастся.

Перед запуском следует модифицировать переменную окружения *LD_LIBRARY_PATH*, в которой хранятся пути для поиска библиотек. Посмотрим, есть ли такая переменная среды:

```
echo $LD_LIBRARY_PATH
```

Если в ответ выводится пустая строка, означающая, что такой переменной среды нет, её можно установить следующей командой:

```
$ LD_LIBRARY_PATH=/home/aer/mylib
```

или командой

```
$ export LD_LIBRARY_PATH=/home/aer/mylib
```

Если же команда

```
echo $LD_LIBRARY_PATH
```

выдает непустую строку в ответ, это означает, что переменная *LD_LIBRARY_PATH* существует, и путь к библиотеке надо добавить в переменную *LD_LIBRARY_PATH.*

[1] Здесь и далее aer – имя пользователя, его домашний каталог `/home/aer`.

Это сделать можно так:

```
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/aer/mylib
```

Если путь к динамической библиотеке нужен не только для этого сеанса работы, а постоянно, то команды

```
$ export LD_LIBRARY_PATH=/home/aer/mylib
```

или

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:
/home/aer/mylib
```

имеет смысл дописать в файл `/home/aer/.bashrc`.

После того как определили значение переменной *LD_LIBRARY_PATH*, можно запускать созданный с помощью динамической библиотеки исполняемый файл.

```
./integralmain-dinamic
```

Рассмотренный механизм создания и хранения динамических библиотек в личных каталогах пользователя удобен при отладке подобных библиотек. Готовые к использованию динамические библиотеки можно поместить в каталог `/usr/lib`. Это делается аналогично, но с учетом прав доступа к каталогу `/usr/lib`. Пример создания динамической библиотеки численного интегрирования в каталоге `/usr/lib` приведён далее (подразумевается, что файлы с исходными кодами находятся в текущем каталоге).

```
$ g++ -fpic -c ./integral1.cpp
$ g++ -fpic -c ./integral2.cpp
$ ld -shared -soname libintegral.so.1 -o libintegral.so.1.0
./integral1.o
./integral2.o
$ sudo cp ./libintegral.so.1.0 /usr/lib/libintegral.so.1.0
$ sudo ldconfig -n /usr/lib/
$      sudo      ln      -s      /usr/lib/libintegral.so.1
/usr/lib/libintegral.so
$ g++ ./integralmain.cpp -lintegral -o integralmain-dinamic
$ ./integral-dinamic
```

Автоматизация сборки библиотек с помощью утилиты `make`

Современные программы для решения вычислительных задач могут состоять из множества отдельно компилируемых

файлов. Если в состав приложения входит несколько файлов, то сборку можно осуществить вручную, как описывалось ранее. Для сборки сложных программных комплексов, включающих в себя большое количество исходных файлов, лучше воспользоваться автоматической сборкой с помощью утилиты *make*.

Утилита *make* в простейшем случае работает следующим образом: последовательно начинают выполняться команды сборки в соответствии с правилами, которые записаны в файле с именем *MakeFile* из текущего каталога. Далее используется введённый одним из авторов русскоязычный термин «*файл с правилами сборки*», соответствующий русско-английскому термину *make-файл*, который используется в среде программистов.

Если правила сборки хранятся в файле с другим именем, то утилита *make* запускается с ключом `-f filename, filename` – имя файла с правилами сборки.

Опишем процесс сборки на примере ранее представленной задачи.

Рассмотрим некоторые понятия файла с правилами сборки.

Правило сборки – одна или несколько команд, которые выполняют некую задачу, формируют целевой файл (цель). *Цель сборки* – имя файла, сформированного в результате выполнения правил сборки.

Чаще всего в качестве цели сборки указывается исполняемый или объектный файл.

Приведем пример автоматического создания исполняемого файла из файлов *integral1.cpp, integral1.h, integral2.cpp, integral2.h, integralmain.cpp.* Вручную сборка осуществляется следующими командами:

```
g++ -c integral1.cpp
g++ -c integral2.cpp
g++ integralmain.cpp integral1.o integral2.o -o integralmain
```

Рассмотрим пример простейшего файла с правилами сборки[1]. Содержимое *MakeFile*:

[1] Комментарии начинаются с символа #.

```
integral1.o: integral1.cpp integral1.h
     g++ -c integral1.cpp
integral2.o: integral2.cpp integral2.h
     g++ -c integral2.cpp
integralmain: integralmain.cpp integral1.o integral2.o
     g++ integralmain.cpp integral1.o integral2.o -o integralmain
```

Опишем более подробно этот файл. Файл состоит из шести строк, в данном файле – три цели, каждая из целей занимает минимум две строчки, первая строчка – заголовок цели. Заголовок цели всегда занимает одну строчку и начинается с первой позиции. Первый параметр заголовка – имя формируемого файла, затем символ «:», потом имена файлов, от которых зависит построение формируемого файла. Вторая и последующие строки цели содержат список команд, которые необходимы для формирования цели. Все строки, содержащие команды, необходимые для формирования файла цели, должны начинаться с символа табуляции. Признаком конца списка команд служит строка, которая не начинается с табуляции.

Первой целью в рассматриваемом нами файле является объектный файл *integral1.o.* Для формирования этого файла необходимы файлы *integral1.cpp* и *integral1.h.* Файл *integral1.o* формируется с помощью команды

```
g++ -c integral1.cpp.
```

Вторая цель – объектный файл *integral2.o,* формирование которого аналогично.

Третьей и итоговой целью файла с правилами сборки является исполняемый файл *integralmain*, построение которого зависит от файлов *integralmain.cpp, integral1.o, integral2.o* и осуществляется с помощью команды

```
g++ integralmain.cpp integral1.o integral2.o -o integralmain
```

Для запуска сборки с помощью файла сборки служит команда

```
make -f файл цель
```

Здесь `файл` – имя файла с правилами сборки; `цель` – итоговая цель, формируемая с помощью файла с правилами сборки. Если этот файл имеет имя *MakeFile,* то параметр `-f файл` можно опустить. Следовательно, в нашем случае команда сборки может быть такой

```
make integralmain
```

или такой

```
make -f MakeFile integralmain
```

В результате выполнения этой команды будет сформирован исполняемый файл. В отличие от ручной сборки при автоматизированной сборке с помощью утилиты make не выполняются команды, для которых цели уже сформированы и не требуют обновлений.

Использование make-переменных

Если необходимо компилировать файлы с ключами `-g -Wall -O0`, то можно просто добавить ключи в команды файла с правилами сборки. Другим подходом является определение специальных переменных (make-переменных). Введём такие переменные, в которых определим используемый компилятор и ключи, применяемые при компиляции. В этом случае файл с правилами сборки *makefile1* может быть таким:

```
#Переменная CCC с именем компилятора
CCC=g++
#Переменная FLAG с ключами компиляции
FLAG=-g -Wall -O0
integral1.o: integral1.cpp integral1.h
	$(CCC) $(FLAG) -c integral1.cpp
integral2.o: integral2.cpp integral2.h
	$(CCC) $(FLAG) -c integral2.cpp
integralmain: integralmain.cpp integral1.o integral2.o
	$(CCC) $(FLAG) integralmain.cpp integral1.o integral2.o -o integralmain
```

Команда сборки в этом случаем будет стандартной:

```
make -f makefile1 integralmain
```

Можно изменять значение переменных *CCC* и *FLAG* в файле *makefile1* и пересобирать исполняемый файл с другими характеристиками. Ещё одной интересной особенностью является возможность изменить значение make-переменных при запуске сборки из командной строки. Например, если необходимо использовать компилятор `g++-13` и флаги `-Wall -O3`, то это можно сделать двумя способами:

1) изменить содержимое файла *makefile1*;

2) запустить сборку с помощью команды

```
make -f makefile1 integralmain CCC='g++-13' FLAG='-Wall -O3'
```

В файлах с правилами сборки можно использовать предопределённые переменные.

Использование предопределённых переменных

В файлах с правилами сборки можно использовать предопределённые переменные. Для более детального знакомства можно обратиться к оригинальной документации по ссылке https://www.gnu.org/software/make/manual/make.html. В сети представлено достаточное количество описаний утилиты *make* на русском языке. Здесь рассмотрены только те переменные, которые необходимы при сборке вычислительных программ.

CC – переменная, в которой хранится компилятор с языка C;

CXX – переменная, в которой хранится компилятор с языка C++;

CF – переменная, в которой хранится имя компилятора языка Фортран;

CFLAGS – переменная, в которой хранятся ключи компилятора языка C;

CXXFLAGS – переменная, в которой хранятся ключи компилятора C++;

CPPFLAGS – переменная, в которой хранятся ключи компиляторов C и Фортран;

FFLAGS – переменная, в которой хранятся ключи компилятора Фортран.

Значения переменных CC, CXX, CF определены, в них хранятся значения соответствующих компиляторов вашей системы. Значения же переменных CFLAGS, CXXFLAGS, CPPFLAGS, FFLAGS пусты. Внутри файла с правилами сборки можно переопределять значения всех предопределённых переменных. Однако их нельзя переопределить при запуске утилиты *make* из командной строки.

Псевдопеременные в файлах с правилами сборки

В файлах с правилами сборки могут быть использованы так называемые псевдопеременные:

– $@ – имя (файл) цели;
– **$<** – имя первого файла из списка зависимостей;
– **$^** – список всех файлов из списка зависимостей.

Последний файл сборки (*makefile1*) можно переписать с применением *make-переменных* и *псевдоопределённых переменных*.

```
CXX=g++-13
CXXFLAGS=-c
integral1.o: integral1.cpp integral1.h
     $(CXX) $(CXXFLAGS) $<
integral2.o: integral2.cpp integral2.h
     $(CXX) $(CXXFLAGS) $<
integralmain: integralmain.cpp integral1.o integral2.o
     $(CXX) $^ -o $@
```

Обобщённые цели

Первые две цели в нашем файле с правилами сборки отличаются только файлами, которые компилируются. Поэтому заголовок обобщённой цели можно записать

```
%.o: %.cpp %.h
```

Тогда вместо двух (или в других случаях нескольких) правил для компиляции файлов получается одно обобщённое правило

```
%.o: %.cpp %.h
     $(CXX) $(CXXFLAGS) $<
```

Однако в этом случае необходимо определить как минимум имена исходных (*integral1.cpp, integral2.cpp*) и объектных (*integral1.o, integral2.o*) файлов[1]. Это можно сделать с помощью следующих двух команд:

```
SRCMODULES=integral1.cpp integral2.cpp
OBJMODULES=$(SRCMODULES:.cpp=.o)
```

Файл с правилами можно переписать так:

[1] Обратите внимание, что файлы с расширением .h нигде в правилах сборки не используются. Они должны быть в каталоге, где осуществляется сборка, но нет никакой необходимости включать эти файлы в правила.

```
CC=g++-13
SRCMODULES=integral1.cpp integral2.cpp
OBJMODULES=$(SRCMODULES:.cpp=.o)
%.o: %.cpp %.h
    $(CC) -c $(CFLAGS) $<
integralmain: integralmain.cpp $(OBJMODULES)
    $(CC) $^ -o $@
```

Получилась довольно универсальная конструкция. Изменяя значения переменной *SRCMODULES*, можно подключать любые исходные файлы, в которых хранятся необходимые функции.

Псевдоцели

В качестве целей могут выступать не только исполняемые файлы, но и другие действия. Дополним наш файл целью *start*, предназначенной для запуска собранного файла, и целью *deleteobj*, предназначенную для удаления файлов с расширением `.o`.

```
CC=g++-13
SRCMODULES=integral1.cpp integral2.cpp
OBJMODULES=$(SRCMODULES:.cpp=.o)
%.o: %.cpp %.h
    $(CC) -c $(CFLAGS) $<
integralmain: integralmain.cpp $(OBJMODULES)
    $(CC) $^ -o $@
start : integralmain
    ./integralmain
deleteobj :
    rm -f *.o
```

В этом случае команда `make` может собирать исполняемый файл, запускать собранный файл на выполнение и удалять `.o` при необходимости. Возможны следующие варианты использования файла с правилами сборки:

`make -f makefile integralmain` – сборка исполняемого файла *integralmain*;

`make -f makefile integralmain start` – сборка исполняемого файла *integralmain* и его запуск;

`make -f makefile integralmain start deleteobj` – сборка исполняемого файла *integralmain*, его запуск, после этого удаляются объектные файлы, которые были созданы на этапе сборки;

`make -f makefile integralmain deleteobj` – сборка исполняемого файла *integralmain* и удаление объектных файлов, которые были созданы на этапе сборки;

`make -f makefile deleteobj` – удаление объектных файлов, которые были созданы на этапе сборки;

`make -f makefile start` – запуск исполняемого файла *integralmain*, если файла нет, то он будет создан.

11.ЛАБОРАТОРНЫЙ ПРАКТИКУМ «ОСНОВЫ РАБОТЫ В LINUX»

Лабораторная работа 1

Установить ОС семейства *Linux* (предпочтительно отечественные *ALT Linux, Green Linux* либо *Debian, Ubuntu, Linux Mint*) на свой ноутбук (стационарный компьютер).

Лабораторная работа 2

Установить пакеты полной русификации `man` (deb-пакет русификации можно получить по ссылке https://disk.yandex.ru/d/6SbKnoytSzYqRA).

Получить справочную информацию о командах и перенаправить ее в файл *lab2.txt:*

1) `whatis, touch;`
2) `apropos, cat;`
3) `cp, file;`
4) `mv, adduser;`
5) `which, find;`
6) `tac, id;`
7) `rm, useradd;`
8) `less, chmod;`
9) `mkdir, chown;`
10) `dpkg, chgrp;`
11) `ls, su;`
12) `rmdir, cal;`
13) `apt, locate;`
14) `sudo, info;`
15) `pwd, echo.`

Лабораторная работа 3

1. Создать каталог *Var(номер варианта)*.

2. В каталоге создать дерево каталогов, как показано на рис. 10.1–10.15.

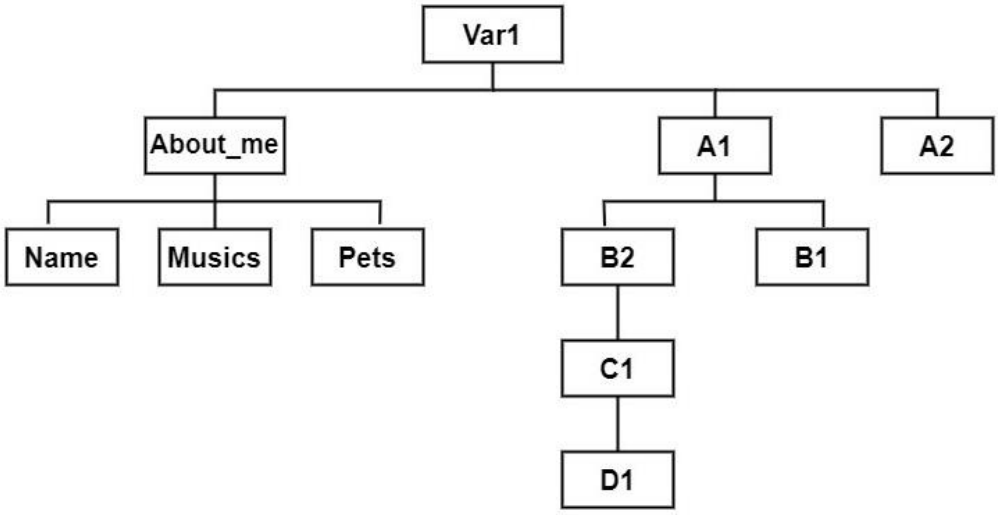


Рис. 10.1. Вариант 1

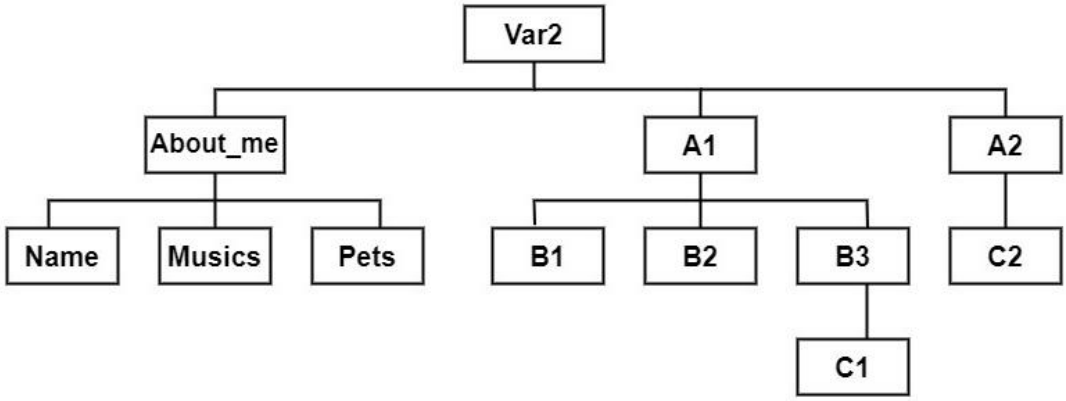


Рис. 10.2. Вариант 2

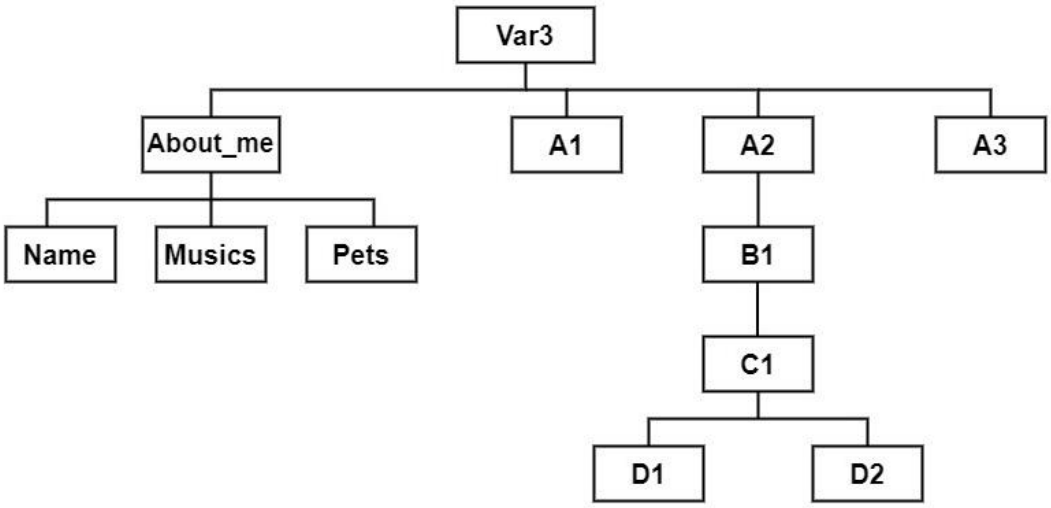


Рис. 10.3. Вариант 3

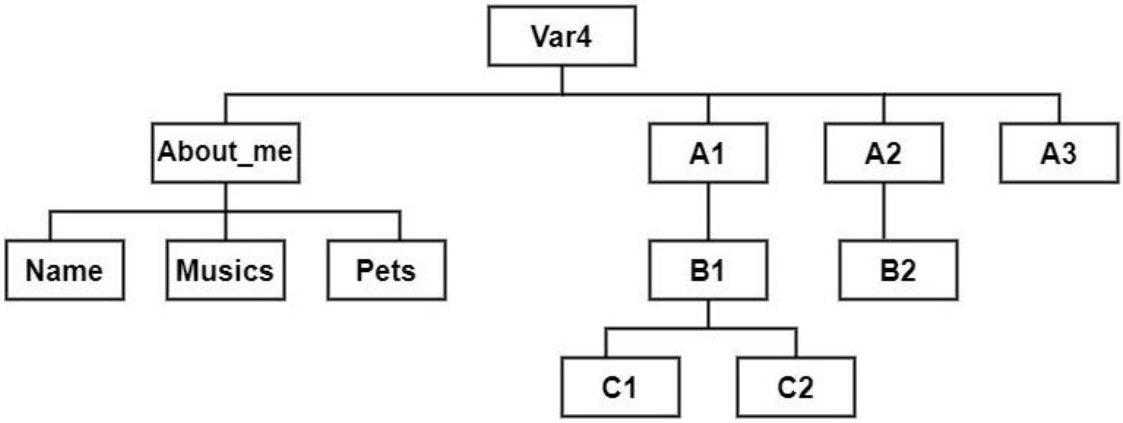


Рис. 10.4. Вариант 4

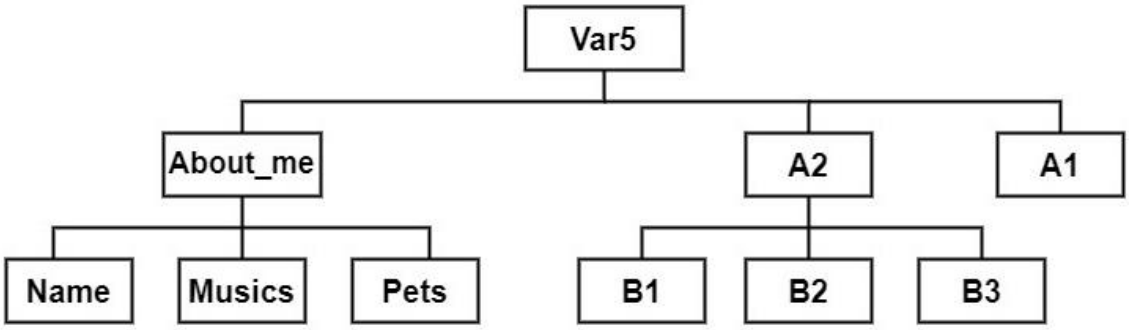


Рис. 10.5. Вариант 5

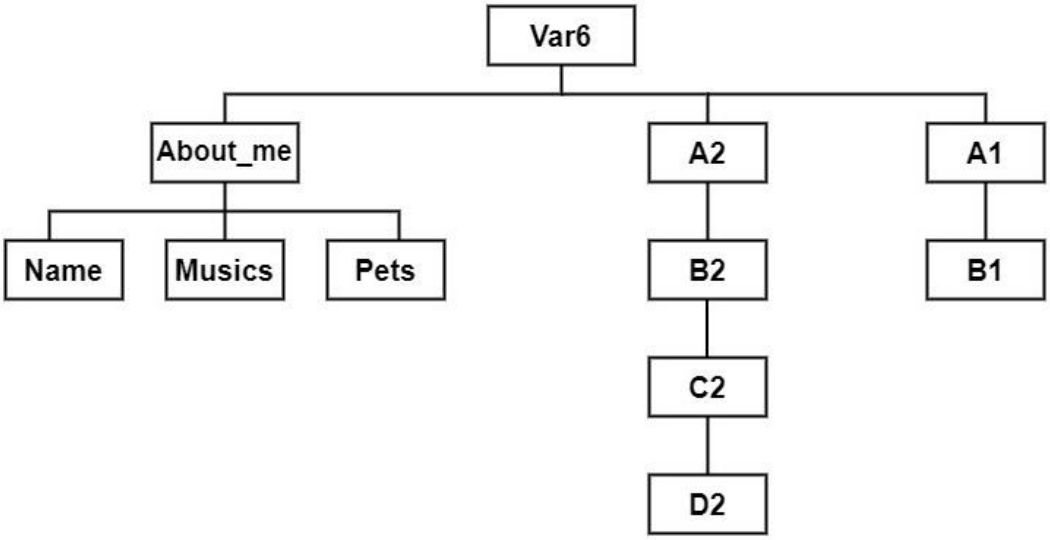


Рис. 10.6. Вариант 6

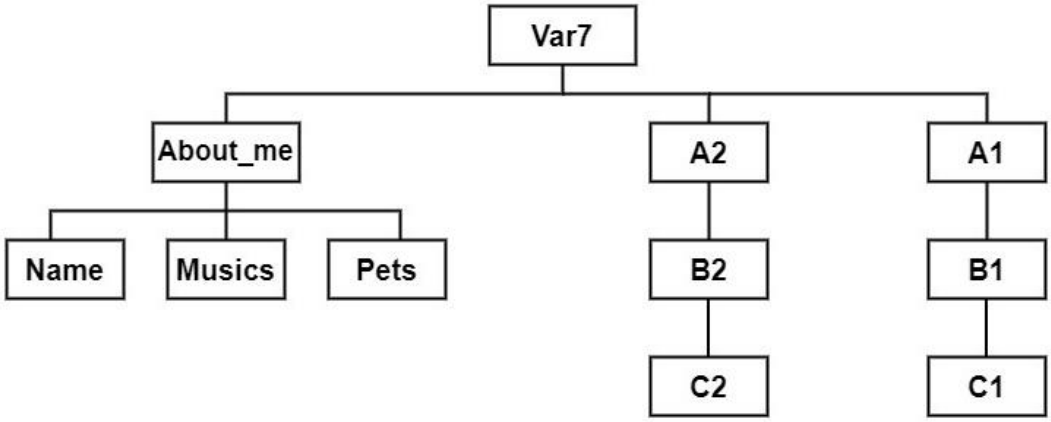


Рис. 10.7. Вариант 7

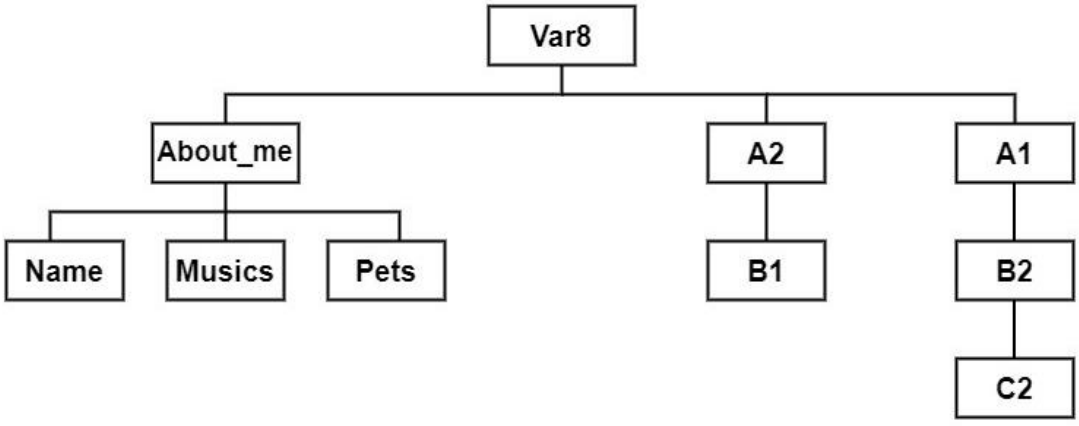


Рис. 10.8. Вариант 8

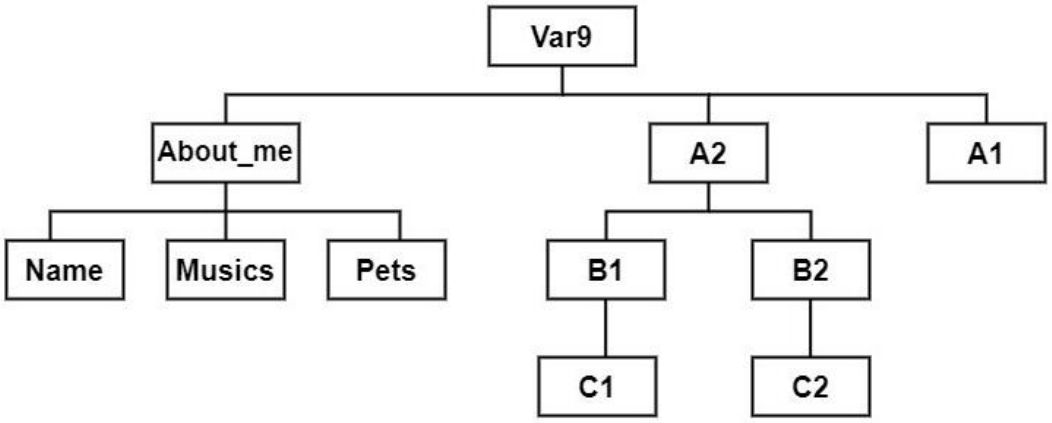


Рис. 10.9. Вариант 9

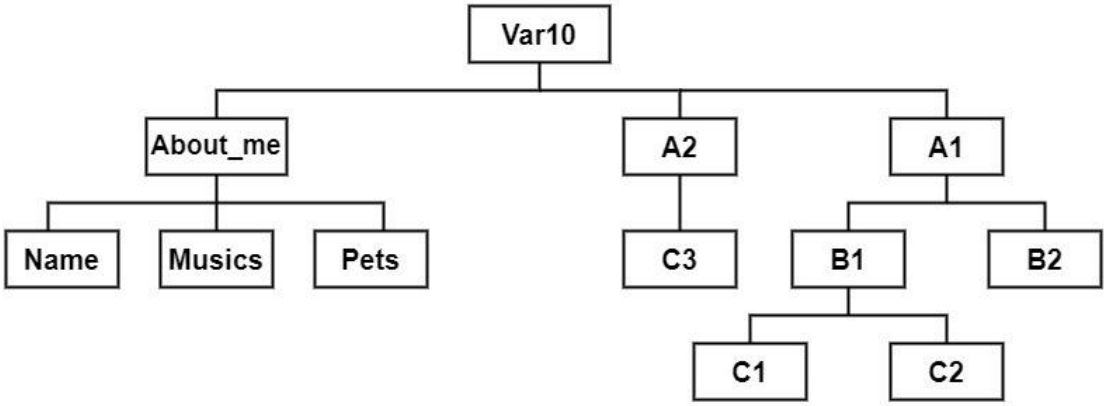


Рис. 10.10. Вариант 10

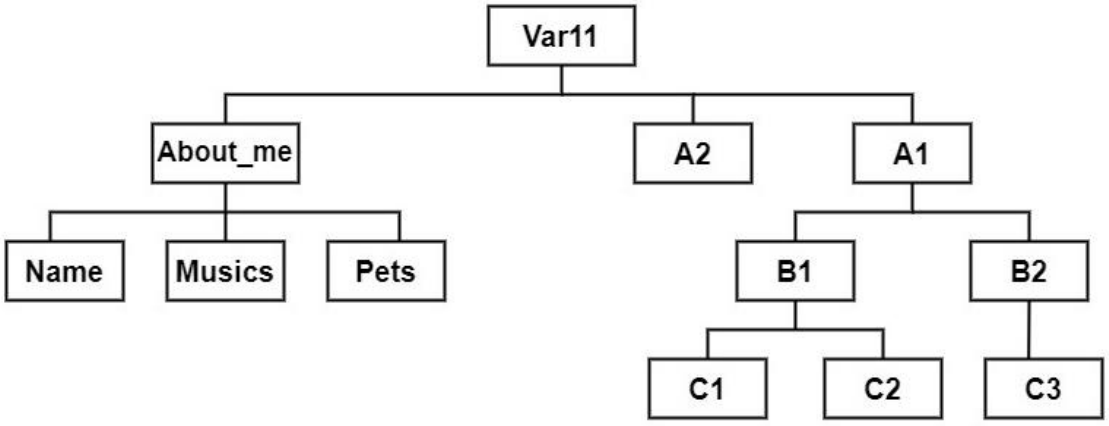


Рис. 10.11. Вариант 11

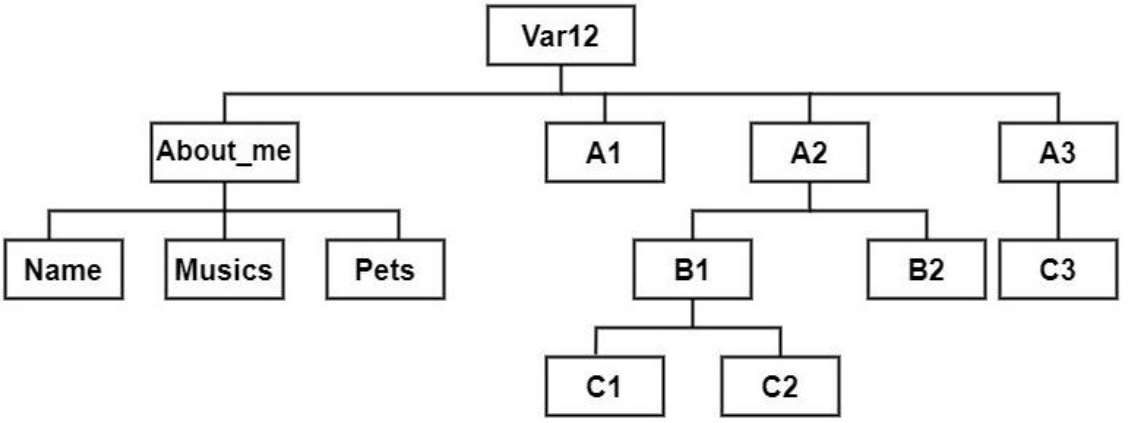


Рис. 10.12. Вариант 12

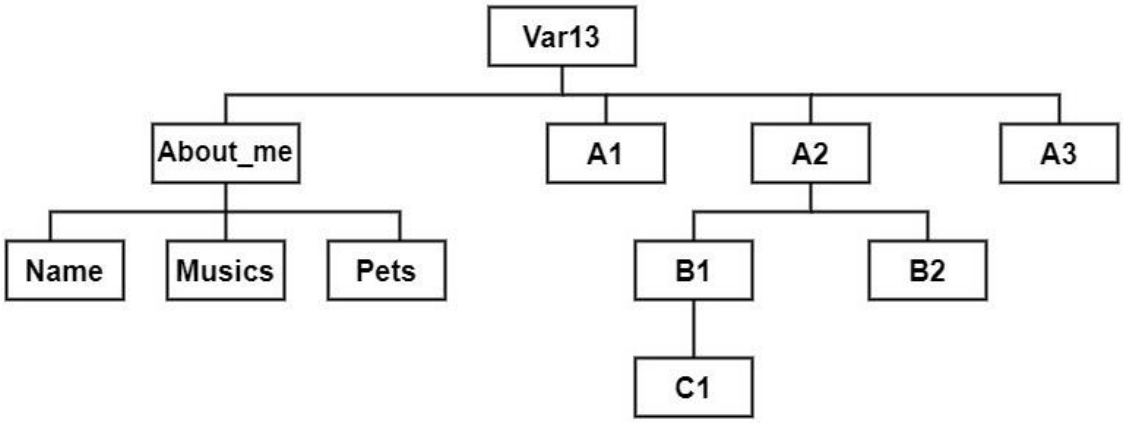


Рис. 10.13. Вариант 13

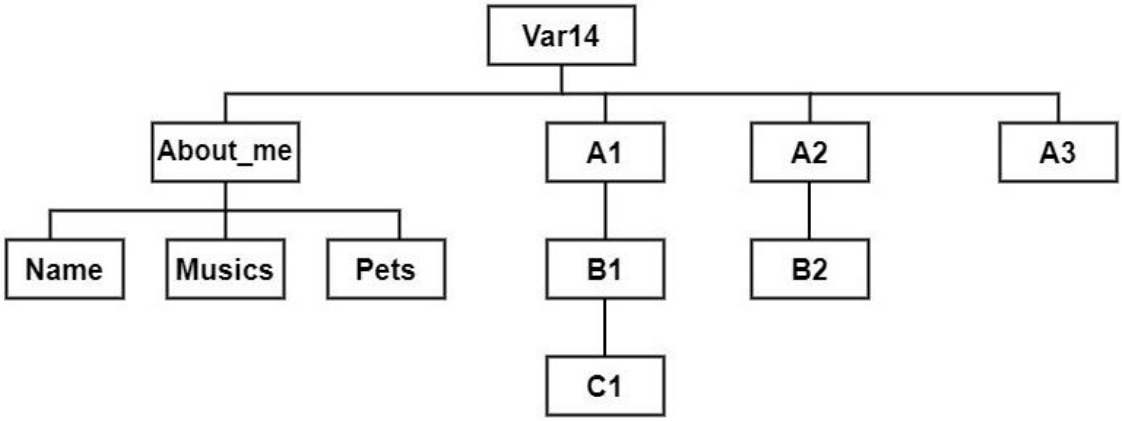


Рис. 10.14. Вариант 14

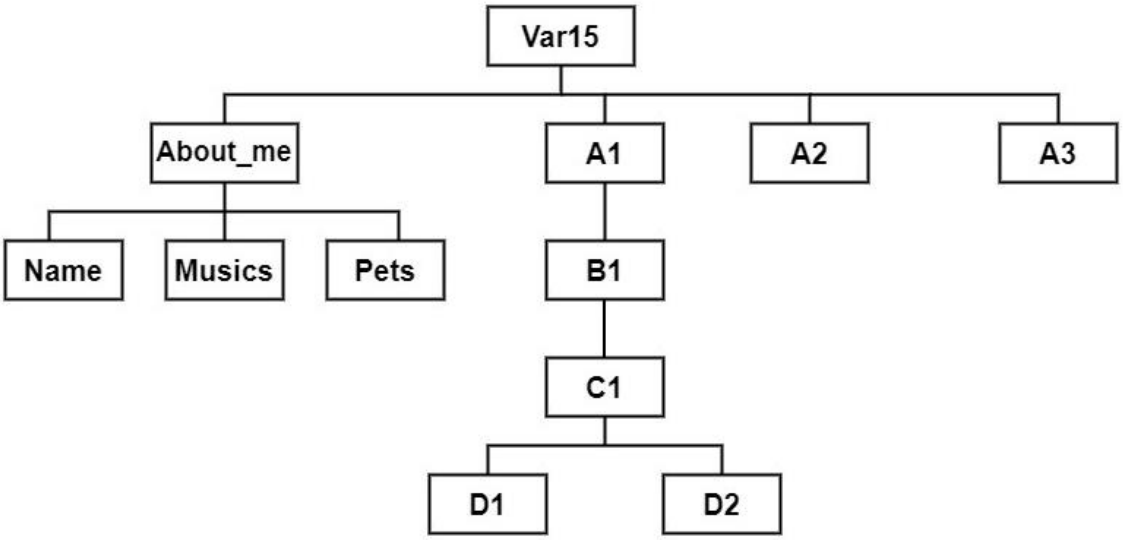


Рис. 10.15. Вариант 15

3. Перейти в каталог *Name* и создать файл *name.txt*, в который нужно вписать свою фамилию, имя, а также дату рождения.

4. Здесь же создать файл *study.txt*, который будет содержать краткую информацию о месте учебы, группе и специальности учащегося.

5. В каталоге *Musics* создать текстовый файл *musics.txt*, содержащий информацию о любимом жанре музыки, а также название любимой группы и песни.

6. В каталоге *Pets* аналогично создать текстовый файл *pets.txt* с описанием домашних животных и их кличками. Если же домашнего животного нет, вписать, хотели бы вы какого-нибудь питомца, и если да, то какого?

7. Объединить файлы *name.txt, study.txt, musics.txt* и *pets.txt* в файл *about_me.txt* каталога *About_me*. После чего отредактировать его так, чтобы получился связный текст.

8. Удалить каталоги *Name, Music* и *Pets*.

9. В каталоге *Var* создать подкаталоги *E1 E2*.

10. Перейти в каталог *E1* и создать текстовый файл *commands.txt*, который будет содержать список команд, использовавшихся при выполнении лабораторной работы.

11. Скопировать файл *about_me.txt* в каталог *Var.*

12. Для защиты лабораторной работы поочередно вывести содержимое файлов *about_me.txt* и *commands.txt* на экран.

Лабораторная работа 4

Используя результаты, полученные в лабораторной работе 3, выполнить следующие задания.

1. Для каталога *About_me* установить права доступа *«только чтение»*.

2. Перейти в каталог *Var(номер варианта)* и выполнить следующие задания:

2.1. Создать каталог *test* и перейти в него.

2.2. Создать файл *file*, определить его права доступа и запретить любые действия с файлом. Записать строку *«тест»* в *file*, а затем прочитать файл.

2.3. Изменить права доступа для *file* так, чтобы только текущий пользователь мог осуществлять в него запись. Записать строку *«test»* в файл. Прочитать содержимое.

2.4. Разрешить группе текущего пользователя читать содержимое файла. Прочитать содержимое.

2.5. Разрешить текущему пользователю читать содержимое файла. Прочитать содержимое – должна появиться ранее добавленная туда строка *«test»*.

2.6. Создать каталог *dir*, а затем создать в нем файл *newfile*, записать в него текст *«file2»*. Просмотреть содержимое каталога.

2.7. Изменить права доступа на каталог *dir* так, чтобы пользователи не имели права на исполнение *(x)*. Просмотреть содержимое файла *newfile* в каталоге *dir*. Попробовать удалить этот файл.

3. Создать в *Var* каталог *Primer*, который будет содержать текстовые файлы *primer1.txt* и *primer2.txt.*

4. Выполнить далее задание своего варианта.

4.1. Установить права *441* каталогу *Primer*.

4.2. Установить файлу *primer1.txt* права *454*, удалить его.

4.3. Установить каталогу *Primer* и его содержимому права *344*.

4.4. Установить всем файлам права *234*.

4.5. Установить права *455* файлу *primer2.txt*.

4.6. Установить права *543* каталогу *Primer*.

4.7. Установить каталогу *Primer* права *552*.

4.8. Установить каталогу *Primer* права *447*.
4.9. Установить каталогу *Primer* права *444*.
4.10. Установить обоим файлам права *333*.
4.11. Установить файлу *primer1.txt* права *411*.
4.12. Установить созданным файлам права *300*.
4.13. Установить каталогу *Primer* и его содержимому права *411*.
4.14. Установить каталогу *Primer* права *432*.
4.15. Установить каталогу *Primer* права *321*.

Лабораторная работа 5

Найти файл (файлы):
– с именем на ваш выбор;
– размер которых меньше 300 байт;
– модифицированные больше четырех дней назад;
– пустые;
– определенного типа (f – обычный файл).

Лабораторная работа 6

1. Изучить назначение и ключи команды *ln*.
2. Создать файл с описанием команд *dpkg, apt, apt-get, ln*.
3. Создать жесткую ссылку на него.
4. Просмотреть содержимое файла, используя ссылку.
5. Создать символьную ссылку на исходный файл.
6. С помощью команды *ls* просмотреть *inod* файла и ссылок.

Лабораторная работа 7

1. Пользуясь командами терминала, установить пакет *ttf-mscorefonts-installer.* Каково назначение этого пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

2. Пользуясь командами терминала, установить пакет *dia*. Каково назначение этого пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

3. Пользуясь командами терминала, установить пакет *wxmaxima*. Каково назначение этого пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

4. Пользуясь командами терминала, установить пакет *g++*. Каково назначение пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

5. Пользуясь командами терминала, установить пакет *geany*. Каково назначение пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

6. Пользуясь командами терминала, установить пакет *gfortran*. Каково назначение пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

7. Скачать пакет *scilab* и установить его, пользуясь командами терминала. Каково назначение пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

8. Пользуясь командами терминала, установить пакет *fpc*. Каково назначение пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

9. Пользуясь командами терминала, установить пакет *gimp*. Каково назначение пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

10. Пользуясь командами терминала, установить пакет *transmission*. Каково назначение пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

11. Пользуясь командами терминала, установить пакет *octave*. Каково назначение пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

12. Пользуясь командами терминала, установить пакет *mathomatic*. Каково назначение пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

13. Пользуясь командами терминала, установить пакет *lazarus*. Каково назначение пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

14. Пользуясь командами терминала, установить пакет *cantor*. Каково назначение пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

15. Пользуясь командами терминала, установить пакет *vlc*. Каково назначение пакета? Создать текстовый файл *Lab7* с ответом на вопрос.

Лабораторная работа 8

1. Создать пользователя *labuser* с настройками по умолчанию. Задайте *labuser* пароль. В каких группах состоит *labuser*? Войти под новым пользователем в систему. Проверить, создался ли домашний каталог пользователя, наполнен ли он файлами и кому принадлежит?

2. Удалить пользователя *labuser*.

3. Создать пользователя со своим именем и паролем (с правами *sudo*). Зайти под новым пользователем в систему. Просмотреть записи для нового пользователя в файлах `/etc/passwd` и `/etc/shadow`.

4. Определить владельца и группу файла `/etc/passwd`.

5. Вывести на экран `UID` и `GID`.

6. Сменить пароль у пользователя с административными правами.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Донцов И.П., Сафин И.В. Linux на примерах. СПб.: Наука и техника, 2017.

2. Кетов Д.В. Внутреннее устройство Linux. СПб.: БХВ-Петербург, 2017.

3. Коэн Д., Штурм К. Linux для разработчиков. Астана: «Спринт Бук», 2025.

4. Курячий Г., Маслинский К. Операционная система Linux: курс лекций. М.: ДМК Пресс, 2010.

5. Матвееев М.Д. Вель Linux. ДЛЯ ТЕХ, КТО ХОЧЕТ СТАТЬ ПРОФЕССИОНАЛОМ. СПб.: Наука и техника, 2024.

6. Негус К., Каэн Ф. Ubuntu и Debian Linux для продвинутых. СПб.: Питер, 2010.

7. Уорд Б. Внутреннее устройство Linux. СПб.: Питер, 2016.

8. Шоттс У. Командная строка Linux. Полное руководство. СПб.: Питер, 2020.

ОГЛАВЛЕНИЕ

Учебное издание

А л е к с е е в Евгений Ростиславович
К о в а л е н к о Юлия Сергеевна
Д о г а Кристина Вячеславовна
О б у х о в а Юлия Николаевна

ОСНОВЫ РАБОТЫ В LINUX

Учебное пособие

Подписано в печать 25.06.2025. Выход в свет 9.07.2025
Печать цифровая.
Уч.-изд. л. 5,7. Тираж 500 экз. Заказ №6142

Кубанский государственный университет
350040, г. Краснодар, ул. Ставропольская, 149.
Издательско-полиграфический центр
Кубанского государственного университета
350040, г. Краснодар, ул. Ставропольская, 149.