

Оглавление

Введение	4
Лабораторная работа №1. Установка операционной системы семейства Linux	11
Лабораторная работа №2. Работа с командной оболочкой bash в ОС семейства Linux	22
Лабораторная работа №3. Написание составных команд в командной оболочке bash	29
Лабораторная работа №4. Основы безопасности в Linux	37
Лабораторная работа №5. Настройка пользовательских демонов с применением systemd	44
Лабораторная работа №6. Работа с сетями и настройка сервера	55
Литература	67
Приложение А	70
Приложение Б	73
Приложение В	75
Приложение Г	76
Приложение Д	78

Введение

Операционные системы семейства Linux находят самое широкое применение в промышленности, администрировании крупных вычислительных сетей, облачных технологиях, а также в машинном обучении. Наличие практических навыков работы с данным классом операционных систем на сегодняшний день является обязательным для инженера, работающего в области информационных технологий. В лабораторном практикуме объединены лабораторные работы по установке и настройке операционной системы Linux, основам сетевого администрирования, информационной безопасности, а также написанию простейших сценариев. Лабораторный практикум предназначен для студентов бакалавриата, обучающихся по направлениям: 12.03.01 Приборостроение, 13.03.01 Теплоэнергетика и теплотехника, 13.03.02 Электроэнергетика и электротехника, 15.03.04 Автоматизация технологических процессов и производств, 15.03.06 Мехатроника и робототехника, 24.03.02 Системы управления движением и навигация, 27.03.04 Управление в технических системах.

Большинство лабораторных работ данного практикума рассчитаны на создание обучающимися некоторого программного обеспечения или настройке каких-либо сервисов, реализуемых в операционной системе Linux.

Все задания могут быть выполнены в виртуальной среде Virtual-Box. По желанию обучающегося вместо виртуальной среды может быть использовано реальное аппаратное обеспечение (персональный компьютер, микрокомпьютер и т.д.). Выбор виртуального или реального окружения не влияет на ход выполнения лабораторных работ.

Постановка каждой задачи лабораторного практикума предельно коротка, однако для понимания задачи необходимо знать материал соответствующего раздела курса «Операционная система Linux», а для выполнения реализации – обладать знаниями о среде выполнения работы. При необходимости более глубокого изучения какой-либо темы, обучающиеся могут обратиться к литературе.

Программа лабораторного практикума

Практикум включает шесть лабораторных работ.

Лабораторная работа 1. Установка операционной системы семейства Linux

Цель работы – получить практические навыки по установке и первичной настройке операционной системы Linux в виртуальном окружении. Работа состоит из трех частей. В первой части осуществляется подготовка таблицы загрузочного накопителя, во второй части создается виртуальная машина, в третьей части выполняется установка дистрибутива операционной системы Linux.

Лабораторная работа 2. Работа с командной оболочкой bash в ОС семейства Linux

Цель работы – научиться работать с командной оболочкой bash операционной системы семейства Linux и изучить основные команды файловой системы. Работа состоит из четырех частей. В первой части выполняется запуск и подготовка виртуального терминала, во второй части выполняются команды навигации по файловой системе, в третьей части изучаются команды манипулирования файлами, в четвертой части исследуются ключи и аргументы команд.

Лабораторная работа 3. Написание составных команд в командной оболочке bash

Цель работы – получение практических навыков по созданию составных команд в оболочке bash. Работа состоит из четырех частей. В первой части осуществляется подготовка виртуального терминала для выполнения команд, во второй части реализуется перенаправление потоков ввода и вывода, в третьей части создается пайплайн команд, в четвертой части создаются переменные командной оболочки.

Лабораторная работа 4. Основы безопасности в Linux

Цель работы – Научиться создавать и администрировать пользователей в операционной системе Linux. Работа состоит из четырех частей. В первой осуществляется подготовка виртуального терминала для выпол-

нения команд, во второй части необходимо получить права суперпользователя в системе, в третьей части создаются новые пользователи и группы, в четвертой части организуется разграничение прав доступа к файлам и директориям для конкретных пользователей и групп.

Лабораторная работа 5. Настройка пользовательских демонов с применением systemd

Цель работы – получение практических навыков по работе с подсистемой инициализации и управления службами systemd. Работа состоит из четырех частей. В первой части осуществляется подготовка программного окружения и генерация варианты работы, во второй части создается bash-скрипт, в третьей части создается юнит файл, в четвертой части запускается пользовательский демон.

Лабораторная работа 6. Работа с сетями и настройка сервера

Цель работы – получение практических навыков по установке и настройке web-сервера nginx. Работа состоит из четырех частей. В первой части осуществляется подготовка программного окружения, во второй части устанавливается и настраивается операционная система Ubuntu Server, в третьей части настраивается удаленный доступ по протоколу ssh, в четвертой части устанавливается и настраивается web-сервер nginx.

Требования к оформлению отчета по лабораторным работам

Общие требования

- 1) Отчет выполняется в виде самостоятельного документа. Материал, изложенный в отчете, должен быть понятен без дополнительных комментариев со стороны исполнителей.
- 2) Отчет выполняется как текстовый документ в соответствии с ГОСТ 2.105-95 и предоставляет в электронном виде.
- 3) В отчёте должна быть представлена последовательность команд, которые были использованы для написания программного модуля или выполнения настройки сервиса операционной системы Linux.

- 4) Листы отчета должны быть пронумерованы, кроме титульного листа, который считается первым.
- 5) Если отчет содержит большое количество листов, рекомендуется добавлять лист с содержанием отчета (разделы и номера листов).

Содержание отчета

Отчет должен содержать следующие разделы:

- 1) Титульный лист.
- 2) Цель работы.
- 3) Задачи, решаемые в работе.
- 4) Теоретическая часть.
- 5) Экспериментальная часть.
- 6) Выводы.

Шаблон отчета представлен в Приложении Д.

Цель работы

В данном разделе должна быть сформулирована цель работы. Допускается копирование цели работы из методических указаний.

Задачи, решаемые в работе

В данном разделе должны быть сформулированы задачи, которые необходимо решить в процессе выполнения лабораторной работы для достижения поставленной цели.

Теоретическая часть

Теоретическая часть отчета по лабораторной работе должна включать в себя всю необходимую информацию о предметной области, к которой относятся описание работы программных компонентов, используемых при ее выполнении, выдержки из технической документации по установке и настройке компонентов операционной системы Linux и т. п.

Данный раздел не является обязательным, однако настоятельно рекомендуется оставлять его в отчете, так как он может быть использован как план ответа при защите лабораторной работы. В случае, если теоретическая часть отчета по лабораторной работе содержит листинги программного кода, таблицы или рисунки, то они должны быть оформлены по правилам, описанным далее.

Экспериментальная часть

Часть отчета, описывающая экспериментальную или практическую часть лабораторной работы, является обязательной и не может быть опущена. Данная часть отражает персональные результаты обучающегося, полученные им при выполнении лабораторной работы. Этими результатами являются листинги написанного программного кода, таблицы и изображения, полученные в процессе решения сформулированных задач. Исходные данные, использованные обучающимся для практической проверки реализованных в ходе практической работы программных компонентов, также должны быть представлены в экспериментальной части.

Данные по выполненной лабораторной работе можно предоставить отдельно, в том числе в виде ссылки на Интернет-ресурс. При изучении ранее выполненных работ обучающимся рекомендуется самостоятельно переписать всю программу в том виде, в каком она была написана, с целью лучшего понимания алгоритма программы.

Если студент не завершил цикл выполнения программы, должно быть проведено ее окончательное редактирование. В экспериментальную часть отчета по лабораторной работе необходимо включить пояснения и комментарии к написанному программному коду (если написание программного кода является одной из задач лабораторной работы).

Выводы

Выводы должны быть кратко изложены в виде списка и отражать наиболее важные аспекты лабораторной работы. В конце отчета студент должен привести список использованной при подготовке отчета и процитированной литературы (в том числе ссылки на стандарты, руководства пользователя и прочую техническую документацию). Текст отчета по лабораторной работе может являться планом ответа при защите лабораторной работы. Прямое чтение текста отчета не допускается.

Правила оформления текста отчета по лабораторной работе

Отчет должен быть представлен в формате PDF. Поля текста могут быть выбраны произвольно, например, 25 мм по всем сторонам печатного листа. Рекомендуется при оформлении отчета использовать шрифт чёрного цвета с засечками (например, DejaVu Serif) высотой 14 пунктов с полуторным интервалом между строками. Обязательным требованием при

оформлении текста отчёта по лабораторной работе является его выравнивание по ширине страницы и использование автоматических переносов.

На рисунках в отчете могут быть представлены блок-схемы, графики, снимки экранов виртуального окружения, а также прочая графическая информация. Рисунки должны быть четкими и легко читаемыми. Если рисунок является снимком экрана, то настоятельно рекомендуется сохранять исходное в форматы без сжатия (например, PNG) или копировать снимок экрана из буфера обмена непосредственно в программу, в которой редактируется текст отчета (например, LibreOffice). Графики и диаграммы рекомендуется создавать в векторных форматах. Ниже представлен пример оформления рисунка.

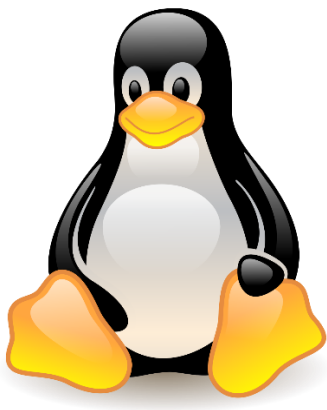


Рисунок X – Пример оформления рисунка

Листинги программного кода должны оформляться как скриншоты окна редактора, либо непосредственно в тексте отчета. В обоих случаях должны быть соблюдены следующие правила:

- 1) Необходимо использовать моноширинные шрифты.
- 2) Строки программы должны быть пронумерованы, нумерация должна начинаться с единицы.
- 3) Должна быть включена «подсветка» синтаксических конструкций используемого языка программирования.

Исходные коды должны быть снабжены комментариями. На рисунки и листинги обязательно должны присутствовать ссылки в тексте. Например

"На рисунке 1 изображен ..." или "Исходный код программы представлен в листинге 2 ...". Листинги, таблицы и рисунки должны быть снабжены подписями, отражающими их содержание.

Защита лабораторной работы

Во время защиты лабораторной работы обещающемуся следует ясно и чётко изложить цель работы и основные решённые задачи, а также ответить на дополнительные вопросы, заданные преподавателем в процессе защиты отчёта. Требования к защите отчета представлены в таблице 1.

Таблица 1 – таблица оценивания защиты лабораторной работы

№ п/п	Требования	Оценка (уровень)		
		высо- кий	сред- ний	низ- кий
1	Уровень оформления отчета	2	1	0,5
2	Навыки устного представления результатов работы	2	1	0,5
3	Понимание (воспроизведение) выполненных команд	2	1	0,5
4	Правильность и полнота понимания достигнутого результата	2	1	0,5
5	Качество ответов на дополнительные вопросы	2	1	0,5
Итого баллов:		10	5	2,5

Лабораторная работа №1

Установка операционной системы семейства Linux

Введение

В настоящее время существует широкое разнообразие дистрибутивов операционных систем (ОС) семейства Linux. Дистрибутив ОС – это комплект программного обеспечения, включающий основу ОС (ядро), модули для поддержки разнообразного аппаратного обеспечения, окружение рабочего стола и минимальный набор необходимых прикладных пакетов. Дистрибутивы могут довольно сильно отличаться друг от друга разным окружением рабочего стола и предустановленными пакетами. Однако все ОС семейства Linux имеют общие принципы организации, такие как общий интерфейс взаимодействия прикладного ПО и ОС (POSIX), общую иерархию файловой системы, общие утилиты ядра, а также универсальный способ взаимодействия с пользователем через терминал.

В этой лабораторной работе можно использовать любой понравившийся дистрибутив Linux. Рекомендуется взять что-то распространенное и Debian-подобное, например, Ubuntu.

Полезные ссылки:

- Интересная статья про дистрибутивы
<https://habr.com/ru/company/lanit/blog/562484/>.
- Стандарт POSIX <https://ru.wikipedia.org/wiki/POSIX>.
- Стандарт HFS <https://ru.wikipedia.org/wiki/HFS>.
- Разнообразие дистрибутивов Linux
https://en.wikipedia.org/wiki/Linux_distribution.

Необходимое ПО:

- Любая ОС, на которую можно установить VirtualBox.
- Среда виртуализации VirtualBox.
- Образ дистрибутива.
- Утилита для создания загрузочных flash-накопителей, например, Rufus.

Описание лабораторной работы

Цель работы

Получить практические навыки по установке и первичной настройке операционной системы Linux в виртуальном окружении.

Указания к выполнению работы

Работа состоит из трех частей. В первой части осуществляется подготовка таблицы загрузочного накопителя, во второй части создается виртуальная машина, в третьей части выполняется установка дистрибутива операционной системы Linux.

Оформление отчета

Отчёт должен быть составлен в соответствии с требованиями, представленными во введении. В отчете необходимо описать процесс выполнения работы, сопроводив его снимками экрана. После установки ОС необходимо самостоятельно исследовать ее интерфейс и внести в отчет свои наблюдения. Следует выявить особенности, которые представляются интересными или непривычными по сравнению с другими операционными системами и провести небольшой сравнительный анализ.

Ход работы

Формирование таблицы разделов

При установке ОС на реальный компьютер важно знать, какая система загрузки ОС используется (BIOS или UEFI). Это можно узнать, загрузив BIOS (UEFI) при старте системы. Если же на ПК уже установлена ОС, то можно определить это ее средствами. А также узнать, какая применяется таблица разделов (MBR или GPT).

- 1) Откройте оснастку для управления дисками. В Windows нажмите правой кнопкой мыши на меню «Пуск» и выберите «Управление дисками».
- 2) Из списка жестких дисков выберите тот, на котором установлена ваша ОС (см. рисунок 1.1).

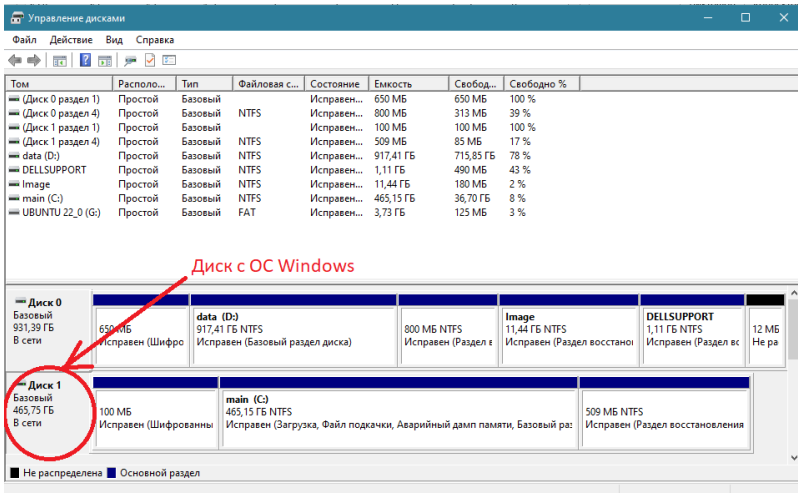


Рисунок 1.1 – Оснастка «Управление дисками»

- 3) Перечислите, какие разделы присутствуют на вашем диске и какой размер они имеют (см. рисунок 1.2).

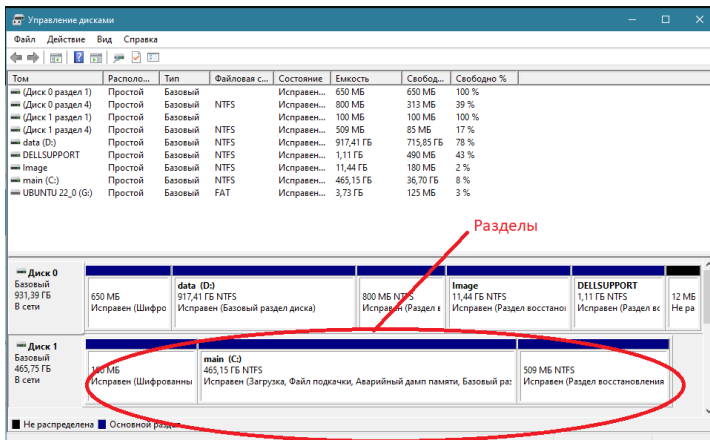


Рисунок 1.2 – Разделы диска

- 4) Определите тип таблицы разделов. Нажмите на название жесткого диска слева (см. рисунок 1.3) правой кнопкой мыши, нажмите «Свойства» и определите тип таблицы разделов.

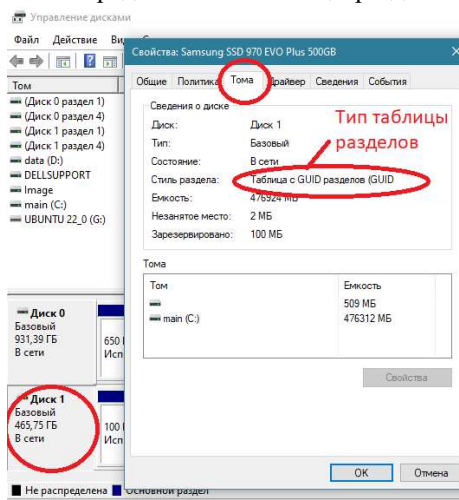


Рисунок 1.3 – Тип таблицы разделов

Создание загрузочного flash-накопителя

- 1) Установите утилиту создания загрузочных flash-накопителей. Скачайте утилиту Rufus по следующей ссылке <https://rufus.ie/ru/>.
- 2) Скачайте интересующий вас образ с дистрибутивом ОС семейства Linux. Например, Ubuntu Mate <https://ubuntu-mate.org/ru/download/>. Вы должны скачать файл с расширением .iso.

Откройте приложение Rufus. В качестве устройства (Device) выберите ваш flash-накопитель. В разделе «Метод загрузки» (Boot Selection) укажите ваш скачанный ISO-образ. В схеме разделов (partition scheme) выберите GPT, в качестве целевой системы (target system) – UEFI. Остальные параметры оставьте без изменений (см. рисунок 1.4).

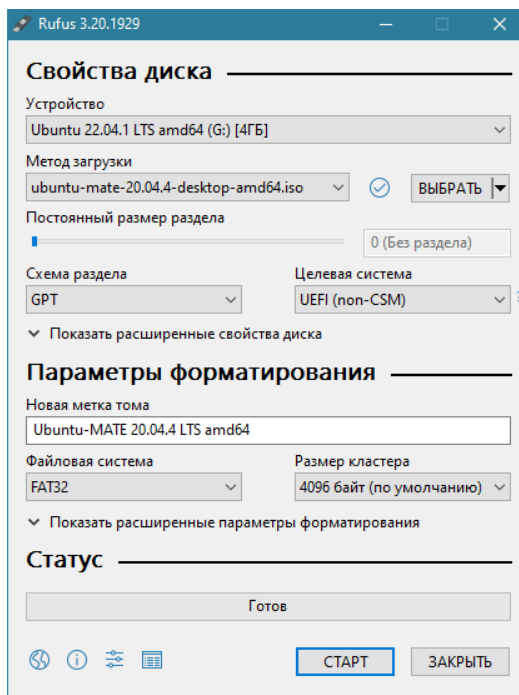


Рисунок 1.4 – Создание загрузочной флешки

- 3) Нажмите «Старт». Обратите внимание, что все данные на flash-накопителе будут удалены.

Создание виртуальной машины

- 1) Установите ПО VirtualBox по ссылке <https://www.virtualbox.org/wiki/Downloads> для своей хостовой системы.
- 2) Запустите VirtualBox и нажмите кнопку «Создать» (см. рисунок 1.5).

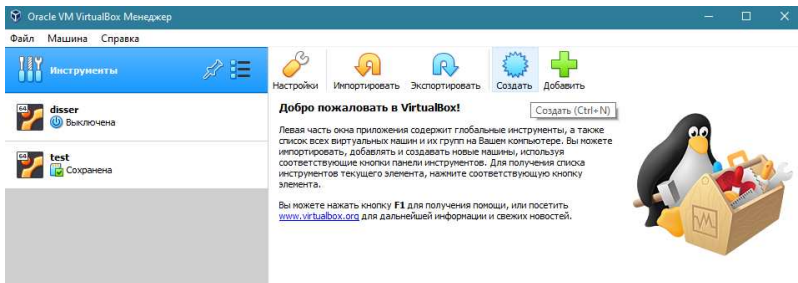


Рисунок 1.5 – ПО VM VirtualBox

- 3) Выберите экспертный режим. Установите тип – Linux, версия – Ubuntu (или другая, соответствующая вашему дистрибутиву). Объем памяти – 2048Мб. В разделе «Жесткий диск» выбрать «Создать новый виртуальный жесткий диск». Желательно, чтобы значения лежали в зеленой зоне на шкале. В противном случае возможно замедление работы вашей ОС (см. рисунок 1.6).

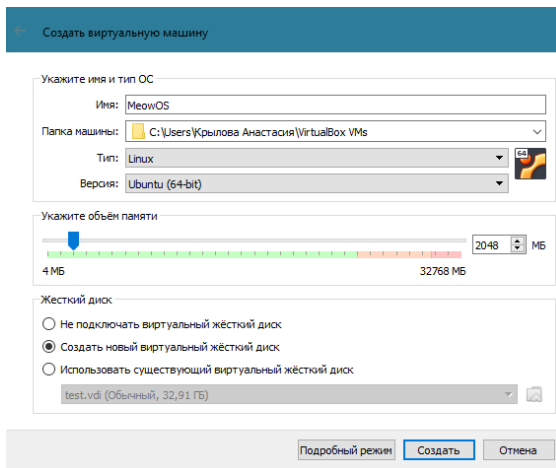


Рисунок 1.6 – Создание виртуальной машины

- 4) Нажмите «Создать».

- 5) В разделе «Создать виртуальный жесткий диск» выберите тип VDI, формат «Динамический виртуальный жесткий диск» ОС (см. рисунок 1.7).

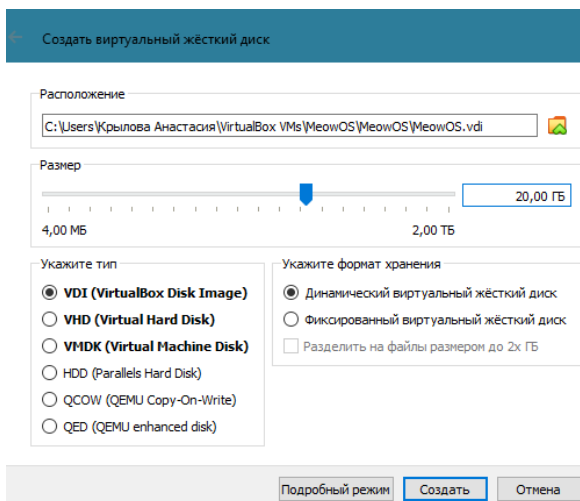


Рисунок 1.7 – Создание виртуального жесткого диска

- 6) Выберите вашу созданную машину и нажмите «Настроить».
- 7) В разделе «Общее» во вкладке «Дополнительно» установите «Общий буфер обмена» и «Drag’n’Drop» в состояние «Двунаправленный».
- 8) В разделе «Система» во вкладке «Процессор» можно увеличить количество процессоров для более быстрой работы системы. В данном случае установлено 4 (см. рисунок 1.8). Значение должно быть в зеленой зоне шкалы. В разделе «Видеопамять» выставляем максимально возможное значение. В данном случае установлено 128 Мб (см. рисунок 1.9).

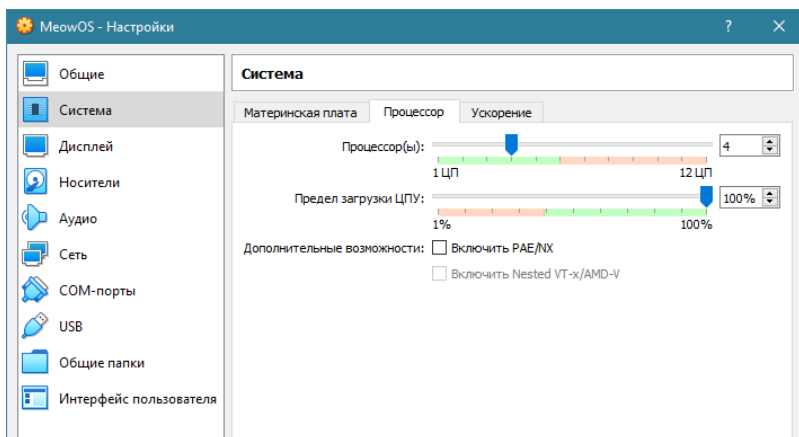


Рисунок 1.8 – Настройка виртуальной машины

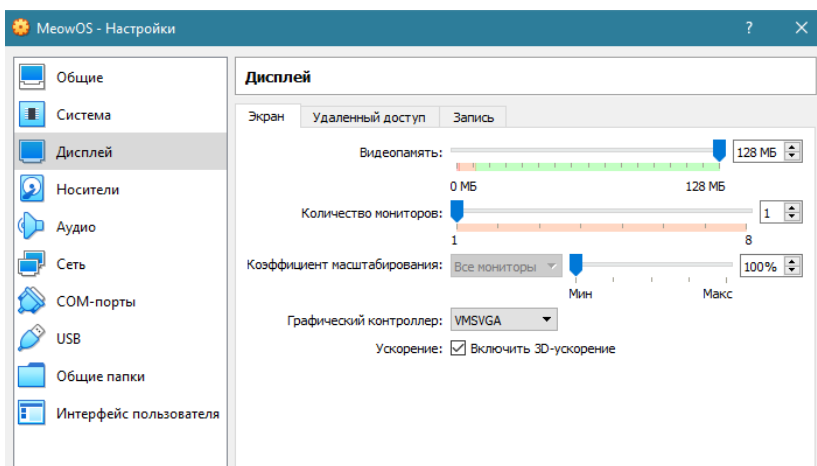


Рисунок 1.9 – Настройка виртуальной машины раздел «Дисплей»

- 9) В разделе «Носители» выберите «Контроллер IDE» и нажмите на изображение оптического диска. В разделе «Атрибуты» снова нажмите на оптический диск → «Выбрать файл диска». Выберите .iso выбранного вами дистрибутива. (см. рисунок 1.10).

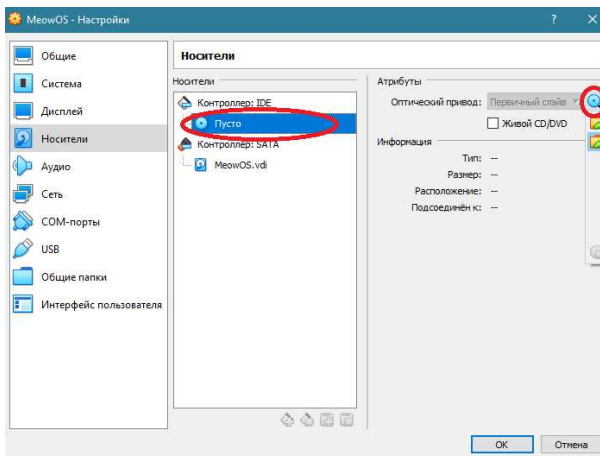


Рисунок 1.10 – Настройка виртуальной машины раздел «Носители»

10) Настройка виртуальной машины завершена. Нажмите «ОК».

Запуск виртуальной машины и установка системы

- 1) Нажмите «Запустить»
- 2) После запуска вы увидите рабочий стол, и начнется процесс установки, нажмите «Install Ubuntu Mate» (см. рисунок 1.11).

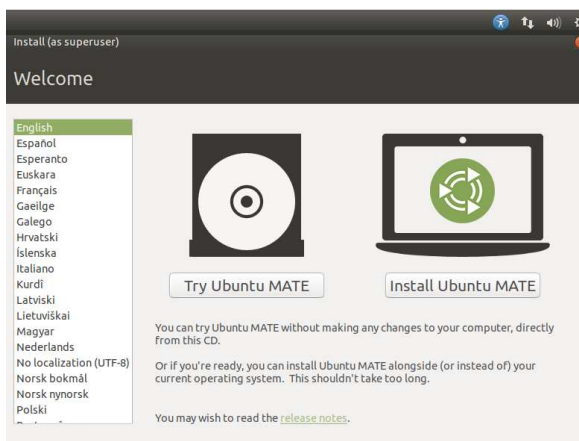


Рисунок 1.11 – Экран при старте Ubuntu Mate

- 3) Далее следуйте согласно инструкциям на экране. Желательно при установке выбрать английский язык, а не русский. Это отразится на именах каталогов в дальнейшем, так как с названиями на кириллице не очень удобно работать из терминала. Также вас попросят создать логин и пароль. Запишите их и не забывайте! Они понадобятся для выполнения следующих лабораторных работ.
- 4) После установки вас попросят перезагрузить систему. Появится рабочий стол или окно логина. Войдите в систему.
- 5) Для поддержки полноэкранного режима нам потребуется установить гостевые утилиты. Для этого на верхней панели нажмите устройства и выберите «Подключить образ диска Дополнений гостевой ОС» (см. рисунок 1.12).

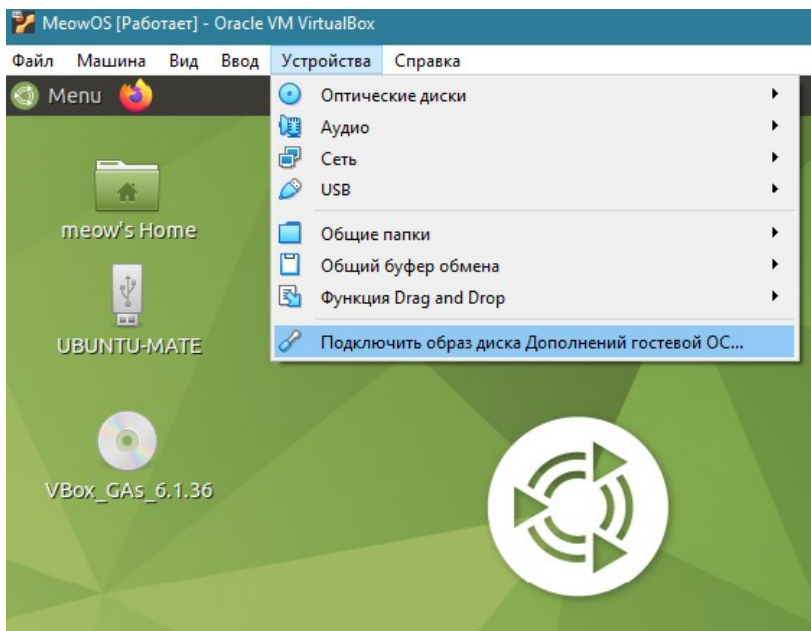


Рисунок 1.12 – Установка «Дополнений гостевой ОС»

- 6) В появившемся окне нажмите Ок, затем Run (см. рисунок 1.13).

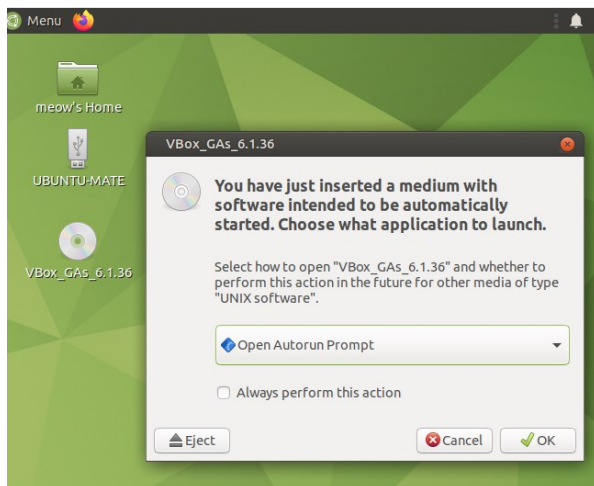


Рисунок 1.13 – Запуск установки дополнений

- 7) Перезагрузите ОС.
- 8) Перетащите на рабочий стол прикладываемый к лабораторной работе файл `blueprint.sh`, либо создайте этот текстовый файл в любом текстовом редакторе. Содержимое файла следующее:

```
hostnamectl >> result
cat result | sha1sum >> result
```

- 9) Нажмите правой кнопкой мыши по файлу. Выберите «Properties». В открывшемся окне зайдите во вкладку Permissions. Установите флаг Allow executing file as program.
- 10) После этого два раза щелкните на иконку файла и нажмите Run.
- 11) На рабочем столе должен появиться файл `result`. Перенесите его на свой компьютер и прикрепите вместе с отчетом.

Контрольные вопросы

- 1) Для чего используются среды виртуализации аппаратного обеспечения?
- 2) Какие инструменты и методы используются при создании загрузочных накопителей?
- 3) Что такое дополнения гостевой ОС и для чего они применяются?

Лабораторная работа №2

Работа с командной оболочкой `bash` в ОС семейства Linux

Введение

Командная оболочка (command shell) предназначена для того, чтобы обеспечивать взаимодействие пользователя и ОС через командную строку. В разных дистрибутивах ОС семейства Linux могут применяться разные командные оболочки. Видов командных оболочек, конечно, несравнимо меньше, чем количество видов дистрибутивов Linux, однако с большой долей вероятности на самых популярных дистрибутивах вам попадется оболочка `bash`.

В отличие от графического интерфейса, командная оболочка – это универсальный способ взаимодействия с Linux-системами. Более того, командная оболочка дает самые широкие возможности по настройке, управлению ОС, а также выполнению рутинных задач.

Доступ к командной оболочке осуществляется через виртуальные терминалы либо через эмулятор терминала. В данной работе мы будем работать с эмулятором терминала и для краткости в дальнейшем будем называть его просто «терминал».

Терминал работает по принципу REPL (Read-Eval-Print-Loop). Фактически взаимодействие с пользователем осуществляется по следующему сценарию:

- 1) Read – оболочка ждёт ввода команды от пользователя.
- 2) Eval – оболочка исполняет введенную команду.
- 3) Print – оболочка выводит результат.
- 4) Loop – возвращаемся к первому пункту.

Полезные ссылки:

- Поучительная притча <https://www.livelib.ru/quote/61563-iskusstvo-programmirovaniya-dlya-unix-erik-c-rejmond>.
- Для тех, кто хочет глубже разобраться с терминалами <https://habr.com/ru/company/neobit/blog/330764/>.
- Удобный сайт, объясняющий введенную команду <https://explainshell.com/>.

Важно! Далее по тексту в строках, где указываются команды, которые необходимо вводить в терминал, знак \$ вводить не нужно. Этим знаком обозначается обычное приглашение консоли.

Описание лабораторной работы

Цель работы

Научиться работать с командной оболочкой `bash` операционной системы семейства Linux и изучить основные команды файловой системы.

Указания к выполнению работы

Работа состоит из четырех частей. В первой части выполняется запуск и подготовка виртуального терминала, во второй части выполняются команды навигации по файловой системе, в третьей части изучаются команды манипулирования файлами, в четвертой части исследуются ключи и аргументы команд.

Оформление отчета

Отчёт должен быть составлен в соответствии с требованиями, представленными во введении. Экспериментальную часть работы необходимо сопроводить несколькими снимками экрана. Весь процесс фиксировать снимками экрана не нужно. В отчете необходимо дать ответы на вопросы, которые задаются по ходу работы, а также добавить требуемые данные (например, содержимое секретного файла). К отчету следует приложить файл `record_result`. Это текстовый файл с последовательностью выполненных команд.

Ход работы

Подготовка

- 1) Запустите виртуальную машину и дождитесь загрузки ОС.
- 2) Откройте терминал. Это можно сделать с помощью основного меню или сочетанием клавиш **Ctrl+Alt+T**. Для запуска через основное меню нажмите на Menu → System Tools → Mate Terminal (см. рисунок 2.1)

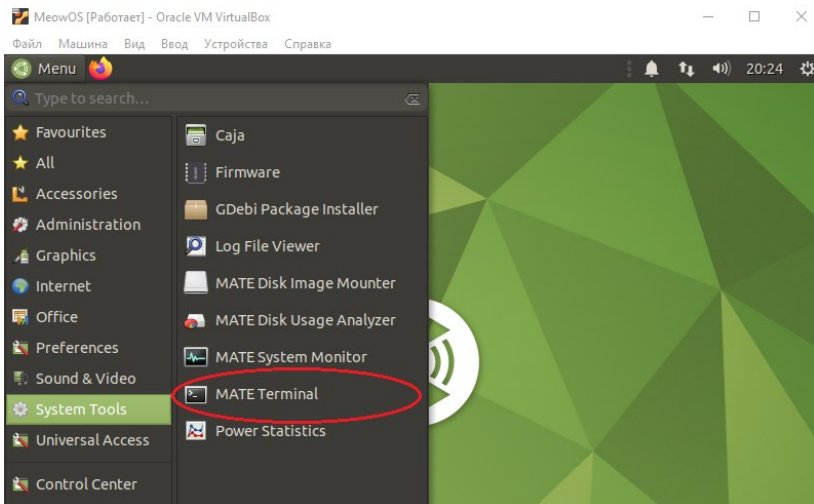


Рисунок 2.1 – Открытие терминала через графический интерфейс

- 3) **Запустите запись терминала. Это важно!** Именно файл с записью истории ваших команд будет основной частью отчета. Для этого введите и запустите команду

```
$ script -a record_result
```

- 4) Убедитесь, что команда выполнилась без ошибок. Введите нижеуказанную команду и найдите имя файла `record_result` в ее выводе:

```
$ ls -l
```

Навигация

- 1) Поместите на рабочий стол файл `lab2.py` (см. приложение А).
- 2) Вернитесь в терминал, в котором уже идет запись (см. Подготовка).
- 3) Выведите путь текущей рабочей директории. Для этого введите:

```
$ pwd
```

- 4) Выведите список всех файлов и каталогов в этой директории. Для этого введите:

```
$ ls -l
```

- 5) Перейдите на рабочий стол. Для этого введите:

```
$ cd ./Desktop
```

в случае с русским языком нужно использовать экранирование пробела:

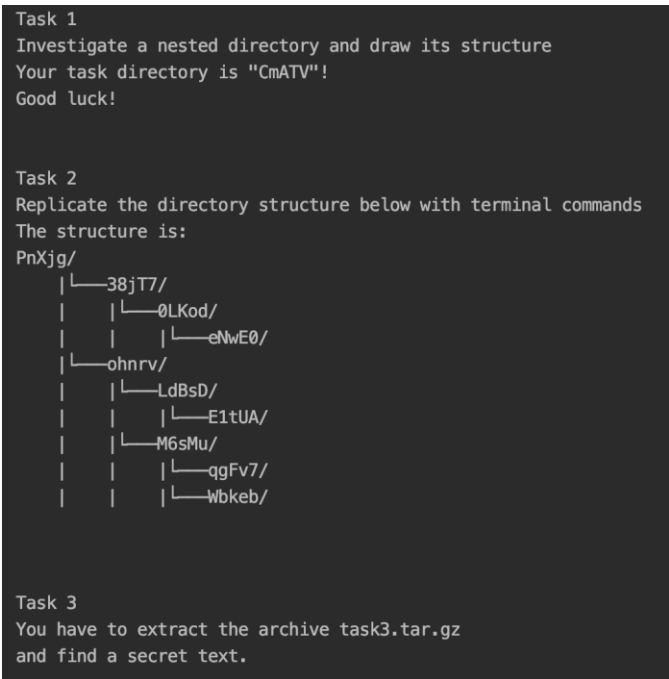
```
$ cd ./Рабочий\ стол/
```

- 6) Выведите список всех файлов и каталогов. Убедитесь, что в выводе есть файл `lab2.py`.

```
$ ls -l
```

- 7) Запустите генерацию вашего варианта (см. рисунок 2.2). Вывод этой команды добавьте в отчет в виде скриншота. Для генерации:

```
$ python lab2.py
```



```
Task 1
Investigate a nested directory and draw its structure
Your task directory is "CmATV"!
Good luck!

Task 2
Replicate the directory structure below with terminal commands
The structure is:
PnXjg/
|  └─38jT7/
|    └─┬─0LKod/
|        └─┬─eNwE0/
|            └─ohnrv/
|              └─┬─LdBsD/
|                  └─┬─E1tUA/
|                      └─M6sMu/
|                        └─┬─qgFv7/
|                            └─Wbkeb/

Task 3
You have to extract the archive task3.tar.gz
and find a secret text.
```

Рисунок 2.2 – Результат генерации варианта

- 8) Посмотрите на задание Task 1. Там указано название директории, которую вам надо исследовать. Task 2 и Task 3 нам понадобятся позднее, пока они не нужны.
- 9) Исследуйте структуру созданной директории через терминал. Для этого используйте следующие команды:

```
$ ls -l # перечисляет папки и файлы рабочей директории
$ cd my_dir # переходит в директорию my_dir
$ cd /absolute/path # можно перейти по абсолютному пути
$ cd ./relative/path # можно перейти по относительному пути
$ cd .. # вернуться на уровень вверх
$ cd ../../.. # вернуться на два уровня вверх
$ cd ~ # перейти в домашнюю директорию
```

- 10) Среди директорий найдите текстовый файл. Выведите его содержимое в терминал. Его содержимое добавьте в отчет. Для вывода:

```
$ cat имя_найденного_файла
```

- 11) Нарисуйте дерево, соответствующее структуре исследованной вами папки, любым удобным способом и поместите в отчет.

Манипулирование файлами

- 1) Посмотрите на раздел Task 2 вашего варианта, который вывела нам ранее запущенная команда генерации заданий `python lab2.py`.
- 2) Перед собой вы увидите дерево директорий, представленное в символьном виде в терминале (см. рисунок 2.2).
- 3) Создайте соответствующую дереву структуру директорий с помощью терминала. Для этого вам понадобятся команды:

```
$ mkdir dir_name # создает директорию
$ mkdir -p вложенные/папки/ # рекурсивно создает директорию
$ cd dir_name # перейти в директорию
$ ls -l # перечисляет папки и файлы рабочей директории
$ rm dir_name # удаляет директорию
```



```
$ rm -rf dir_name # удаляет директорию рекурсивно
```

- 4) Перейдите в корневую папку вашего дерева.
- 5) В корневой папке создайте три файла следующей командой:

```
$ touch фамилия_1 фамилия_2 фамилия_3
```

- 6) Откройте каждый файл и запишите в него что-нибудь. Для этого откройте консольный редактор nano. Внесите произвольные изменения. Чтобы сохранить, нажмите `ctrl+x`, затем «Y»:

```
$ nano file_name
```

- 7) Файл фамилия_1 скопируйте и сохраните в той же папке. Для этого введите:

```
$ cp фамилия_1 copy_name
```

- 8) Файл фамилия_2 переместите в папку на уровень ниже (глубже) и переименуйте. Для этого:

```
$ mv фамилия_2 ./dir_name/new_name_2
```

- 9) Удалите файл фамилия_3:

```
$ rm -rf фамилия_3
```

- 10) Удалите рекурсивно все файлы, название которых заканчивается на цифру

```
$ find . -type f -name '[0-9]' -delete
```

Ключи и аргументы команд

- 1) Посмотрите справку команды `tar`. Для этого воспользуйтесь следующими командами справки:

```
$ tar --help
```

```
$ man tar
```

```
$ info tar
```

- 2) Изучите формат команды. Ответьте на вопросы. Какой ключ используется, чтобы создать архив? Какой ключ используется, чтобы извлечь архив? Какой ключ используется для выбора формата компрессии? Какой ключ используется для установки имени архива/директории?

- 3) Создайте архив `my_new_archive.tar.gz` из директории, созданной в предыдущем разделе «Манипулирование файлами» с использованием команды `tar`. Прикрепите его к отчету.
- 4) Извлеките файл из архива, созданного в предыдущем разделе «Навигация», в `Task 3` с использованием команды `tar`. Прочитайте содержимое `secret_file` из архива и добавьте в отчет.
- 5) Завершите запись команд, набрав в терминале:
`$ exit`

Контрольные вопросы

- 1) Как расшифровывается аббревиатура `bash`?
- 2) Что такое виртуальный терминал?
- 3) В чем отличие системных и встроенных команд оболочки?
- 4) Что такое архивация, чем она отличается от сжатия?
- 5) Что такое текущая рабочая директория?

Лабораторная работа №3

Написание составных команд в командной оболочке `bash`

Введение

Команды сами по себе являются программами, скомпилированными в бинарные файлы, которые лежат в директории `/bin`. Вызывая команду в терминале, вы фактически запускаете отдельный процесс. Каждый запущенный процесс в системе Linux имеет связанные с ним три информационных канала:

- Стандартный поток ввода `stdin`.
- Стандартный поток вывода `stdout`.
- Стандартный поток ошибок `stderr`.

Каждый из этих каналов имеет свой файловый дескриптор. Файловый дескриптор – это индекс (целое неотрицательное число), указывающий на строку в специальной таблице, содержащую информацию об открытых процессом файлах. Стандартные потоки имеют фиксированные значения файловых дескрипторов: `STDIN` – 0, `STDOUT` – 1, `STDERR` – 2.

Кроме этого, каждый процесс связан со своим управляющим терминалом. Стандартные потоки процесса по умолчанию связаны с таким терминалом. Именно поэтому в окне нашего терминала отображается результат вывода команд. А также благодаря этой связи введенный текст при необходимости попадает на вход запущенных команд.

Эти три стандартных потока фактически выступают универсальным интерфейсом, с помощью которого мы можем взаимодействовать с процессом, связывать команды, выстраивать цепочки (пайплайны) или перенаправлять потоки в места отличные от терминала (файлы, сокет и т. д.).

Кроме ключей, аргументов и стандартного потока ввода процесс может брать входную информацию из переменных окружения. Окружение – это не что иное как строки, содержащие пары «ключ=значение». Чаще всего их используют для конфигурирования, чтобы изменять и настраивать поведение тех или иных команд. **Важно!** Далее по тексту, где указываются команды, которые необходимо вводить в терминал, знак \$

вводить не нужно. Этим знаком обозначается обычное приглашение консоли.

Полезные ссылки:

- Про параметры оболочки: <https://www.gnu.org/software/bash/manual/bash.html#Shell-Parameter-Expansion>
- Организация циклов в bash: <https://www.cyber-citi.biz/faq/linux-unix-bash-for-loop-one-line-command/>

Описание лабораторной работы

Цель работы

Получение практических навыков по созданию составных команд в оболочке bash.

Указания к выполнению работы

Работа состоит из четырех частей. В первой части осуществляется подготовка виртуального терминала для выполнения команд, во второй части реализуется перенаправление потоков ввода и вывода, в третьей части создается пайплайн команд, в четвертой части создаются переменные командной оболочки.

Оформление отчета

Отчёт должен быть составлен в соответствии с требованиями, представленными во введении. Экспериментальную часть отчета необходимо сопроводить несколькими снимками экрана. К отчету необходимо приложить файл `record_result_lab_3`. Это текстовый файл с введенными командами. По результатам выполнения работы необходимо ответить на вопросы, насколько удобно писать сложные команды на bash, а также в чем достоинства и недостатки использования пайплайна?

Ход работы

Подготовка

- 1) Запустите виртуальную машину и дождитесь загрузки ОС.

- 2) Откройте терминал. Это можно сделать с помощью основного меню или сочетанием клавиш **Ctrl+Alt+T**. Для запуска через основное меню нажмите на Menu → System Tools → Mate Terminal.
- 3) **Запустите запись терминала. Это важно!** Именно файл с записью истории ваших команд будет основной частью отчета. Для этого введите и запустите команду:

```
$ script -a record_result_lab_3
```

- 4) Убедитесь, что команда выполнилась без ошибок. Введите нижеуказанную команду и найдите имя файла `record_result_lab_3` в ее выводе:

```
$ ls -l
```

Перенаправление

Поместите на рабочий стол файлы, прилагаемые к лабораторной работе (`lab3.py`, `lab3_2.py`, `the-matrixreloaded.csv`, `variants.txt`). Исходный код программ `lab3.py` и `lab3_2.py` находятся в приложении Б.

- 1) Вернитесь в терминал, в котором уже идет запись (см. Подготовка).
- 2) Перейдите на рабочий стол. Для этого:

```
$ cd ./Desktop
```

в случае с русским языком нужно использовать экранирование пробела:

```
$ cd ./Рабочий\ стол/
```

- 3) Выведите список всех файлов и каталогов. Убедитесь, что в выводе есть файл `lab3.py`.

```
$ ls -l
```

- 4) Запустите генерацию вашего варианта. Вывод этой команды добавьте в отчет в виде скриншота. Для генерации:

```
$ python lab3.py
```

- 5) Снова запустите команду выше и перенаправьте стандартный поток вывод `STDOUT` в файл `фамилия_out`. Для этого используйте символ перенаправления потока `STDOUT` «>»:

```
$ python lab3.py > фамилия_out
```

- 6) Снова запустите команду выше и перенаправьте стандартный поток ошибок `STDERR` в файл `фамилия_егг`. Для этого используйте символ перенаправления потока `<>` и номером файлового дескриптора, соответствующего этому потоку. Обратите внимание, что `<>` не должен обрамляться пробелами:

```
$ python lab3.py 2>фамилия_егг
```

- 7) Откройте файлы и убедитесь, что там появился текст:

```
$ nano фамилия_out
```

```
$ nano фамилия_егг
```

- 8) Две предыдущие команды можно записать в одну строку. Созданные файлы предварительно удалите командой `rm`.

```
$ python lab3.py>фамилия_out 2>фамилия_егг
```

- 9) Проверьте, что файлы снова создались и наполнены текстом.

- 10) Как вы могли заметить, текст в этих файлах представляет собой диалог, но часть фраз находится в одном файле, а часть в другом. Давайте восстановим диалог. Для этого нам требуется перенаправить `STDOUT` и `STDIN` в один файл. Символ `&` позволяет понять командой оболочке, что `1` — это номер файлового дескриптора (`STDOUT`), а не имя файла. Для этого:

```
$ python lab3.py>фамилия_final 2>&1
```

- 11) Теперь давайте посчитаем количество слов в каждом созданном файле. Для этого воспользуемся утилитой `wc`:

```
$ wc -w < имя_файла
```

- 12) В отчет добавьте содержание полученных файлов и подсчитанное количество слов.

Пайплайн

Теперь научимся связывать стандартные потоки команд. Такая связь называется пайплайном. Вместе с этим файлом лабораторной работы для вас прикладываются файлы `the-matrix-reloaded.csv`, `variants.txt`, `lab3_2.py`.

- 1) Попробуем поискать в `the-matrix-reloaded.csv` некоторые строки по составным запросам. Для этого будем использовать утилиту `grep`. Изучите руководство пользователя по этой утилите.

```
$ man grep
```

```
$ grep pattern_for_search file.txt # Пример использования
```

- 2) Сгенерируйте свой вариант запустив скрипт `lab3_2.py`. В той же папке должен находиться файл `variants.txt`.

```
$ python lab3_2.py
```

- 3) В терминале вы увидите следующее, пример приведен на рисунке 3.1:

- а) Чьи фразы необходимо найти
- б) Кто должен упоминаться в этих фразах
- в) Какой знак препинания должен содержаться
- г) Какой объект, явление, событие, действие упоминается

От вас потребуется найти все строки для запроса «а», затем для «а и б», затем для «а, б и в» и так далее. Далее работа будет рассматриваться на примере задания с рисунка 3.1. Вам же потребуется сделать на примере своего варианта.

```
a) The character that says a phrase is "Neo"
b) The character that is talked with/about is "Trinity"
c) Should contain punctuation ","
d) The subject that is talked about "love"
```

```
Process finished with exit code 0
```

Рисунок 3.1 – Пример результата генерации задания для пайплайна

- 4) Используйте утилиту `cat`, чтобы вывести содержимое файла в консоль. Но вывод перенаправьте с терминала на утилиту `grep`. Для этого используется символ «`|`». Найдите строки, в которых говорит «Нео» (запрос «а» из задания), для этого выполните команду:

```
$ cat the-matrix-reloaded.csv | grep Neo
```

Можно заметить, что в выводе появляются фразы, где говорит не сам Нео, а где говорят о нём. Чтобы их убрать, давайте скажем `grep`, чтобы строка начиналась с Нео. Для этого используем символ «^»:

```
$ cat the-matrix-reloaded.csv | grep ^Neo
```

- 5) Теперь добавьте к этой команде еще поиск строк, в которых Нео упоминает Trinity (запрос «б» из задания):

```
$ cat the-matrix-reloaded.csv | grep ^Neo | grep Trinity
```

Как можно увидеть из команды выше, мы можем использовать утилиту повторно и соединять результат первого использования утилиты `grep` с входом второго вызова утилиты `grep`. Это работает для всех утилит.

- 6) Теперь добавьте к этой команде еще поиск строк, в которых присутствует запятая (запрос «в» из задания).

```
$ cat the-matrix-reloaded.csv | grep ^Neo | grep Trinity  
| grep ,
```

- 7) Теперь добавьте к этой команде еще поиск строк, в которых используется слово «love»:

```
$ cat the-matrix-reloaded.csv | grep ^Neo | grep Trinity  
| grep , | grep love
```

- 8) В конце посчитаем количество полученных строк. Для этого воспользуемся утилитой `wc` (известной нам по предыдущей лабораторной работе).

```
$ cat the-matrix-reloaded.csv | grep ^Neo | grep Trinity  
| grep , | grep love | wc -l
```

- 9) А теперь давайте выведем количество строк без использования `wc`. Вы же прочитали `man` для `grep`? Найдите там необходимый ключ.

Переменные

Создание переменных может пригодиться при составлении сложных команд в терминале. Чтобы создать переменную достаточно сделать следующее:

```
$ MY_VAR=3 # создали переменную равную 3
```



```
$ echo $MY_VAR
```

Обратите внимание, что знак «=» не обрамляется пробельными символами. Далее по тексту \$ в начале строки означает приглашение к вводу и не набирается, однако **\$ в середине команды** имеет значение «раскрытие значения переменной» и должен набираться в терминале.

- 1) Возьмите персонажа из предыдущего задания по пайплайну (в примере это «Neo»). Создадим переменную с именем актера.

```
$ name='Keanu Reeves'
```

- 2) Прочитайте мануал для команды `sed`. Это утилита является потоковым редактором текста.

```
$ man sed
```

- 3) Выведите все фразы заданного персонажа так, чтобы вначале вывелось «Text for Keanu Reeves». Для этого напишем следующую команду:

```
cat the-matrix-reloaded.csv | grep ^Neo | sed "1i\ --  
Text for $name "
```

- 4) Чтобы не листать вывод, выведем только первые 5 строк:

```
$ cat the-matrix-reloaded.csv | grep ^Neo | sed "1i\  
Text for $name" | head -5
```

- 5) Теперь заменим во всех фразах имя персонажа на имя актера и запишем в файл.

```
$ cat the-matrix-reloaded.csv | grep ^Neo | sed  
"s/Neo,/$name:/" > keanu_text
```

- 6) Теперь пронумеруем строки в файле, для этого воспользуемся циклом `while`. Для этого напишем следующую команду:

```
counter=1; while read line; do echo $counter $line;  
((counter++)); done < keanu_text
```

- 7) Рассмотрим команду подробнее. Вначале создается переменная счетчика

```
counter=1
```

- 8) Затем создается шапка цикла `while`. Цикл `while` имеет следующий синтаксис:

```
while [ condition ]; do <commands>; done
```

или

```
while <control-command>; do <COMMANDS>; done
```

- 9) В шапке цикла `while` используем команду `read`. Команда `read` считывает одну строку из стандартного потока ввода.

```
while read line;
```

- 10) Затем переходим к телу цикла. В теле цикла командой `echo` выводим значение переменных `$counter` `$line`.

```
echo $counter $line
```

- 11) Затем в теле цикла инкрементируем счетчик. Это арифметическое выражение. Для подстановки результата арифметического выражения используются двойные круглые скобки:

```
((counter++))
```

- 12) Тело цикла закрывается ключевым словом `done`.

- 13) На вход всей этой сложной конструкции подаем вывод из файла `keanu_text` с помощью конструкции:

```
(ваша длинная команда с циклом) <keanu_text
```

- 14) Завершите запись команд, набрав в терминале:

```
$ exit
```

Контрольные вопросы

- 1) Перечислите потоки ввода и вывода, ассоциированные с любым процессом в операционной системе Linux.
- 2) Что такое пайплайн?
- 3) Как осуществляется перенаправление потоков ввода/вывода в командной оболочке `bash`?
- 4) Для чего используется утилита `grep`?

Лабораторная работа № 4

Основы безопасности в Linux

Введение

Пользователь является ключевым понятием в системе доступа Linux. Как правило, в ОС домашнего компьютера редко бывает больше двух пользователей, однако в случае распределённых локальных систем учётных записей может быть десятки или даже сотни. В Ubuntu, как, впрочем, и в Windows, у пользователя есть своё выделенное пространство – рабочая или домашняя директория, доступ к которой имеет только он и пользователь с расширенными правами доступа (администратор). Но он также может иметь доступ к общим программам и утилитам – и здесь важно организовать всё правильно с точки зрения безопасности файлов и данных.

В данной работе мы рассмотрим процесс создания и удаления пользователей, назначение файлам прав доступа, а также некоторые важные аспекты работы в ОС.

Полезные ссылки:

- https://sysadminium.ru/groups_and_users_in_linux/ – про пользователей и группы.
- <https://habr.com/ru/post/469667/> – про права доступа к файлам.
- <https://habr.com/ru/post/423049/> – про процессы.

Необходимое ПО:

- Любая ОС, на которую можно установить VirtualBox.
- VirtualBox.
- Образ Ubuntu версии 16.04 или выше.

Описание лабораторной работы

Цель работы

Научиться создавать и администрировать пользователей в операционной системе Linux.

Указания к выполнению работы

Работа состоит из четырех частей. В первой осуществляется подготовка виртуального терминала для выполнения команд, во второй части необходимо получить права суперпользователя в системе, в третьей части создаются новые пользователи и группы, в четвертой части организуется разграничение прав доступа к файлам и директориям для конкретных пользователей и групп.

Оформление отчета

Отчёт должен быть составлен в соответствии с требованиями, представленными во введении. В отчете необходимо описать процесс выполнения работы, сопроводив его снимками экрана.

Ход работы

Перед началом работы

Включите запись терминала командой:

```
$script -a l4_result
```

Если вы работаете в нескольких терминалах, то запустите запись в каждом, назвав файлы `l4_result_<пользователь1>`, `l4_result_<пользователь2>` и т. д. Обязательно приложите файлы к отчёту!

Root права

В Unix-подобных системах пользователь с правами `root` всемогущ – он обладает доступом ко всему, что можно делать в системе, в том числе имеет возможность сделать с ней что-нибудь непоправимое. Поэтому правами `root` следует пользоваться с осторожностью, чтобы не навредить другим пользователям в системе. Пользователь с данными правами создаётся автоматически при установке системы.

Чтобы выполнить команду от имени пользователя `root`, добавьте в начало команды `sudo`. К примеру, попробуйте обратиться к файлу `etc/passwd`:

```
$ sudo cat /etc/passwd
```

Опишите структуру файла и поля, которые в нём содержатся.

Пользователи

Создание и удаление пользовательских учётных записей – рутинная работа администратора. При создании нового пользователя в Ubuntu происходят три события:

- 1) Заводится запись в файле `/etc/passwd`, в которой указано уникальное имя пользователя, UID и другие данные.
- 2) Создаётся домашний каталог, в котором применяются соответствующие права доступа.
- 3) Вся система получает обновление конфигурационных файлов, а в домашний каталог помещаются файлы инициализации командной оболочки.

Поскольку создание пользователей подразумевает изменения в системных файлах, создать пользователя можно, только обладая root-правами. Для того, чтобы создать пользователя, наберите:

```
$ adduser username
```

Чтобы удалить пользователя, воспользуйтесь командой `userdel`:

```
$ userdel username
```

Пользователей можно объединять в группы – так гораздо удобнее устанавливать права доступа. Чтобы сразу создать пользователя, принадлежащего к группе, нужно указать флаг – `ingroup`:

```
$ adduser username --ingroup groupname
```

Создать новую группу можно с помощью команды:

```
$ groupadd groupname
```

За группы отвечает файл `etc/group` – в нём появится запись о созданной группе. Теперь можно добавить в неё существующих пользователей:

```
$ usermod -aG groupname username
```

Для того, чтобы сменить пользователя, наберите

```
$ su username
```

или

```
$ su - username
```

Задание

Посмотрите структуру команды `adduser` и создайте двух обычных пользователей `<ваше_имя_на_латинице>[1-2]` – например, `ivanov1` и `ivanov2`. Также вам надо будет задействовать своего основного пользователя – того, которого вы создали при установке системы в ходе выполнения первой лабораторной работы. Можно открыть больше одного терминала (`Ctrl+Alt+F[1-5]`), переключаясь между ними при помощи `Alt+F[1-5]`; сделайте так, чтобы в каждом терминале шла работа от имени соответствующего пользователя. Дайте основному пользователю действовать от имени пользователя `root` командой:

```
$ sudo su
```

Создайте группу `Students` и поместите туда двух созданных пользователей. От имени созданных пользователей проверьте с помощью команды `groups`, что пользователи действительно состоят в группе. Под первым пользователем поменяйте себе пароль при помощи команды `passwd`. Под вторым пользователем попробуйте зайти в каталог `/root` и объясните результат. Основным пользователем выйдите из `root`-прав командой `exit`. Этой же командой можно выйти из сессии от имени любого другого пользователя.

Файлы

Обычно владельцем файла является создавший его пользователь. У файлов есть два типа владельцев: пользователи и группы. Под первым пользователем создайте файл `schedule`, куда при помощи редактора `nano` поместите следующий текст:

```
10:00 Maths
11:40 Physics
13:30 Economics
15:20 Programming
```

Затем назначьте владельцем этого файла группу `Students` через команду

```
$ chgrp Students schedule
```

Убедитесь, что файл доступен пользователю 2, зайдя под его учётной записью. Попробуйте добавить в этот файл строку:

17:00 Databases

Под основным пользователем создайте файл Important, где будет записан текст:

```
All students need to be in recreation area at 18:00!  
Sign below if you got this message.
```

Нам нужно, чтобы все лица, которым предназначено это объявление, могли подтвердить прочтение, отписавшись в нём. Надо сделать его доступным остальным пользователям через изменение прав доступа при помощи утилиты `chmod`.

Следует обратить внимание, что в Unix-системах есть три типа доступа:

R – read, доступ к чтению файла или просмотру содержимого каталога.

W – write, доступ к записи в файл или добавлению файлов в каталог.

X – execute, доступ к запуску исполняемого файла или возможность зайти в каталог с помощью команды `cd`.

По команде `ls` с ключом `-la` вы увидите права доступа к файлам вида:

```
drw-r-r- 2 adam adam 96 2007-09-05 18:04 blob  
drwxr-xr-x 2 adam adam 176 2007-09-04 15:57 tapety  
-rw-r-r- 1 adam adam 125 2007-08-29 18:31 fme.py
```

Символ `d` в начале означает директорию, а «`-`» – файл. Записи прав доступа приводятся в следующем порядке: **Пользователи** (Users), **Группы** (Groups) и **Остальные** (Others). То есть запись

```
drw-r-r- 2 adam adam 96 2007-09-05 18:04 blob
```

можно расшифровать как «папке blob разрешены чтение и запись для пользователей, чтение для групп и чтение для остальных». Команда на изменение прав выглядит, например, так:

```
$ chmod u+x script.sh
```

Здесь мы добавляем у файла `script.sh` доступ на исполнение для пользователей. Группы и остальные обозначаются как `g` и `o` соответственно. Права можно как добавлять (+), так и удалять (-).

Итак, вернёмся к основному пользователю. Он не состоит в группе Students, поэтому ему нужно сделать этот файл доступным для записи остальным (Others). Воспользуйтесь командой `chmod` для этого, но сперва **запомните, какие права есть у файла по умолчанию**. Права на файл должны быть следующими (дата и время для примера):

```
-rw-r--rw- <user> <user> 4 2022-10-10 16:24 Important
```

Зайдя под первым и вторым пользователями, проверьте доступность файла и отпишитесь в конце этого файла о получении сообщения. Снова дайте основному пользователю root-права и создайте файл `very important!` с текстом:

```
To all personnel! University will be closed tomorrow!
```

Посмотрите права на этот файл через `ls -la`. Чем они отличаются от стандартных прав на файл, созданный пользователем?

Процессы

Как и в Windows, в Unix-системах существует браузер процессов, запущенных в системе. Чтобы вывести на экран список действующих процессов, наберите

```
$ ps
```

Список процессов можно фильтровать, к примеру:

- a # Выдать все процессы системы, включая лидеров сеансов.
- d # Выдать все процессы системы, кроме лидеров сеансов.
- e # Выдать все процессы системы.
- o # Определяет формат вывода в виде списка полей, разделенных символом «,»
- u # Выдать процессы, принадлежащие указанному пользователю.

Процессами можно управлять. К примеру, можно запускать программы в фоновом режиме, указывая после имени программы символ `&`. Чтобы перевести процесс из фонового режима в основной, нужно воспользоваться командой `fg`, а наоборот – `bg`. Наконец, командой `kill` можно завершить процесс, а `killall` завершает всю группу.

Задание

Здесь можно пользоваться только одним терминалом.

- 1) Используя перенаправление, выведите список процессов от имени пользователя `root` в файл `proc_list`. Приложите его к отчёту.
- 2) Проверьте команду `kill -9 1`.

- 3) В окне терминала запустите программу `firefox`. Изучите команду `ps tree` и получите с помощью неё информацию о дереве процессов и их идентификаторах. Найдите поддереву для процесса `firefox`, изучите список составляющих его процессов.
- 4) Завершите поддереву процессов `firefox` при помощи `killall`.

Контрольные вопросы

- 1) Какими правами в операционной системе Linux обладает супер-пользователь?
- 2) В чем отличия команд `su` и `sudo`?
- 3) С помощью какой команды можно посмотреть дерево процессов?
- 4) С помощью какой команды изменяются права на доступ к файлами и директориям?

Лабораторная работа №5

Настройка пользовательских демонов с применением `systemd`

Введение

Демон (`daemon`) – термин, пришедший из UNIX систем и означающий программу, которая работает в фоне и не взаимодействует с пользователем напрямую. Такие программы обычно запускаются при старте системы, по таймеру или событию. Для того, чтобы настроить и контролировать запуск и работу демонов в современных дистрибутивах, используется системный менеджер `systemd`.

Для настройки демонов `systemd` использует юнит-файлы (юниты). Фактически они являются обычными текстовыми файлами. Каждый файл отвечает за отдельного демона. Кроме того, существуют специальные юниты, которые позволяют объединять демоны в группы, настраивать зависимости демонов друг от друга, задавать таймеры и т. д.

Юниты могут располагаться в трех местах файловой системы:

- `/etc/systemd/system` – здесь располагаются пользовательские юниты.
- `/run/systemd/system` – здесь располагаются юниты, генерирующиеся «на лету», т. е. в рантайме.
- `/usr/lib/systemd/system` – здесь располагаются юниты из установленных пакетов стороннего программного обеспечения (`mysql`, `nginx` и т. д.).

Юниты делятся на разные типы, которые отличаются друг от друга расширением файла. Примеры юнитов:

- `.service` – основной файл, отвечающий за запуск конкретного демона
- `.target` – файл, выполняющий две функции – синхронизация работы демонов и их группировка. На эти файлы можно ссылаться внутри `.service` файлов, задавая зависимость демона от других демонов или от уровней запуска ОС.
- `.timer` – файл для настройки запуска заданного демона по расписанию.

- `.path` – файл настройки запуска демона по событию в файловой системе (например, появление папки, изменение файла и т.д.)
- `.device` – файл настройки запуска демона по событию подключения устройства.

Стоит отметить, что `systemd` – это не одна утилита, а система с большим количеством компонентов. Существуют несколько утилит командной оболочки для работы с `systemd` из терминала:

- `systemctl` – позволяет включать/выключать демонов, смотреть их статус
- `journalctl` – утилита просмотра логов юнитов
- `systemd-notify` – утилита для записи статуса сообщений
- `systemd-analyze` – утилита для анализа времени загрузки демонов
- `cgls`, `cgtop`, `loginctl`, `nspawn` и другие

Важно! Далее по тексту, где указываются команды, которые необходимо вводить в терминал, знак `$` вводить не нужно. Это касается только тех мест, где `$` находится в самом начале строки.

Полезные ресурсы:

- <https://habr.com/ru/post/503816/> – немного истории и бурного обсуждения по теме `systemd`.
- <https://highload.today/systemd-linux/> – выжимка о возможностях `systemd`.
- <https://linuxrussia.com/systemd-create-own-unit.html> – короткая справка по `systemd`.
- <https://sysadminium.ru/adm-serv-linux-systemd-target/> – про типы таргетов в `systemd`.
- <https://wiki.archlinux.org/title/systemd/Timers> – про настройку таймеров в `systemd`.

Описание лабораторной работы

Цель работы

Получение практических навыков по работе с подсистемой инициализации и управления службами.

Указания к выполнению работы

Работа состоит из четырех частей. В первой части осуществляется подготовка программного окружения и генерация варианты работы, во второй части создается bash-скрипт, в третьей части создается юнит файл, в четвертой части запускается пользовательский демон.

Оформление отчета

Отчёт должен быть составлен в соответствии с требованиями, представленными во введении. Отчет необходимо сопроводить несколькими снимками экрана. К отчету необходимо приложить файл `record_result_lab_5`. Это текстовый файл с выполненными командами.

Ход работы

Подготовка

- 1) Запустите виртуальную машину и дождитесь загрузки ОС.
- 2) Откройте терминал. Это можно сделать с помощью основного меню или сочетанием клавиш **Ctrl+Alt+T**. Для запуска через основное меню нажмите на Menu → System Tools → Mate Terminal.
- 3) **Запустите запись терминала. Это важно!** Именно файл с записью истории ваших команд будет основной частью отчета. Для этого введите и запустите команду:

```
$ script -a record_result_lab_5
```

- 4) Убедитесь, что команда выполнилась без ошибок. Введите нижеуказанную команду и найдите имя файла `record_result_lab_5` в ее выводе:

```
$ ls -l
```

Генерация варианта

- 1) Поместите на рабочий стол файл `lab5.py` (см. приложение В), прилагаемый к этому руководству, а также прилагаемый архив `faces.tar.gz`.
- 2) Вернитесь в терминал, в котором уже идет запись (см. Подготовка).
- 3) Перейдите на рабочий стол. Для этого:

```
$ cd ./Desktop
```

в случае с русским языком нужно использовать экранирование пробела:

```
$ cd ./Рабочий\ стол/
```

- 4) Распакуйте архив на рабочий стол. Для этого:

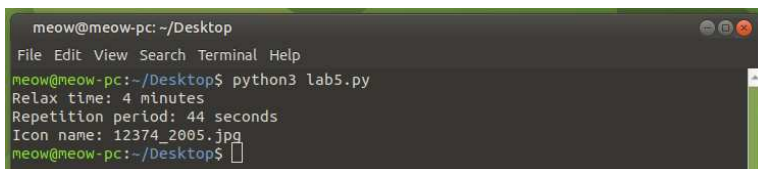
```
$ tar -xvzf faces.tar.gz
```

- 5) Выведите список всех файлов и каталогов. Убедитесь, что в выводе есть файл `lab5.py` и директория `images`:

```
$ ls -l
```

- 6) Запустите генерацию вашего варианта. Нашей задачей будет написать демон, представляющий собой программу для отдыха глаз. Демон будет показывать уведомление о начале отдыха раз в заданный промежуток времени (`repetition_period`) и уведомление об окончании отдыха через заданный промежуток времени (`relax_time`). Пример генерации варианта на рисунке 5.1. Результат генерации необходимо добавить в отчет. Для генерации:

```
$ python lab5.py
```



```
meow@meow-pc: ~/Desktop
File Edit View Search Terminal Help
meow@meow-pc:~/Desktop$ python3 lab5.py
Relax time: 4 minutes
Repetition period: 44 seconds
Icon name: 12374_2005.jpg
meow@meow-pc:~/Desktop$
```

Рисунок 5.1 — Результат генерации варианта

Создание *bash*-скриптов

- 1) Напишем *bash*-скрипт, запуск которого потом настроим в `systemd`. Для этого перейдем на рабочий стол, создадим файл скрипта и сделаем его исполняемым:

```
$ cd ~/Desktop
```

```
$ touch eyes-start.sh
```

```
$ chmod +x eyes-start.sh
```

- 2) Откроем файл скрипта в редакторе `nano`:

```
$ nano eyes-start.sh
```

- 3) Напишем в файл нашу программу:

```
#!/bin/bash
```

```
period_s=$1
```

```
period_ms=$((period_s*1000))
```

```
icon=/absolute/path/to/icon
```

```
notify-send -i $icon -t $period_ms "Time to relax,  
senpai!" "$period_s seconds"
```

Рассмотрим принцип работы этой программы.

```
#!/bin/bash
```

Это так называемый шебанг. Шебанг позволяет указать загрузчику программы, какой интерпретатор (командную оболочку) использовать для исполнения данной программы.

```
period_s=$1
```

Создаем переменную с именем `period_s` и присваиваем ей значение первого параметра, передаваемого нашему скрипту из командной оболочки. Например, при вызове `./eyes_start.sh 10` переменная `period_s` внутри скрипта будет равна 10.

```
period_ms=$((period_s*1000))
```

Преобразуем значение секунд в миллисекунды. Синтаксическая конструкция вида `$((арифметическое_выражение))` позволяет подставлять в текст скрипта результат выполнения арифметического выражения в скобках.

```
icon=/absolute/path/to/icon
```

Переменной `icon` необходимо присвоить абсолютный путь до изображения соответственно вашему варианту. **Внимание!** Не пишите `/absolute/path/to/icon`, напишите настоящий путь к файлу на вашей машине.

```
notify-send -i $icon -t $period_ms "Time to relax,  
senpai!" "$period_s seconds"
```

Команда `notify-send` – это утилита, которая позволяет генерировать всплывающие уведомления на рабочем столе. Ключ `-i` позволяет задать картинку, располагающуюся в углу уведомления. Ключ `-t` позволяет задать время отображения уведомления. Далее в кавычках идет основной текст уведомления. Затем в кавычках идет дополнительный текст уведомления.

- 4) Убедимся в работоспособности программы, запустив ее вручную:
`$./eyes-start.sh 10`
- 5) Если в консоль не вывелись ошибки, а на рабочем столе вы увидели уведомление (см. рисунок 5.2), то можно продолжать. Если этого не произошло, значит, что-то пошло не так.

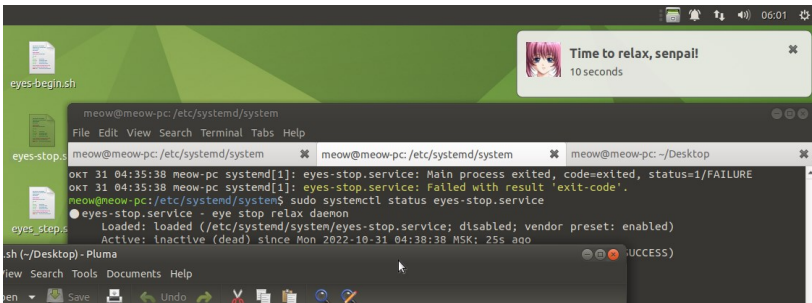


Рисунок 5.2 – Всплывающее уведомление

- 6) Теперь самостоятельно напишите второй скрипт. Текст получившейся программы добавьте в отчет. Скрипт должен соответствовать следующим требованиям:
 - а) Имя скрипта `eyes-stop.sh`.
 - б) Работа скрипта должна начинаться с паузы длительностью `relax_time` (значение взять из варианта). Вам пригодится команда `sleep`.
 - в) Затем на рабочий стол должно вывестись уведомление с текстом "Time to get back to work, senpai!" и иконкой в соответствии с вариантом.
 - г) Длительность показа уведомления 5 секунд.
 - д) Не забудьте сделать файл исполняемым.

Создание юнит-файлов

- 1) Настроим запуск наших скриптов с использованием `systemd`. Это позволит нам запускать наши скрипты как демонов. Для начала перейдем в папку с юнит-файлами:

```
$ cd /etc/systemd/system
```

- 2) Создадим юнит-файл и сделаем его исполняемым:

```
$ sudo touch eyes-relax.service
```

```
$ sudo chmod +x eyes-relax.service
```

- 3) Откроем его в редакторе:

```
$ sudo nano eyes-relax.service
```

- 4) Узнаем несколько параметров системы, которые пригодятся дальше.

```
$ xauth
```

Вывод будет примерно следующего содержания:

```
Using authority file /home/meow/.Xauthorit # Запомните  
этот путь (у вас он будет отличаться)
```

- 5) Узнаем адрес системной шины межпроцессорного взаимодействия.

```
$ echo $DBUS_SESSION_BUS_ADDRESS
```

Вывод будет примерно следующего содержания:

```
unix:path=/run/user/1000/bus # Запомните этот адрес  
(у вас он будет отличаться)
```

- 6) Напишем в файл конфигурацию нашего демона. Вместо <ваше имя> напишите имя своего пользователя.

```
[Unit]
```

```
Description=eye relax daemon
```

```
After=eyes-stop.service
```

```
Wants=eyes-stop.service
```

```
[Service]
```



```

Type=simple
Environment="DISPLAY=:0"
Environment="XAUTHORITY=/home/meow/.Xauthority" # Сюда
записать путь из пункта 4
Environment="DBUS_SESSION_BUS_ADDRESS=
unix:path=/run/user/1000/bus" # Сюда записать адрес
из пункта 5
User=<ваше имя>
Group=<ваше имя>
WorkingDirectory=/home/<ваше_имя>/Desktop/
ExecStart=/home/<ваше_имя>/Desktop/eyes-start.sh 10
# Вместо 10 поставьте сюда значение relax_time из вашего
варианта

```

Рассмотрим структуру юнита подробнее. Файл содержит несколько секций:

- **Unit** – обязательный раздел, который содержит текстовое описание демона и его зависимость от других демонов (юнитов).
- **Service** – обязательный раздел, который конфигурирует, что запускать (какую программу) и как запускать.
- **Install** – обязательный раздел, который может быть вынесен в другие специальные юнит-файлы (например, в файл таймера). В файле конфигурируется режим ОС (уровень запуска), в котором запускается демон. Например, это может быть режим с графикой, сетевой режим без графики, режим перезагрузки и т. д.

Рассмотрим атрибуты раздела Unit:

- **Description** – текстовое описание демона.
- **Wants** – список демонов, активация которых желательна для работы данного демона.
- **After** – список демонов, после которых должен запуститься данный демон.

Рассмотрим атрибуты Service:

- **Type** – описывает, как запускается демон. Есть несколько вариантов: **simple** (по умолчанию) – **systemd** ожидает, что

служба запустится незамедлительно. Процесс не должен разветвляться. `forking` – после запуска демон отвечается (делает форк), родительский процесс завершается. Такой подход используется для запуска классических демонов. `oneshot` – одноразовое выполнение. Используется для скриптов, которые запускаются и завершаются после выполнения.

- `Environment` – список переменных окружения.
- `WorkingDirectory` – рабочая директория. Становится текущей перед запуском стартовой программы.
- `ExecStart` — команда для старта демона (программа демона).

- 7) Чтобы обеспечить периодичность запуска нашего демона, создадим специальный юнит `eyes-relax.timer`. Его название должно совпадать с `service`-файлом, созданным ранее, благодаря этому они будут связаны.

```
$ sudo touch eyes-relax.timer
$ sudo chmod +x eyes-relax.timer
$ nano eyes-relax.timer
```

- 8) Напишем в файл конфигурацию таймера. В поле `OnUnitActiveSec` укажите значение времени из варианта (`repetition_period`). В отчете опишите, за что отвечают секции и поля в файле таймера.

```
[Unit]
```

```
Description=scheduler for eyes service
```

```
[Timer]
```

```
OnBootSec=60s
```

```
OnUnitActiveSec=50s
```

```
[Install]
```

```
WantedBy=timers.target graphical.target
```

- 9) Создадим юнит-файл для написанного вами скрипта (см. раздел Написание `bash`-скриптов, пункт 6).

```
$ sudo touch eyes-stop.service
$ sudo chmod +x eyes-stop.service
$ nano eyes-stop.service
```

10) В созданный файл написать:

```
[Unit]
Description=eye stop relax daemon
Requires=eyes-start.service

[Service]
Type=simple
Environment="DISPLAY=:0" Environment=
"XAUTHORITY=/home/meow/.Xauthority" # Сюда записать путь
из пункта 4
Environment="DBUS_SESSION_BUS_ADDRESS=
unix:path=/run/user/1000/bus" # Сюда записать адрес из
пункта 5
User=<ваше имя>
Group=<ваше имя>
WorkingDirectory=/home/<ваше_имя>/Desktop/
ExecStart=/home/<ваше_имя>/Desktop/eyes-stop.sh

[Install]
WantedBy=graphical.target
```

Запуск демона

Теперь, когда готовы все нужные файлы, необходимо научиться управлять демоном.

- 1) Итак, сперва перезагрузим всех демонов командой

```
$ sudo systemctl daemon-reload
```
- 2) В ходе выполнения лабораторной работы мы создали трех демонов – таймер (`eyes-relax.timer`), демона, который уведомляет о

начале отдыха (`eyes-relax.service`), и демона, который ждет заданное время и уведомляет об окончании отдыха (`eyes-stop.service`).

- 3) Теперь запустим стартового демона, который, в свою очередь, активирует выполнение остальных. Из созданной нами логической структуры можно понять, что таким демоном является таймер.

```
$ sudo systemctl start eyes-relax.timer
```

- 4) Проверим статусы всех демонов:

```
$ sudo systemctl status eyes-relax.timer
```

```
$ sudo systemctl status eyes-relax.service
```

```
$ sudo systemctl status eyes-stop.service
```

- 5) Чтобы установить демон в автозагрузку:

```
$ sudo systemctl enable eyes-relax.timer
```

- 6) Убедитесь, что демон работает, перезагрузив виртуальную машину.

- 7) Чтобы остановить демон, воспользуйтесь командой:

```
$ sudo systemctl stop eyes-relax.timer
```

- 8) Если демон вам больше не нужен, можно отключить автозагрузку командой:

```
$ sudo systemctl disable eyes-relax.timer
```

Контрольные вопросы

- 1) Что такое демон?
- 2) Для чего в операционной системе Linux применяется подсистема `systemd`?
- 3) Какие типы юнит-файлов вы знаете?
- 4) С помощью какой команды осуществляется управления демонами в подсистеме `systemd`?

Лабораторная работа № 6

Работа с сетями и настройка сервера

Введение

Локальная вычислительная сеть — сеть компьютеров, охватывающая, как правило, небольшую территорию (здание, комплекс зданий, микрорайон и т. д.). Локальные сети могут быть спроектированы одноранговыми. Они работают в случае, когда компьютеров в сети немного, а если их 10 или более, то такая сеть будет непроизводительной. Поэтому чаще всего приходится иметь дело с серверами и централизованной архитектурой, построенной вокруг них. ОС Linux — частое решение для серверных оболочек, на её основе можно легко сконфигурировать сервер под свои нужды. В данной работе мы рассмотрим основы работы с утилитами `ping` и `ifconfig`, разберёмся с установкой и настройкой сервера `nginx`.

Полезные ссылки:

- Настройка `nginx`: <https://highload.today/nginx/>.
- Руководство по Ubuntu Server: <https://launchpadlibrarian.net/135831926/serverguide-precise-ru.pdf>.
- Про SSH и его настройку: <https://help.reg.ru/support/server-vps/oblachnyye-servery/rabota-s-serverom/kak-ustanovit-i-nastroit-ssh>.

Необходимое ПО:

- Любая ОС, на которую можно установить VirtualBox.
- Образ Ubuntu Server версии 16.04 или выше.

Описание лабораторной работы

Цель работы

Получение практических навыков по установке и настройке web-сервера `nginx`.

Указания к выполнению работы

Работа состоит из четырех частей. В первой части осуществляется подготовка программного окружения, во второй части устанавливается и

настраивается операционная система Ubuntu Server, в третьей части настраивается удаленный доступ по протоколу `ssh`, в четвертой части устанавливается и настраивается web-сервер `nginx`.

Оформление отчета

Отчёт должен быть составлен в соответствии с требованиями, представленными во введении. В отчете опишите процесс выполнения работы, сопроводив его снимками экрана.

Ход работы

Подготовка

- 1) Запустите виртуальную машину и дождитесь загрузки ОС.
- 2) Откройте терминал. Это можно сделать с помощью основного меню или сочетанием клавиш **Ctrl+Alt+T**. Для запуска через основное меню нажмите на Menu → System Tools → Mate Terminal.
- 3) **Запустите запись терминала. Это важно!** Именно файл с записью истории ваших команд будет основной частью отчета. Для этого введите и запустите команду:

```
$ script -a record_result_lab_6
```

- 4) Убедитесь, что команда выполнилась без ошибок. Введите нижеуказанную команду и найдите имя файла `record_result_lab_6` в ее выводе:

```
$ ls -l
```

Утилита ping

Основное назначение утилиты `ping` заключается в проверке состояния соединения между узлами. При запросе эта утилита посылает пакеты на указанный IP-адрес, замеряя время до полученного ответа или фиксирующая его отсутствие. В терминале наберите:

```
$ ping habr.com
```

Мы получим количество отправленных байт, время ответа в миллисекундах и время жизни пакета (TTL, Time To Live). У утилиты `ping` есть параметры, которые настраивают её поведение. Изучите список параметров и выполните следующее:

- 1) Наберите `ping -D <адрес>`. Что изменилось?
- 2) Установите TTL пакета в 150 мс.
- 3) Пошлите 3 пакета с интервалом в 7 секунд.

Утилита ifconfig/ip

Утилита `ifconfig` в новых дистрибутивах заменена на утилиту `ip` и может не присутствовать в новых дистрибутивах Linux. Для того, чтобы её поставить, выполните:

```
$ sudo apt install net-tools -y
```

Данная утилита предназначена для настройки и проверки состояния сетевых интерфейсов. Выполните:

```
$ ifconfig -a
```

Утилита выведет список всех сетевых интерфейсов на данной машине и их параметры. Можно написать после команды название конкретного интерфейса и вывести только его параметры. Выведите параметры интерфейса `esp0s3`. **Внимание! В зависимости от дистрибутива Linux названия устройств могут отличаться!**

В утилите `ip` схожие команды будут иметь вид:

```
$ ip a
```

```
$ ip a show esp0s3
```

Ubuntu Server

Ubuntu Server – это специальная версия ОС, которая предназначена для работы на серверном оборудовании. Как правило, она не имеет графической оболочки, а все утилиты работают посредством командной строки. Её установка ничем не отличается от установки Ubuntu Mate, однако стоит обратить внимание на следующий экран (см. рисунок 6.1):

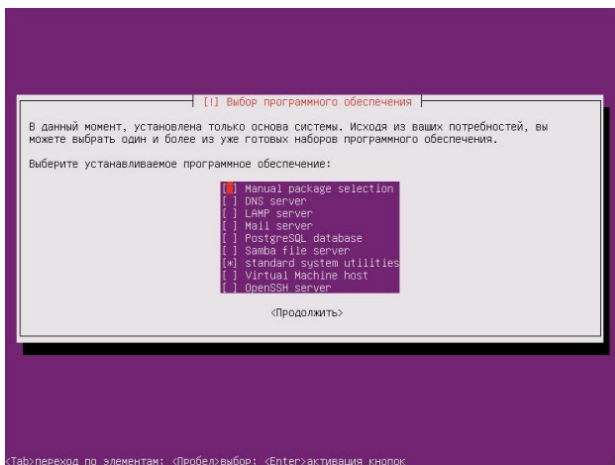


Рисунок 6.1 – Список сетевых надстроек

Здесь мы можем сразу настроить роль сервера, установив соответствующие пакеты. Роль сервера определяется кругом задач, которые он будет выполнять. Он может хранить данные об учётных записях, быть файловым хранилищем, обрабатывать почту и так далее. Роль сервера задаётся установкой и конфигурацией специализированных программных пакетов. 4 общих этапа:

- 1) Установка серверного пакета ПО через команду:
`$ sudo apt-get install <package_name>`
- 2) Создание текстового файла конфигурации сервера (различны в зависимости от роли).
- 3) Запуск сервера командой `sudo <server_name> start`.
- 4) Установка и настройка клиента сервера.

Настройка Виртуальной машины

Здесь мы попробуем организовать подключение по SSH. Но прежде, чем мы перейдём к установке и настройке OpenSSH, необходимо настроить саму виртуальную машину. **Внимание! Данный пункт 6 лабораторной работы ориентирован на тех, кто запускает Ubuntu через Oracle VirtualBox.**

В VirtualBox существует несколько типов подключения к сети Интернет:

- *не подключен* – сетевой адаптер может отображаться, но сетевой кабель отображается неподключенным;
- *NAT (Network Address Translation)* – режим по умолчанию, подключение идёт через маршрутизатор, в роли которого выступает сетевой модуль VirtualBox;
- *Сетевой мост* – данные проходят через физическое устройство хостового компьютера;
- *Виртуальный адаптер хоста* – используется для создания сети из хоста и нескольких виртуальных машин через виртуальный интерфейс.

Есть и другие режимы, но они используются редко (см. рисунок 6.2).

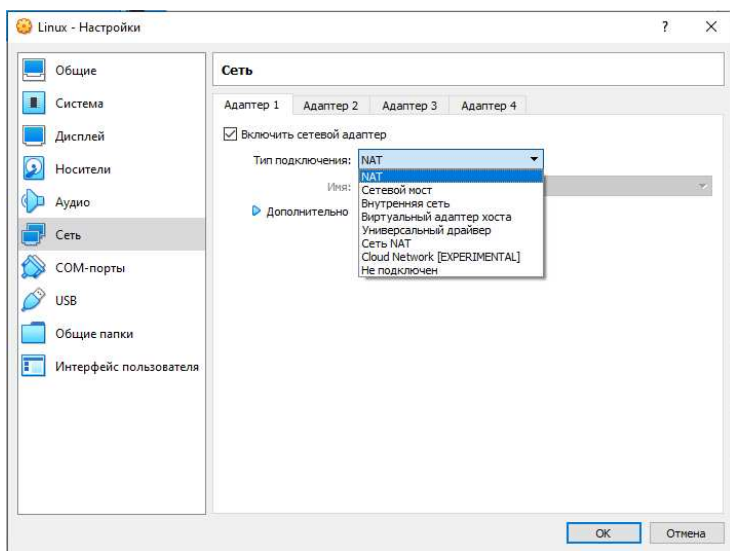


Рисунок 6.2 – раздел «Сеть»

Подключение по SSH будет организовано через второй адаптер. VirtualBox поддерживает до 4 адаптеров. Надо зайти в настройки виртуальной машины, кликнув по жёлтой шестерёнке, как показано на рисунке 6.3. Откроется окно, которое мы уже видели на рисунке 6.2.

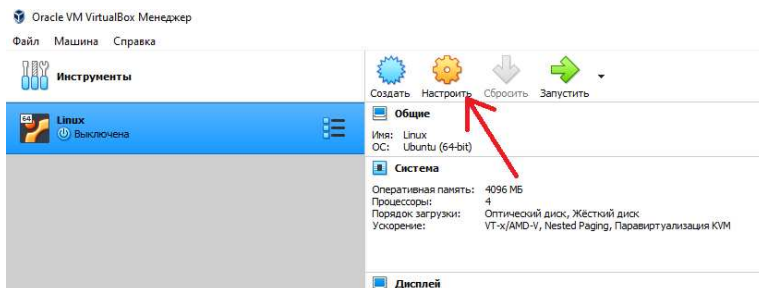


Рисунок 6.3 – Кнопка «Настройки» для виртуальной машины

Первая вкладка «Адаптер 1» - оставляем, как есть (тип подключения – NAT, см. рисунок 6.4).

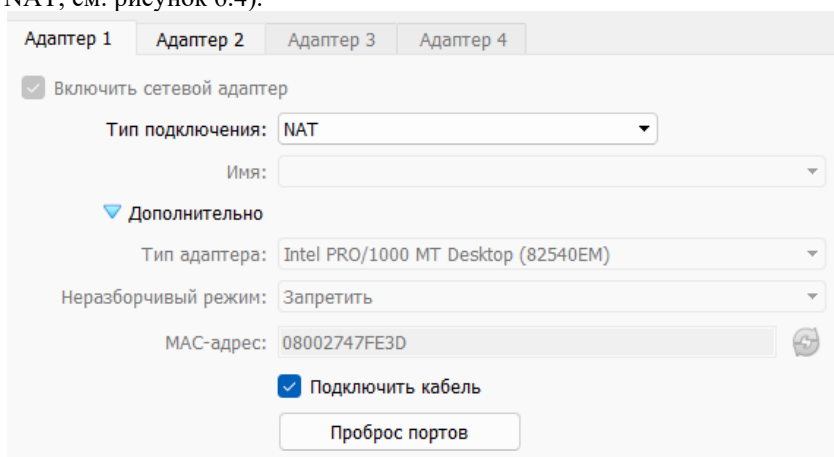


Рисунок 6.4 – Настройка адаптера 1

Переходим во вкладку «Адаптер 2». Включаем адаптер, тип подключения «Виртуальный адаптер хоста», имя – «VirtualBox Host-Only Ethernet Adapter» (см. рисунок 6.5).

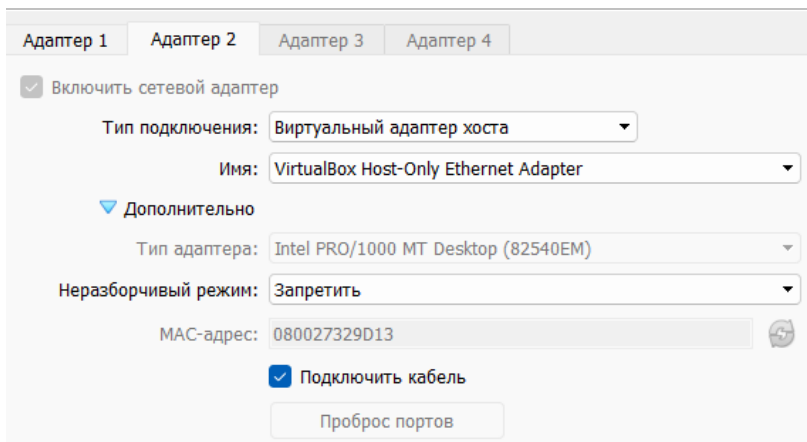


Рисунок 6.5 – настройка адаптера 2

Нажимаем «OK». Настройка VirtualBox завершена. Запустите виртуальную машину и проверьте соединение при помощи утилиты `ping`.

Подключение по SSH

SSH (Secure Shell) – протокол для удалённого управления операционной системой. Чаще всего подключение по SSH используется для устройств, не обладающих дисплеем: микрокомпьютеры типа Raspberry Pi, серверные станции, разного рода удаленное оборудование. Сперва попробуем настроить подключение по SSH. Для начала необходимо установить ssh-сервер командой:

```
$ apt-get install openssh-server
```

Если `ssh` не установлен, то начнётся его установка. Далее начинаем настройку. Открываем файл `/etc/network/interfaces` и помещаем туда следующее:

```
source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback
```

```
# The primary network interface
auto enp0s3
iface enp0s3 inet dhcp
```

```
auto enp0s8
iface enp0s8 inet static
    address 192.168.56.10
    netmask 255.255.255.0
```

Подключение к Ubuntu будет выполняться посредством сетевого устройства `esp0s8` по адресу `192.168.56.10`. Мы его добавили как «Адаптер 2» в VirtualBox ранее. Далее назначим пароль для пользователя `root`. Набираем:

```
$ sudo passwd
```

Вводим и повторяем пароль для `root`. **Запишите и запомните его!** Затем нам нужен файл `/etc/ssh/sshd_config`:

```
$ sudo nano /etc/ssh/sshd_config
```

Ищем там строку:

```
#PermitRootLogin prohibit-password
```

Раскомментируем её, убрав символ `#` в начале строки, а вместо `prohibit-password` пишем `yes`:

```
PermitRootLogin yes
```

Далее перейдём в Windows, нужно открыть PowerShell (от имени администратора) и набрать:

```
Install-Module -Name Posh-SSH
```

Начнётся установка клиента, при необходимости вводим `Y` для продолжения установки. После окончания перезапускаем powershell. Теперь можно попробовать подключиться:

```
$ ssh root@192.168.56.10
```

При успешном подключении powershell попросит пароль от `root`, который мы установили ранее (см. рисунок 6.6).

```
root@sergei-VirtualBox: ~
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Установите последнюю версию PowerShell для новых функций и улучшений! https://aka.ms/PSWindows

PS C:\WINDOWS\system32> ssh root@192.168.56.10
The authenticity of host '192.168.56.10 (192.168.56.10)' can't be established.
ECDSA key fingerprint is SHA256:U9hcmlp3hYFvj4hyZmGGNHFL9QsQPzbW02yvwerbVFU.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.56.10' (ECDSA) to the list of known hosts.
root@192.168.56.10's password:
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-48-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

87 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@sergei-VirtualBox:~#
```

Рисунок 6.6 – Вывод при успешном подключении по SSH

Готово! Теперь можно работать в терминале Ubuntu удаленно.

Настройка сервера nginx

В этой части работы мы попробуем установить и сконфигурировать веб-сервер **nginx**. **Nginx** – это веб-сервер с открытым исходным кодом, созданный для работы под высокой нагрузкой, чаще всего используемый для отдачи статического контента, например html-страниц, медиафайлов, документов, архивов, картинок и т.д. или в роли прокси для другого серверного ПО, например, Samba.

```
$ sudo apt update
```

```
$ sudo apt install nginx
```

Дожидаемся окончания установки. После этого можно проверить работу **nginx**, перейдя в браузер и набрав адрес, в данном случае: 192.168.56.10. Если вы видите этот текст, то всё в порядке (см. рисунок 6.7):

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Рисунок 6.7 – Приветственное окно веб-сервера nginx

Для запуска `nginx` нужно выполнить одноименный исполняемый файл. Также с помощью флага `-s` можно управлять состоянием сервера, к примеру:

```
$ nginx -s stop # моментально остановить
$ nginx -s quit # завершить работу
$ nginx -s reload # перезапустить конфигурационный файл
после внесения изменений
```

Просмотреть список процессов `nginx` можно командой:

```
$ ps -ax | grep nginx
```

Nginx обладает модульной структурой. Каждый модуль настраивается т. н. директивами, которые бывают простыми (имя и параметры, разделённые пробелами) и блочными (имеют вложенные инструкции, окружаются фигурными скобками). Конфигурационный файл имеет иерархическую структуру: каждая директива находится в своём контексте. Чтобы найти расположение конфигурационного файла, выполните:

```
$ locate nginx.conf
```

Перед каждым изменением конфигурации стоит проверить её на валидность командой

```
$ nginx -t
```

Настройка отдачи контента

Основное назначение `nginx` – отдача статического контента. Для этого нужно прописать директивы `location`, которые помещаются в контекст `server`. Переместите файл `lab6.py` (см. приложение Г) на рабочий стол запустите его через:

```
$ python3 lab6.py
```

Создайте папки `/var/www` и `/var/images` и поместите туда несколько файлов: в первую папку поместите созданную `html`-страницу `index.html`, во вторую – несколько изображений в формате `.jpg`. Теперь обратимся к конфигурационному файлу:

```
$ nano /etc/nginx/nginx.conf
```

В директиву `http` добавьте `server` с префиксами:

```
http {  
    server {  
        listen 8080;  
        location / {  
            root /var/www;  
        }  
        location /images/ {  
            root /var;  
        }  
    }  
}
```

Обратите внимание на параметры директив. В первом случае это префикс `/`, во втором – `/images/`. Это и есть условия перенаправления запроса. `Nginx` получает запрос, сравнивает `URL` из запроса с имеющимися префиксами в `location` и, если есть совпадение, перенаправляет в указанный каталог. Сначала `nginx` проверяет префиксы блоков `location`. Он запоминает директиву с самым длинным подходящим префиксом. Затем `nginx` проверяет регулярные выражения. Если обнаружено совпадение, то выбирается соответствующий `location`. Если совпадения нет, запрос идет на `location`, который `nginx` запомнил ранее. Теперь можно обратиться за контентом, набрав в браузере:

```
http://192.168.56.10:8080
```

```
http://192.168.56.10:8080/images/<example>.jpg
```

Сервер идет в каталог `/var/images/` и возвращает файл `example.jpg`. Если такого файла нет, то возвращается ответ с ошибкой 404. Выведите в браузере html-страницу и прочитайте полученный текст. Если вы видите ошибку 403 `Forbidden`, то, скорее всего, у файлов нет прав на открытие. Измените права при помощи утилиты `chmod`.

Контрольные вопросы

- 1) Что такое NAT, для чего он применяется?
- 2) Для чего применяется утилита `ifconfig/ip`?
- 3) В чем отличие систем адресации IPv4 и IPv6?
- 4) Что означает код возврата сервера 403?

Литература

- 1) Брайн Уорд. Внутреннее устройство LINUX. СПб.: Питер, 2016—2019. 384 с.
- 2) Уильям Штос. Командная строка Linux. Полное руководство. – СПб.: Питер, 2017—2019. 480 с.
- 3) Барнет С., Такет Дж. Использование Linux М.: Вильямс, 2000.
- 4) Далхаймер М. К., Кауфман Л., Уэлш М. Запускаем Linux СПб.: Символ-Плюс, 2000.
- 5) Бендел Д., Нейпир Р. Использование Linux М.: Вильямс, 2002.
- 6) Курячий Г. В. Операционная система UNIX: Курс лекций. Учебное пособие М.: ИНТУИТ.РУ, 2004.
- 7) Комолкин А. В., Немнюгин С. А., Чаунин М. П. Эффективная работа с UNIX СПб.: Питер, 2002.
- 8) Петцке К. От понимания к применению М.: ДМК, 2000.
- 9) Робачевский А. Операционная система Unix СПб.: BHV, 1999.
- 10) Немец Э., Сибасс С., Снайдер Г., Хейон Т. UNIX. Руководство системного администратора. / Серия: Для профессионалов СПб.: Питер, 2020. 5-е изд.
- 11) Куль, Т.П. Операционные системы : учебное пособие / Т.П. Куль. – Минск : РИПО, 2015. – 312 с. : ил. – Библиогр. в кн. – ISBN 978-985-503-460-6 ; То же [Электронный ресурс]. – URL: <http://biblioclub.ru/index.php?page=book&id=463629>.
- 12) Назаров, С.В. Современные операционные системы: учебное пособие / С.В. Назаров, А.И. Широков. Москва : Интернет-Университет Информационных Технологий, 2011. – 280 с. : ил., табл., схем. – (Основы информационных технологий). – ISBN 978-5-9963-0416-5 ; То же [Электронный ресурс]. – URL: <http://biblioclub.ru/index.php?page=book&id=233197>.
- 13) Костромин, В.А. Основы работы в ОС Linux [Электронный ресурс] : учебное пособие / В.А. Костромин. – Электрон. дан. – Москва : , 2016. – 810 с. – Режим доступа: <https://e.lanbook.com/book/10033>.
- 14) Кондратьев, В.К. Операционные системы и оболочки : учебно-практическое пособие / В.К. Кондратьев, О.С. Головина ; Международный консорциум «Электронный университет», Московский

- государственный университет экономики, статистики и информатики, Евразийский открытый институт. Москва : Московский государственный университет экономики, статистики и информатики, 2007. – 172 с. – ISBN 5-374-00009-8 ; То же [Электронный ресурс]. – URL: <http://biblioclub.ru/index.php?page=book&id=90663>.
- 15) Гриценко, Ю.Б. Операционные среды, системы и оболочки : учебное пособие / Ю.Б. Гриценко ; Томский межвузовский центр дистанционного образования (ТУСУР). – Томск : Томский государственный университет систем управления и радиоэлектроники, 2005. - 281 с. : табл., схем. ; То же [Электронный ресурс]. – URL: <http://biblioclub.ru/index.php?page=book&id=208656>.
 - 16) Карпов, В. Основы операционных систем : практикум / В. Карпов, К. Коньков. – Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 301 с. : ил. – Библиогр. в кн. ; То же [Электронный ресурс]. – URL: <http://biblioclub.ru/index.php?page=book&id=429022>.
 - 17) Аленичев Д., Боковой А., Бояршинов А. ALT Linux изнутри. ДМК-Пресс. 2009. – 416 с. То же [Электронный ресурс]. - URL: <https://e.lanbook.com/book/1197>.
 - 18) Гончарук, С.В. Администрирование ОС Linux / С.В. Гончарук. - 2-е изд., испр. – Москва : Национальный Открытый Университет «ИНТУИТ», 2016. – 165 с. : ил., табл. – Библиогр. в кн. ; То же [Электронный ресурс]. – URL: <http://biblioclub.ru/index.php?page=book&id=429014>.
 - 19) Пахмурин, Д.О. Операционные системы ЭВМ : учебное пособие / Д.О. Пахмурин ; Министерство образования и науки Российской Федерации, Томский Государственный Университет Систем Управления и Радиоэлектроники (ТУСУР). – Томск : ТУСУР, 2013. – 255 с. : ил. – Библиогр. в кн. ; То же [Электронный ресурс]. – URL: <http://biblioclub.ru/index.php?page=book&id=480573>
 - 20) Э. Таненбаум, Т. Остин. Архитектура компьютера. – СПб.: Питер, 2015.
 - 21) Э. Таненбаум, А. Вудхалл. Операционные системы: Разработка и реализация. – СПб.: Питер, 2006.

- 22) К. Симмондс. Встраиваемые системы на основе Linux. / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2017.
- 23) Э. Таненбаум, Х. Бос. Современные операционные системы. – СПб.: Питер, 2011.
- 24) Tool Interface Standard Executable and Linking Format (ELF) Specification Version 1.2.
- 25) System V Application Binary Interface. Edition 4.1. – <http://www.sco.com/developers/devspecs/gabi41.pdf>.
- 26) J. Corbet, A. Rubini, G. Kroah-Hartman. Linux Device Drivers. 3 rd Edition. – O'Reilly Media, 2005.

Приложение А

Исходный код программы для лабораторной работы №2

```
import os
import pathlib
import random
import string
import shutil
import tarfile

NAME_LEN = 5
MAX_DEPTH = 3
MAX_WIDTH = 3
random_choice = [True, False]

def generate_name():
    name = random.choices(string.digits + string.ascii_letters,
                           k=NAME_LEN)
    name = ''.join(name)
    return name

def make_dir(name):
    path = pathlib.Path.cwd().joinpath(name)
    path.mkdir(parents=True, exist_ok=True)

def generate_nested(depth=1, max_width=MAX_WIDTH,
                    max_depth=MAX_DEPTH):
    start = 2 if depth == 1 else 1
    number = random.randrange(start, max_width)
    for i in range(number):
        name = generate_name()
        make_dir(name)
        os.chdir(f'./{name}')
        if depth >= max_depth:
            if random.choice(random_choice):
                secret = generate_name()
                with open(secret, 'w') as f:
                    f.write(secret)
                random_choice.remove(True)
            os.chdir('..')
        else:
            depth = generate_nested(depth + 1)
```

```

os.chdir('..')
return depth - 1

def generate_text_struct():
    print('Task 2')
    print('Replicate the directory structure below with terminal commands ')
    print('The structure is:')
    name = generate_name()
    make_dir(name)
    startpath = f'./{name}'
    os.chdir(startpath)
    generate_nested()
    for root, dirs, files in os.walk(startpath):
        level = root.replace(startpath, '').count(os.sep)
        indent = '    |' * (level) + '└──' if level != 0 else
        print('{}{}/'.format(indent, os.path.basename(root)))
        subindent = '|──' * (level + 1)
        for f in files:
            print('{}{}/'.format(subindent, f))
    shutil.rmtree(startpath)
    print('\n\n')

def generate_dirs():
    name = generate_name()
    make_dir(name)
    os.chdir(f'./{name}')
    generate_nested()
    print('Task 1')
    print('Investigate a nested directory and draw its structure')
    print(f'Your task directory is "{name}"!')
    print('Good luck!\n\n')

def generate_archive():
    cite = ['Отвлекись от всего, Нео!',
            'Следуй за белым кроликом.',
            'Матрица повсюду. Она окружает нас.',
            'Ты только раб, Нео. Как и все, ты с рождения в цепях.',
            'Ты веришь в судьбу, Нео?',
            'Ложки не существует.',
            'Продам гараж!']

```

```

with open('secret_file', 'w') as f:
    f.write(random.choice(cite))

archive_name = 'task3.tar.gz'
tar = tarfile.open(archive_name, 'w:gz')
for name in ['secret_file']:
    tar.add(name)
tar.close()

os.remove('secret_file')

print('Task 3')
print(f'You have to extract the archive {archive_name}')
print('and find a secret text.')

if __name__ == '__main__':
    generate_dirs()
    generate_text_struct()
    generate_archive()

```

Приложение Б

Исходные коды программ для лабораторной работы №3

Файл lab3.py

```
import sys
import csv
import random

CHUNK_MAX = 20
CHUNK_MIN = 10
LINE_NUM = 769

chunk_size = random.randrange(CHUNK_MIN, CHUNK_MAX)
slice_start = random.randrange(1, LINE_NUM-chunk_size)
slice_end = slice_start + chunk_size

out = [sys.stdout, sys.stderr]
with open('the-matrix-reloaded.csv', newline='') as csvfile:
    dialog_lines = list(csv.reader(csvfile))
    chosen_lines = dialog_lines[slice_start:slice_end]
    for line in chosen_lines:
        random_stream = random.choice(out)
        print(f'{line[0]}: {line[1]}', file=random_stream,
flush=True)
```

Файл lab3_2.py

```
import random

with open('variants.txt', newline='') as f:
    lines = f.readlines()
    sentence = random.choice(lines).split(' ')
    sentence = [word.strip() for word in sentence]
    saying_char = sentence[0]
    about_char = sentence[1]
    subject = sentence[2]
    punc = sentence[3]
    print(f'a) The character that says a phrase is "{say-
ing_char}"')
```


Приложение В

Исходный код программы для лабораторной работы №5

```
import os
import random

relax_period = random.choice(range(10, 20))
repetition_period = random.choice(range(30, 60))

arr = os.listdir('./images')
icon = random.choice(arr)
print(f'Relax time: {relax_period} minutes')
print(f'Repetition period: {repetition_period} seconds')
print(f'Icon name: {icon}')
```

Приложение Г

Исходный код программы для лабораторной работы №6

```
import random

headers = ['Товарищи!', 'Друзья!', 'Коллеги!', 'Однопартийцы!',
           'Господа!', 'Участники заседания!']

fir_parts = ['Несмотря на замедление падения ремиссионного де-
прессирования',
             'Особенно хотелось отметить то, что',
             'Подводя итоги, необходимо сказать о том, что',
             'Не вызывает сомнений факт, что',
             'Не может пройти незамеченным то обстоятельство,
что',
             'В настоящее время главным образом стоит обратить
внимание на то, что',
             'Хотя кадастровая модель важна, не следует забы-
вать о том, что']

sec_parts = ['субсидирование ассигнационной части бюджетной со-
ставляющей',
             'рекуперация ротационных секвестров',
             'регламентация государственных реестров',
             'валоризация базового компонента и варьируемого
сегмента',
             'ликвидация профицита недодефицитирования',
             'уверенное прогрессирование в любом звене указан-
ного диапазона',
             'дефлятирование текущих тарифов']

thi_parts = ['позволяет уверенно говорить о новой вехе стабили-
зации',
             'обнаруживает тенденцию к устойчивому росту',
             'служит толчком для нового витка подъема, который
был сформирован',
             'является квотой и заделом для повышения общего
уровня',
             'имеет место быть в каждой сфере деятельности',
             'демонстрирует определенный экспоненциальный ры-
вок',
             'будет иметь успех только в том случае, когда при-
рост издержек расходов покажет свою лабильность']
```

```

for_parts = ['за истекший период.',
             'на данном этапе развития.',
             'как де-юре, так и де-факто.',
             'на стадии пилотной сегрегации',
             'после прохождения отчётной фазы',
             'по окончании достагнационного промежутка вре-
мени.',
             'во всех своих проявлениях']

```

```

header = random.choice(headers)
fir_part = random.choice(fir_parts)
sec_part = random.choice(sec_parts)
thi_part = random.choice(thi_parts)
for_part = random.choice(for_parts)

```

```

sequence = f'{header} {fir_part} {sec_part}\n {thi_part}
{for_part}'

```

```

index_file = open('index.html', 'w')
index_file.write(f'<html>\n\t<head><meta charset=utf-8><ti-
tle>Вариант {random.randint(1,36)}'
                 f'</title></head>\n\t<body>{se-
quence}</body></html>')

```