Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени первого Президента России Б. Н. Ельцина»

УТВЕРЖДАЮ

Директор по образовательной деятельности

_____ С.Т. Князев

«24» декабря 2021 г.

Операционная система Linux

Учебно-методические материалы по направлению подготовки
09.04.01 Информатика и вычислительная техника
Образовательная программа «Инженерия искусственного интеллекта»

Екатеринбург

2021

РАЗРАБОТЧИКИ УЧЕБНО-МЕТОДИЧЕСКИХ МАТЕРИАЛОВ

Делопроизводитель ООД ИРИТ-РТФ		Токарева Виолетта Михайловна
Ассистент Кафедры информационных технологий и систем управления		Токарев Александр Владимирович

СОДЕРЖАНИЕ

Модуль 1. Введение.

Содержание модуля:
В этом модуле вы познакомитесь с дисциплиной «ОС Linux». Информация в модуле представляет описание большинства задач дисциплины. В рамках дисциплины «ОС Linux» слушатели научатся решать стандартные задачи профессиональной деятельности средствами ОС Linux; создавать и проводить первичную настройку виртуальной машины; устанавливать ОС Linux и выполнять пользовательские настройки; администрировать пользователей ОС Linux; работать с файловой системой в ОС Linux; настраивать сетевое подключение как для выхода в интернет, так и для работы в локальной сети в ОС Linux; проводить настройку среды программирования на Python в ОС Linux . Более подробно эти сведения изучаются в следующих модулях курса.

В этом модуле слушатели:

- узнают, чему посвящена дисциплина «ОС Linux»,
- ознакомятся с программой и формами взаимодействия, системой оценивания,
- изучат технологию создания виртуальной машины и опробуют ее на практике,

Знания, полученные при изучении этого модуля, создадут хорошую основу для понимания структуры следующих модулей. Удачи!

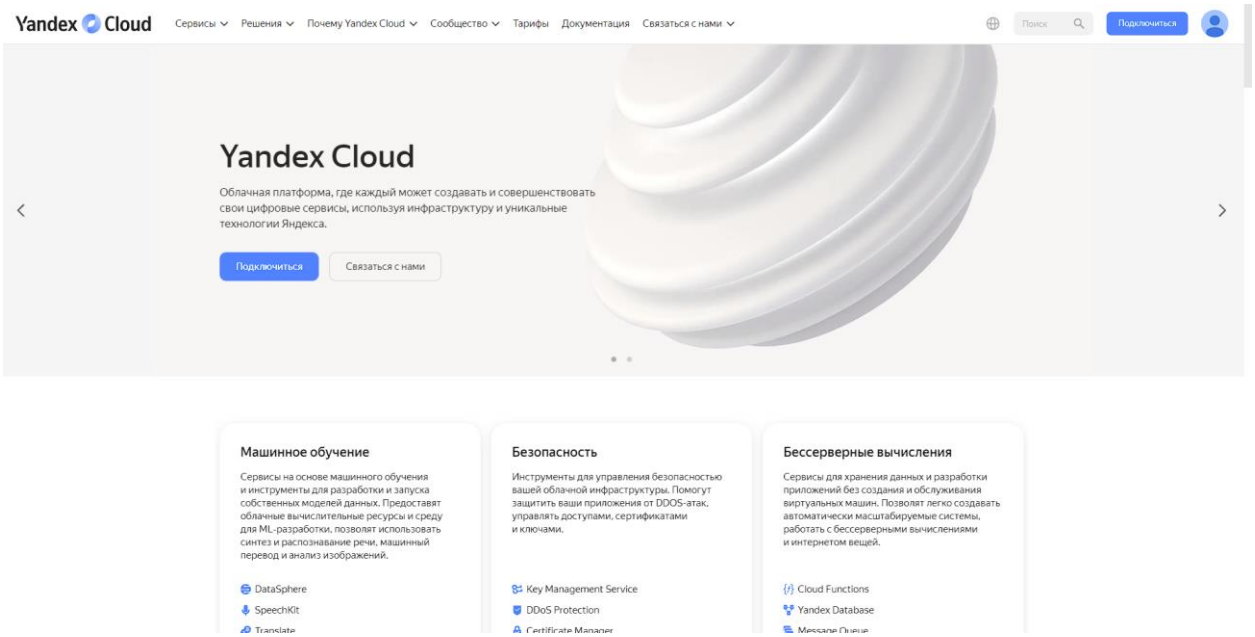
Модуль 1. Юнит 1: Инструкция по созданию виртуальной машины

Что бы создать виртуальную машину на **Yandex**.**Cloud** нужно:

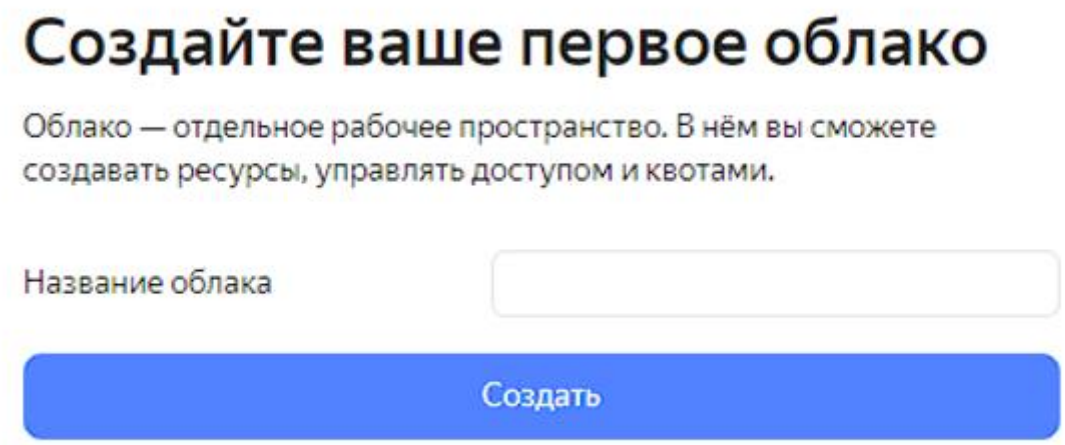
1) Создать аккаунт на Яндекс и перейти в https://cloud.yandex.ru
2) Подключить платежный аккаунт.
3) Создать пары SSH-ключей.
4) Создать виртуальную машину и подключиться к ней.

Подробная последовательность действий:

После того, как был создан аккаунт на Яндекс и Вы перешли в Яндекс облако, Вы попадете на стартовую страницу, которая будет выглядеть так:

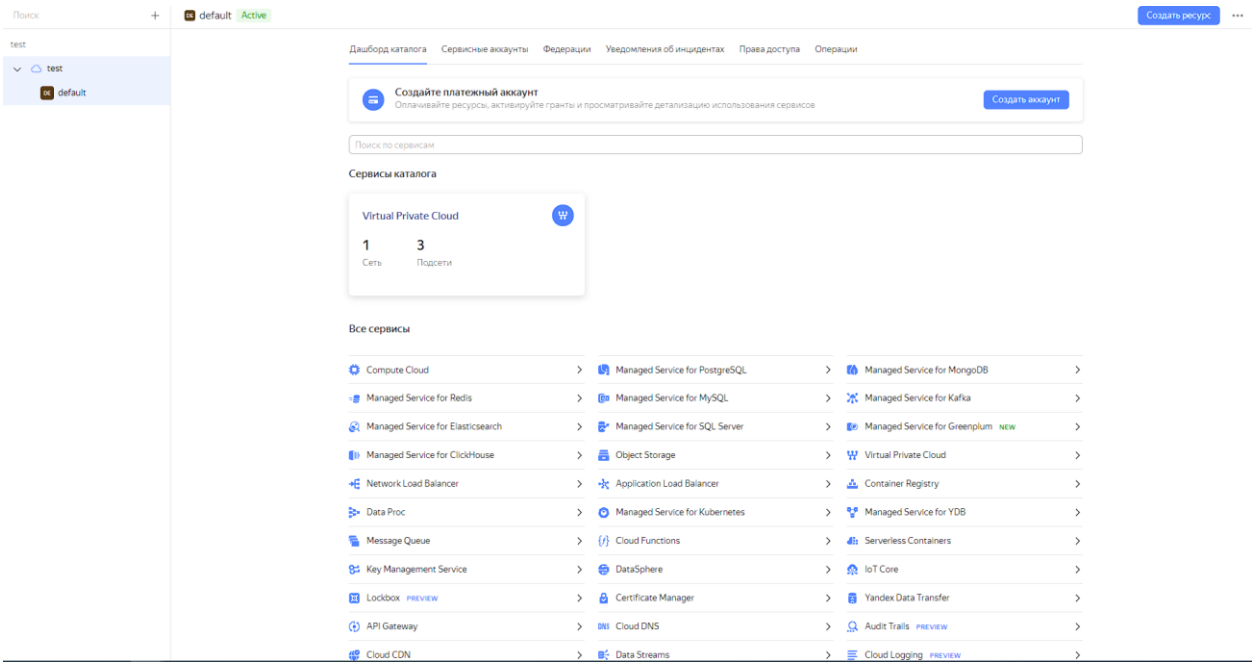


В правом верхнем углу находиться кнопка подключиться. Нажимаем ее, далее нужно будет принять все правила, затем сайт предложит нам создать наше первое облако.



В поле введите название, которое вам больше нравится.

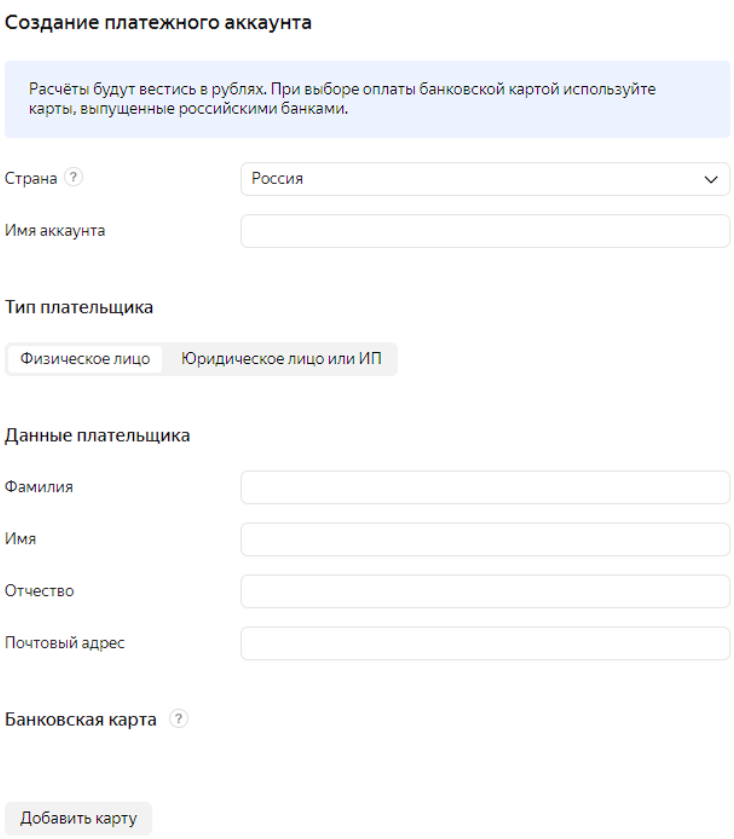
После небольшой загрузки мы попадаем в консоль управления.



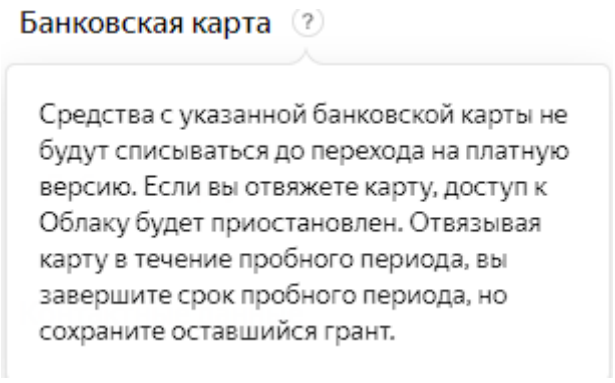
С первым пунктом закончили.

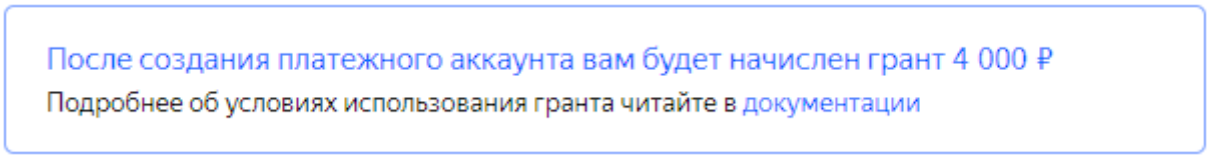
Теперь нам нужно создать платежный аккаунт. Для этого нужно нажать на кнопку СОЗДАТЬ АККАУНТ.

После этого вы попадете на страницу создания платежного аккаунта.

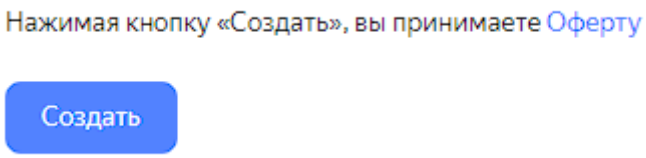


Заполняем все строки. Банковская карта добавляется обязательно.



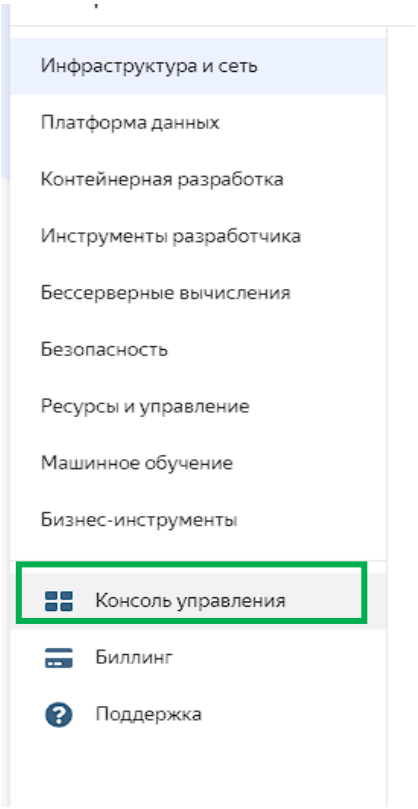


После заполнения всех пунктов и привязки карты внизу страницы будет кнопка создать. Нажимаем ее.



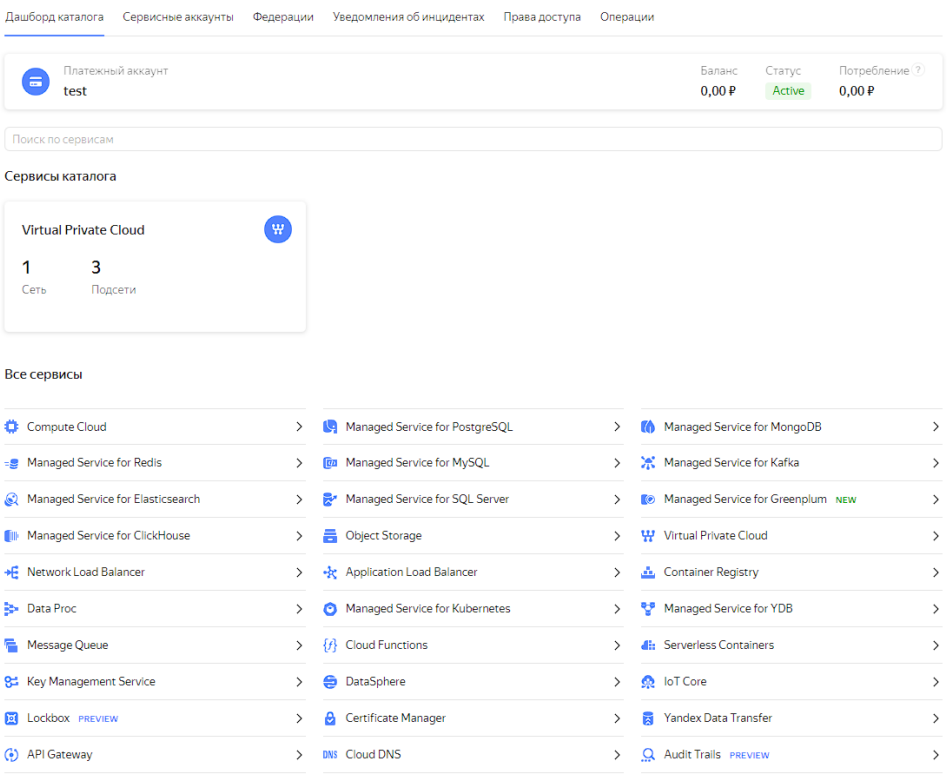
Далее Вы увидите данные своего аккаунта с доступным грантом на 4000р, 1000р из которых можно потратить на создание виртуальной машины (Compute Cloud).

После этого Вам нужно будет перейти обратно в консоль управления. Для этого в левом верхнем углу нужно нажать на значок меню . В открывшемся меню нажать на консоль управления



На странице консоли управления видно, что платежный аккаунт подключен и имеет статус Active. Можно заметить, что автоматически создался каталог с сетью

и подсетями.



На этом 2 пунк закончен.

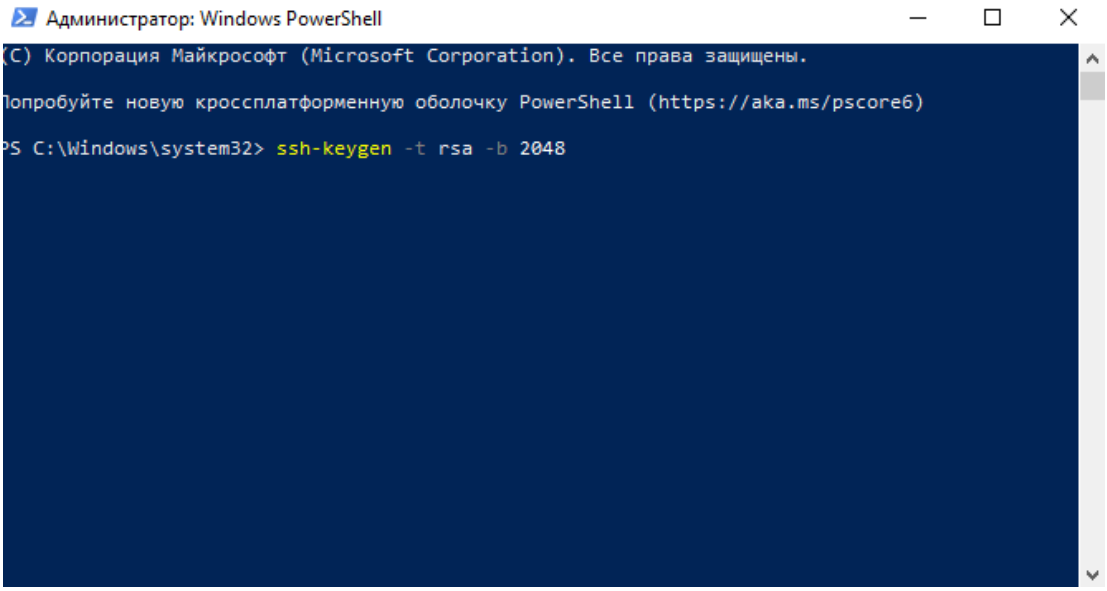
Теперь нам нужно создать пару SSH-ключей.

Для того что бы создать пару ключей на Windows 10 нужно:

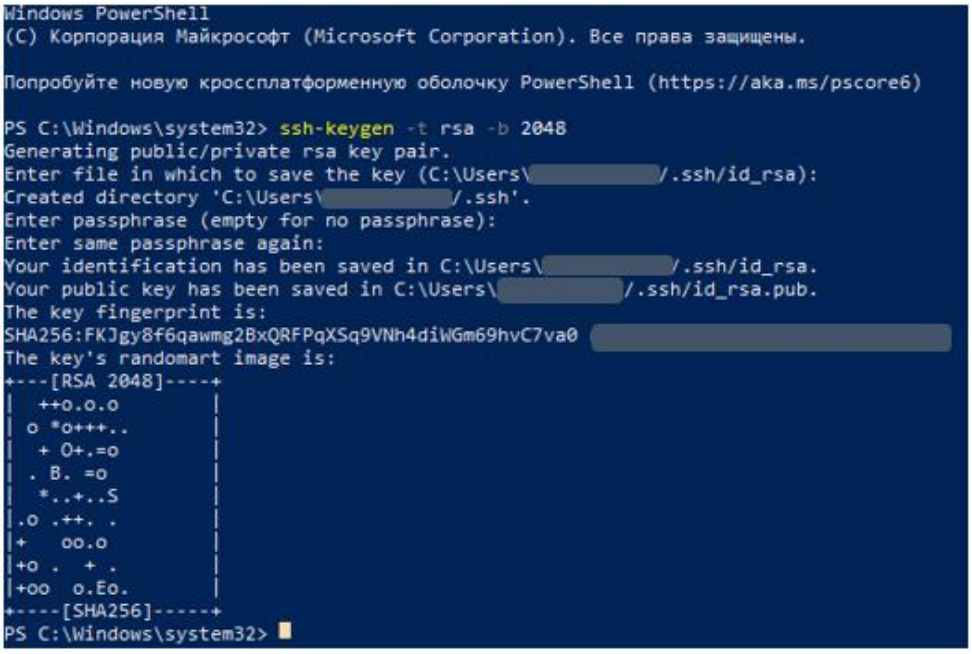
1. Запустите cmd.exe или powershell.exe. Для это нужно нажать правой кнопкой мыши на пуск и выбрать windows powershell(администратор) или в поиске ввести 

и нажать Enter.

2. Создайте новый ключ с помощью команды ssh-keygen. В открывшемся powershell Выполните команду: ssh-keygen -t rsa -b 2048



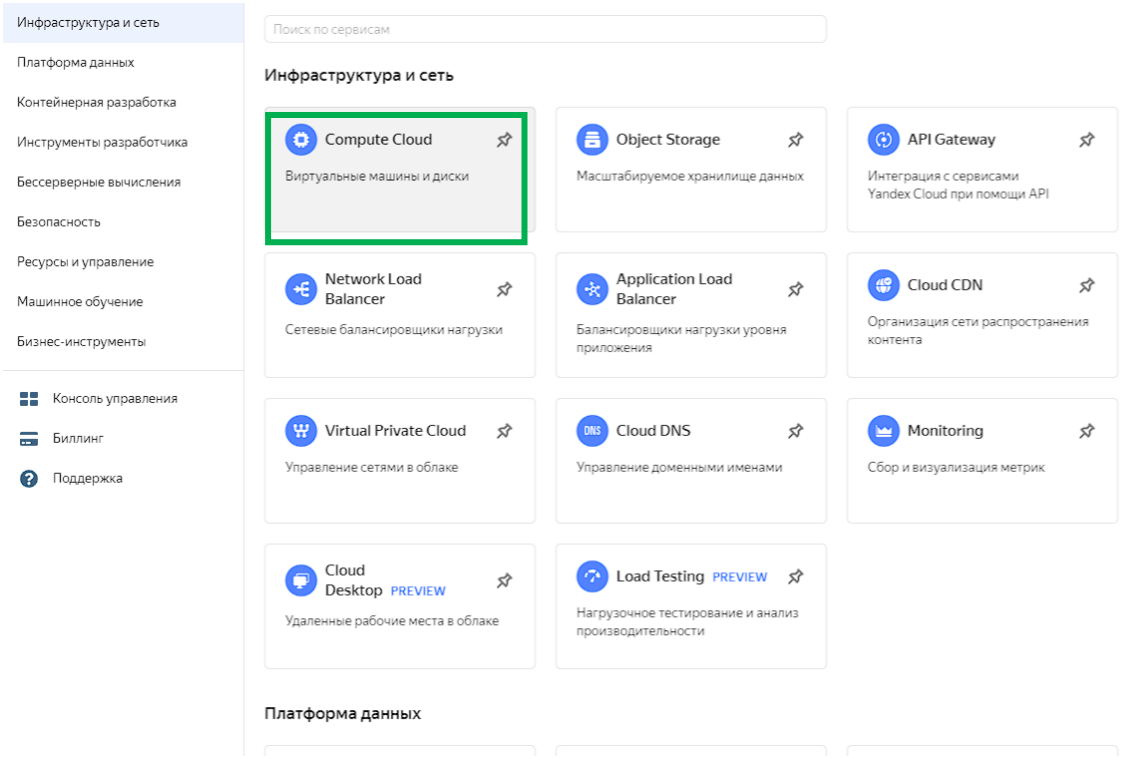
3. После выполнения команды вам будет предложено указать имена файлов, в которые будут сохранены ключи и ввести пароль для закрытого ключа. По умолчанию используется имя id_rsa. Ключи создаются в директории C:\Users\<имя_пользователя>\.ssh\ или C:\Users\<имя_пользователя>\ в зависимости от интерфейса командной строки. Публичная часть ключа будет сохранена в файле с названием <имя_ключа>.pub. Если оставлять все параметры по умолчанию, можно просто нажимать Enter, пока ключ не создастся. После этого Вы должны увидеть следующий вывод (персональные данные в инструкции скрыты, на их местах будут либо имя пользователя, либо имя компьютера и т.д.):



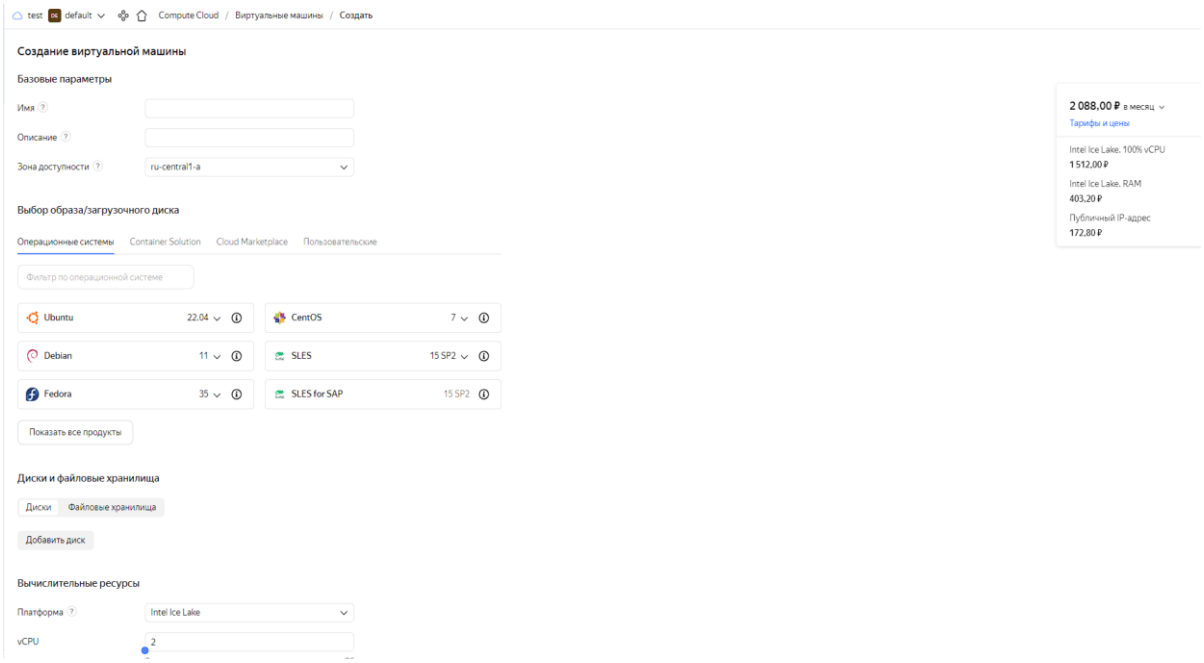
Важно

Надежно сохраните закрытый ключ: без него подключиться к ВМ будет невозможно.

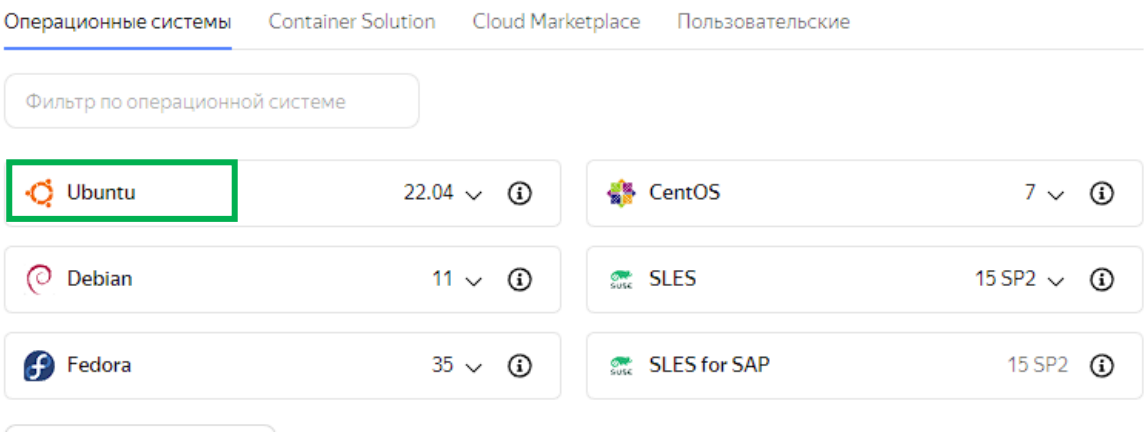
На этом создание пары ключей закончено. Теперь нам нужно создать виртуальную машину и подключиться к ней. Для того что бы создать ВМ нужно нажать на значок меню  в левом верхнем углу. В открывшемся меню нужно выбрать Compute Cloud.



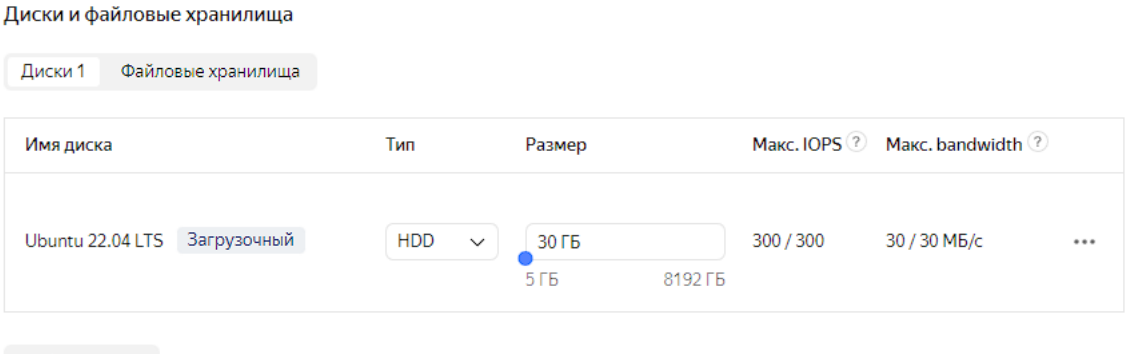
Далее нужно нажать Создать ВМ в правом верхнем углу вы должны увидеть данную страницу:



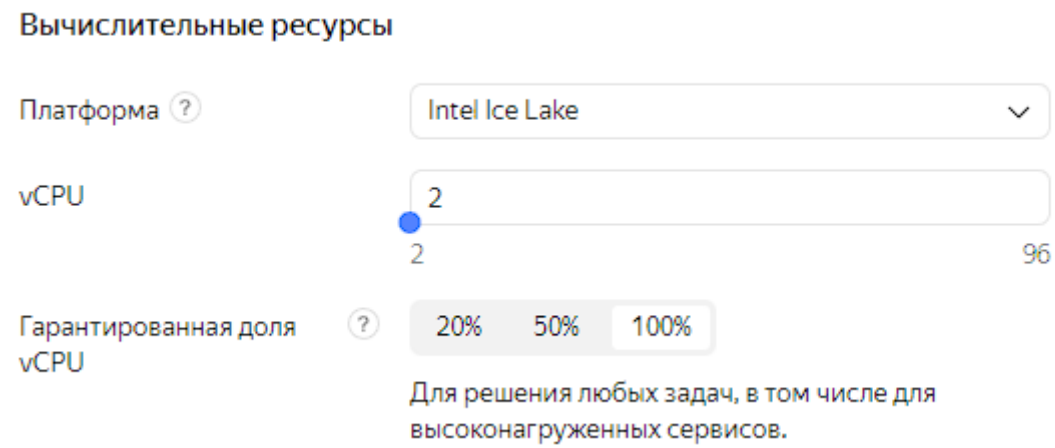
Нужно заполнить строки Имя, Описание. В операционной системе выбрать Ubuntu.



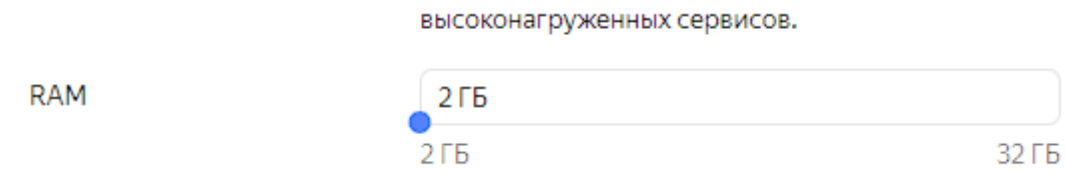
В диски и файловые хранилища поставим размер 30 ГБ.



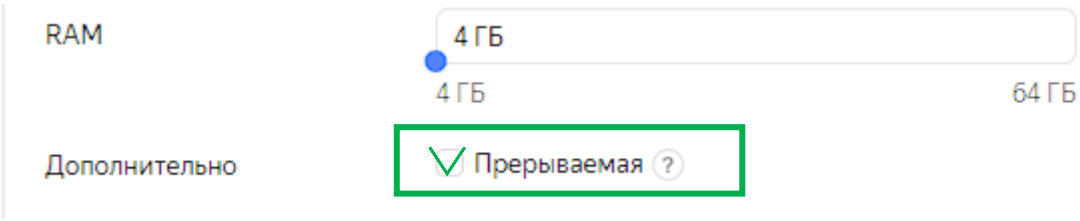
В вычислительных ресурсах оставим 2 vCPU.



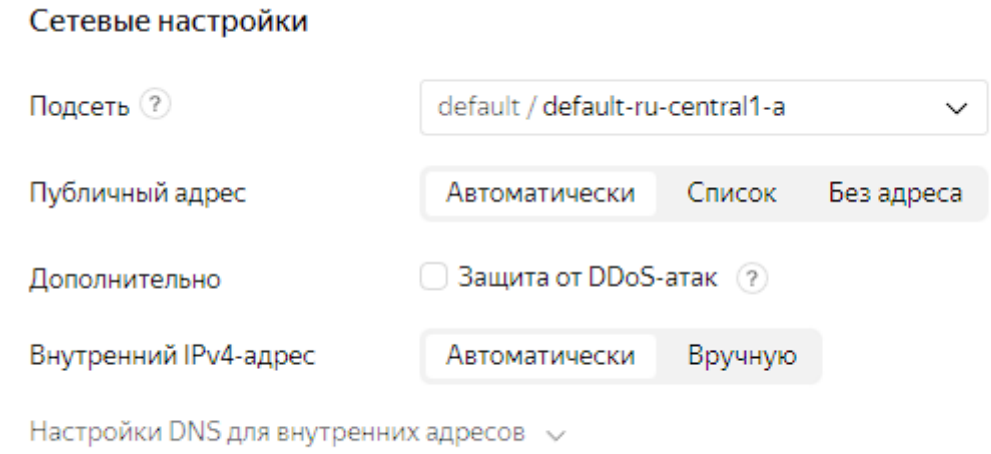
Оперативной памяти (RAM) можно поставить 4 ГБ (можно оставить и 2 ГБ).



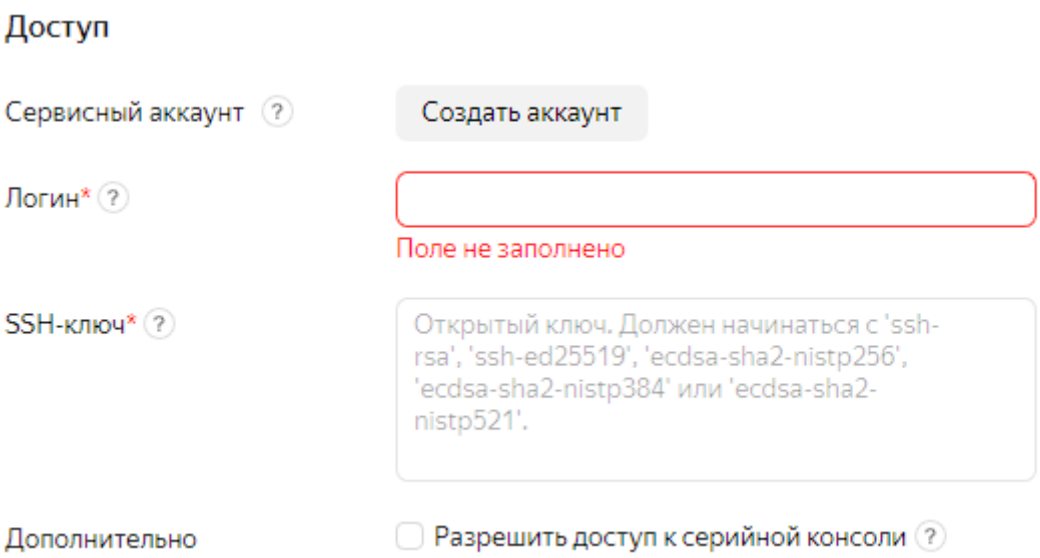
Необходимо установить галочку Прерываемая.



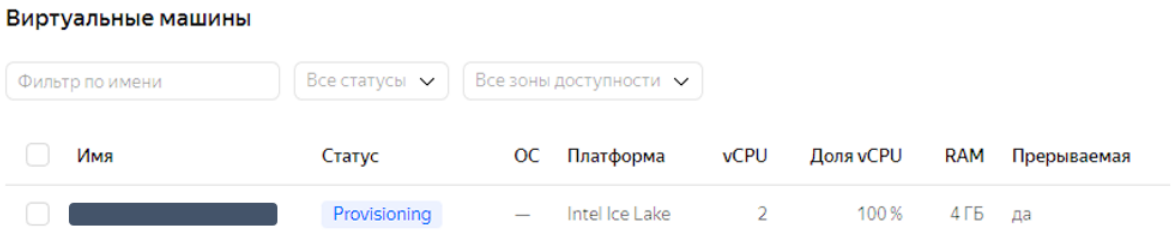
Сетевые настройки оставим неизменными.



Нужно заполнить поле логин и запомнить его или записать (он нам понадобится для подключения к ВМ), далее нужно вставить ключ, который мы заранее создали (если Вы не меняли имя, он должен называться id_rsa.pub). Данный ключ расположен по пути C:\Users\<Имя пользователя> \.ssh по умолчанию.

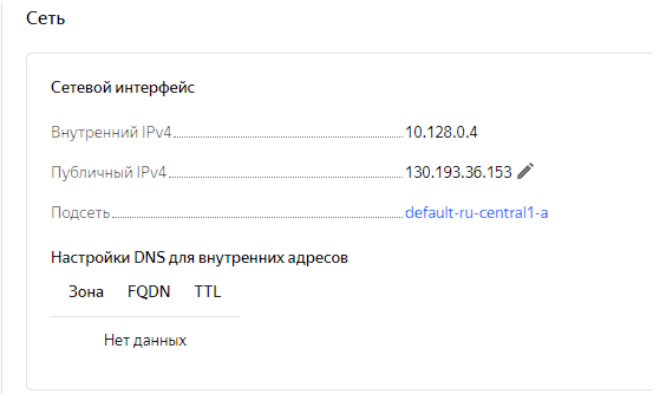


После заполнения всех строк и выбора всех параметров нужно нажать создать ВМ. После того, как Вы нажали на кнопку, Вы автоматически перейдете на страницу Виртуальные машины, где Вы увидите созданную Вами машину со всеми параметрами.



Виртуальная машина создана. Теперь нужно подключиться к ней. Для этого можно использовать утилиту ssh или программу PuTTY.

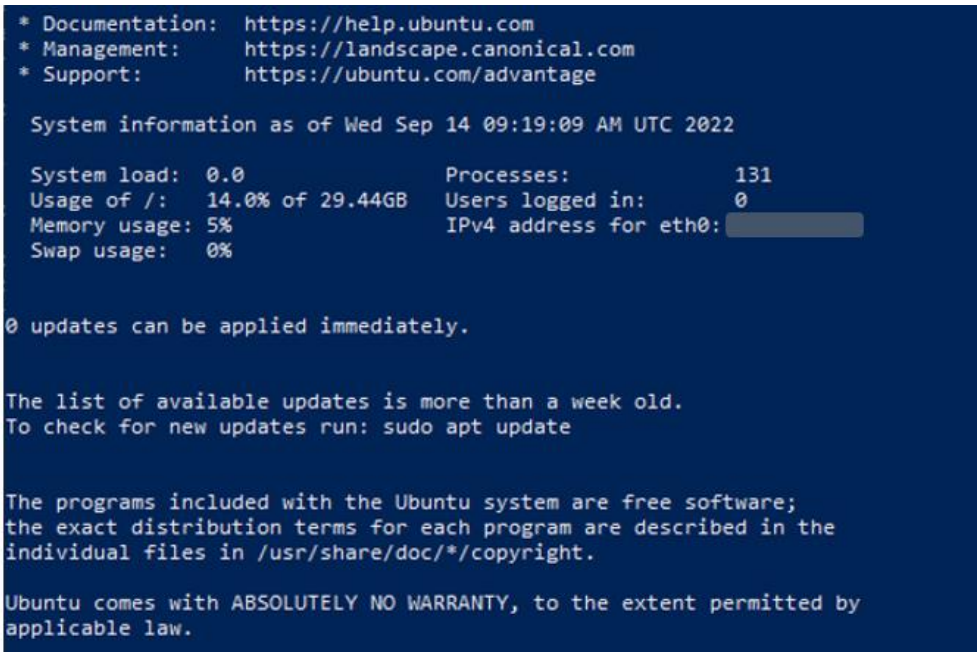
Для подключения необходимо указать публичный адрес виртуальной машины. Публичный IP-адрес можно узнать в консоли управления в поле **Публичный IPv4** блока **Сеть** на странице виртуальной машины. Для того, чтобы посмотреть **Публичный IPv4,** нужно выбрать созданную нами ВМ и перейти к ее параметрам. Там найти раздел **Сеть** и в этом блоке найти **Публичный IPv4**



После того, как посмотрели Публичный IPv4, нам нужно перейти в PowerShell (также, как мы делали это ранее) и ввести команду ssh <имя пользователя(это логин который вы должны были запомнить)>@<публичный_IP-адрес_виртуальной_машины>

При первом подключении к машине появится предупреждение о неизвестном хосте, нужно ввести yes.

Затем появится следующий вывод:



Поздравляю, мы создали ВМ и подключились к ней с помощью ssh утилиты!

Также можно подключиться с помощью программы PuTTY.

Скачать ее можно тут
https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

Модуль 2. Работы с командной строкой и скрипты в Linux

Содержание модуля:

В этом модуле рассматривается работа с терминалом ОС Linux: функциональные возможности и внешний вид окна терминала, структура команд в ОС Linux. Происходит работа с интерпретатором и языком программирования bash и sh. Рассматриваются переменные среды и окружения – создание, чтение, удаление, возможности использования. Объясняется понятие потока, перенаправление потоков, туннелирование команд. Слушатели узнают основные принципы написания сценариев командной строки, выполнения сценариев. Рассматриваются регулярные выражения. И практикуется написание скриптов для различных задач.

Модуль 2. Юнит 1: Текстовая лекция Основы работы с командной строкой.

ОСНОВЫ РАБОТЫ С КОМАНДНОЙ СТРОКОЙ

Простейшая команда в Linux состоит из одного слова — названия программы, которую необходимо выполнить.

Примером простейшей команды является запуск программы **who**. Она выводит по одной строке на каждого зарегистрированного в системе пользователя в системе на настоящий момент: в первой колонке указывается имя пользователя, во второй — «точка входа» в систему, далее следует дата и время регистрации и имя хоста.

```
student@astra:~$ who

student tty1 Dec 23 16:31 (localhost)

student tty2 Dec 23 17:12 (localhost)

student@astra:~$

student@astra:~$ who am i

student tty2 Dec 23 17:12 (localhost)

student@astra:~$
```

Еще одна программа, выдающая информацию о пользователях, работавших в системе в последнее время, называется **last**. Выводимые этой программой строки напоминают вывод программы **who**, только здесь перечислены еще и завершившие работу пользователи.

```
student@astra:~$ last

student tty2 localhost Thu Dec 23 17:12 still logged in

student tty1 localhost Thu Dec 23 16:31 still logged in

cacheman ??? localhost Thu Dec 23 16:15 - 16:17 (00:01)

cacheman ??? localhost Thu Dec 23 16:08 - 16:08 (00:00)

cyrus ??? localhost Thu Dec 23 16:08 - 16:08 (00:00)

reboot system boot 4.15.3-generic Thu Dec 23 16:03 (04:13)
```

В этом примере можно обнаружить неизвестных нам пользователей *cacheman* и *cyrus* — и мы точно знаем, что не создавали учетных записей с такими именами. Это **системные пользователи** — специальные учетные записи, которые используются для работы некоторыми программами. Поскольку эти «пользователи» регистрируются в системе без помощи монитора и клавиатуры, их «точка входа» в систему не определена (во второй колонке записано «???»). В выводе программы **last** появляется даже пользователь reboot (перезагрузка). В действительности такой учетной записи нет, программа **last** таким способом выводит информацию о том, когда была загружена система.

Команды, подаваемые с клавиатуры с помощью Ctrl, как и символы, передаваемые при этом системе, принято обозначать знаком «^», после которого следует имя клавиши, нажимаемой вместе с Ctrl: например, одновременное нажатие Ctrl и «a» обозначается «^A». Так, для завершения работы программы **cat**, которая считывает построчно данные с клавиатуры и выводит их на терминал, можно воспользоваться командой «^C» или «^D»:

В большинстве случаев при разборе командной строки первое слово считается именем команды, а остальные — ее **параметрами**.

ДОКУМЕНТАЦИЯ LINUX

Работать с Linux, не заглядывая в документацию, практически невозможно. Все утилиты, все демоны Linux, все функции ядра и библиотек, структура большинства конфигурационных файлов описаны либо в руководствах, либо в info-страницах, либо в сопроводительной документации.

Больше всего различной полезной информации содержится в страницах руководства (manpages, info).

Для того, чтобы посмотреть страницу руководства, нужно дать команду системе **man** объект:

```
[student@astra:~$ man cal

CAL(1) General Commands Manual CAL(1)

NAME

cal - displays a calendar

SYNOPSIS

cal [-smjy13] [[month] year]

DESCRIPTION
```

```
Cal displays a simple calendar. If arguments are not specified,
the current month is displayed. The options are as follows:
```

«Страница руководства» занимает, как правило, больше одной страницы экрана. Для того, чтобы читать было удобнее, **man** запускает программу постраничного просмотра текстов — **less**. Управлять программой **less** просто: страницы перелистываются пробелом, а когда читать надоест, надо нажать «q» (Quit). Перелистывать страницы можно и клавишами Page Up/Page Down, для сдвига на одну строку вперед можно применять Enter или стрелку вниз, а на одну строку назад — стрелку вверх. Переход на начало и конец текста выполняется по командам «g» и «G» соответственно (Go). Полный список того, что можно делать с текстом в **less**, выводится по команде «H» (Help).

Модуль 2. Юнит 2: Тест для самоконтроля.

1. Какая команда выводит список пользователей, которые в настоящий момент зарегистрированы в системе (вошли в систему)?
   - **who**
   - users
   - userslist
   - whoami

2. Какая команда позволяет просмотреть справочную информацию или руководства?
   - which
   - **man**
   - mal
   - what

Модуль 2. Юнит 3: Контрольное задание.

Задание на «перетаскивание» объектов в нужном порядке (только для первой колонки).

Затем – на соотношение позиций (к составленным в верном порядке командам из первой колонки надо соотнести описание из второй колонки).

Задание: ***1. Для каждой команды расположите элементы в верном порядке*.** ***2. Сопоставьте команду с описанием ее результата*.**	
`systemctl mask NetworkManager`	Маскировать юнит NetworkManager
`rm -r /tmp/folder`	Удалить папку folder со всем содержимым
`cat -n file`	Вывести содержимое файла file с пронумерованными строками
`cat -h`	Вывести справку по команде cat

Модуль 2. Юнит 4: Текстовая лекция Создание файла скрипта на bash.

СОЗДАНИЕ ФАЙЛА СКРИПТА НА BASH

Скрипт на языке bash иногда называют сценарием командной строки. И это не случайно, ведь bash-скрипт представляет собой последовательность bash-команд, которые можно вводить с клавиатуры, собранные в единый файл и объединённые некоторой общей целью. Конечно, в командной строке мы можем выполнять несколько команд через разделитель – точку с запятой. Но объединение команд в файл, во-первых, структурирует сам вид записи нужной последовательности, во-вторых, позволяет более ясно обозначить составные части и модули создаваемой программы, и главное – файл скрипта может быть сохранен и выполнен в любое удобное время, либо даже поставлен в автоматическое периодическое выполнение.

Так, создав сценарий командной строки, мы, по сути, создаем программу. И, как и у любой другой программы, у bash-скрипта есть свои особенности. Любой программе нужен интерпретатор. Для bash-скрипта есть несколько вариантов указать интерпретатор.

Первый: в начале самого файла скрипта. Он указывается с помощью специальной конструкции и выглядит так: #!/bin/bash . Первая часть – специальные символы #! , а вторая часть – полный путь к самому интерпретатору. Интерпретатором может выступать не только bash – можно ведь создавать и выполнять скрипты на любом языке, а значит, с любым интерпретатором (главное, чтобы он был установлен в вашей системе). Так, можно создать скрипт на python (естественно, последующие после указания интерпретатора команды тоже должны быть на языке python соответственно). Тогда указание интерпретатора будет выглядеть следующим образом: #!/usr/bin/python3 . Может быть следующий вариант строки: #!/usr/bin/env php . Данный вариант подходит, если мы не знаем точный полный путь к интерпретатору, указываем путь к утилите env и она находит нужный интерпретатор по имени, которое мы указываем как аргумент для нее (в данном случае это php, соответственно, в системе будет найден и вызван интерпретатор для языка php).

Второй способ указания интерпретатора – непосредственно в команде для выполнения скрипта. Т.е. указываем сначала полный путь до интерпретатора, а затем в качестве аргумента передаем путь к нашему файлу с записанными командами. Интерпретатор также исполнит все, что находится в файле, если оно написано на соответствующем языке. Правда, указывать его таким способом придется каждый раз при запуске скрипта. Выглядеть это может так: например, мы хотим запустить bash-скрипт с названием myscript.sh: $ /bin/bash /home/user/myscript.sh .

Но есть очень важная особенность: если вы просто создадите файл, запишете в него верные команды на нужном языке, укажете нужный интерпретатор и

захотите запустить – ваш скрипт не запустится. Вы увидите ошибку Permission Denied. Все потому, что сам файл еще не стал исполняемым. Так, чтобы система могла выполнить файл как последовательность команд, у файла должны быть специальные права доступа, а именно - право на исполнение. Их можно назначить отдельно для владельца файла, для группы владельца, либо для всех остальных пользователей. Подробнее о назначении прав доступа вы узнаете в следующих разделах курса. Для корректировки прав доступа к скрипту на данном этапе достаточно будет знание команды chmod ugo+x <название скрипта/полный путь к файлу скрипта> (либо chmod a+x <название скрипта/полный путь к файлу скрипта>). Эта команда добавляет разрешение на исполнение файла для всех пользователей. После чего можно запускать написанный скрипт. Запуск происходит путем указания названия созданного скрипта (включая ссылку на текущий каталог – т.е. указывая впереди точку и слеш). Например, $ ./myscript.sh . Данный способ сработает, если мы находимся в том же каталоге, в котором находится и скрипт. Если в процессе работы мы перешли в другой каталог, то необходимо указывать полный путь к файлу скрипта. Например, $ /home/user/myscript.sh .

Итак, приведем несколько примеров содержимого файлов скриптов.

Скрипт, выводящий в консоль Hello world! :

```
#!/bin/bash
echo “Hello world!”
```

Скрипт, выводящий в консоль Hello world из содержимого двух переменных:

```
#!/bin/bash
a=”Hello “
b=world
echo $a$b
```

Как вы поняли, скрипт - программа, представляющая собой последовательность команд bash, но, как и в любой программе, может возникнуть необходимость не только последовательно выполнять команды, но и хранить данные, получать их от пользователя либо из другой программы. Для этого нам понадобятся переменные – именованные области памяти, в которых можно хранить, записывать и считывать различные данные. В bash, чтобы создать переменную, нужно объявить ее имя, затем через знак равенства присвоить желаемое значение. Тип переменной не указывается – в bash все считается строкой. Пример задания переменной var1: var1=My_first_var .Чтобы получить значение переменной, нужно указать знак доллара, а дальше сразу имя переменной. Так, например, чтобы вывести в консоль значение переменной var1 нужно написать команду echo $var1 . Также можно получить значения не только созданных переменных, но и выполнения команд. Например, чтобы получить значение команды date нужно ввести ту же конструкцию, что и для получения значения переменной, но имя команды взять в скобки: $(date). Чтобы выполнить арифметическую операцию

существует команда let. Она выполнит операцию, которая будет указано после имени let в кавычках. Например, среднего арифметического 2 и 5 :

```
#!/bin/bash
i=2
w=5
let  s="($i+$w)/2"
echo $s
```

Иногда возникает потребность выполнять действия не с одними и теми же данными, а с меняющимися значениями, например, введенными пользователем с консоли. Такие введенные значения – это параметры скрипта, т.е. значения, которые скрипт «принимает на входе», затем выполняет с ними какие-либо действия и, в зависимости от них, выдает результат. Чтобы внутри скрипта объявить параметр, нужно указать конструкцию в виде знака доллара и следующего за ним номера параметра (начиная отсчет с единицы). Так, выражение $1 будет обозначать первый переданный скрипту параметр, $2 – второй, $3 – третий и т.д. Пример скриптов с использованием параметров:

Скрипт	Команда вызова скрипта	Результат
#!/bin/bash echo $1	./script.sh test my script	test
#!/bin/bash echo $2	./script.sh test my script	my
#!/bin/bash echo $3	./script.sh test my script	script

Здесь ./script.sh test my script – команда вызова самого скрипта, параметр1, параметр2, параметр3.

Но и этого может оказаться недостаточно. Иногда в скриптах необходимо использование специальных конструкций, которые позволяют неоднократно повторять некоторые действия, либо выполнять действия, отталкиваясь от неких условий. Это так называемые циклы и ветвления. Рассмотрим их подробнее.

Ветвления в программе осуществляются с помощью условных операторов. В bash данные оператор, как и во многих других языках, называется if. Полностью данный оператор включает в себя следующие ветки: **если** какое-либо условие выполнилось, **то** выполни такие действия, **в противном случае** – выполни другие действия, **закончи**. Синтаксис на bash выглядит так:

if команда_условие

then

команда

else

команда

fi

Команда условие в данном случае должна иметь код возврата либо 0 – ложь, либо 1- истина, как и в других языках программирования. Если условие истинно, действия после первого then выполняются, если ложно – выполняются действия после else. Но не всегда нам удобно ориентироваться на код возврата именно команды, чаще возникает потребность сравнить числа или строки. Для такого типа сравнения была придумана конструкция [[параметр1 оператор параметр2]]. Оператор сравнения здесь стандартный: <, >, =, != и т.д. (меньше, больше, равно, не равно и т.д.). Параметрами могут выступать строки либо числа. Пример использования такой конструкции для сравнения двух чисел, введенных с консоли:

Скрипт	Команда вызова скрипта	Результат
#!/bin/bash *If [[$1 > $2]]* *then* *echo $1 is bigger* *else* *echo $2 is bigger* *fi*	*./script.sh 8 5*	*8 is bigger*
#!/bin/bash *If [[$1 > $2]]* *then* *echo $1 is bigger* *else* *echo $2 is bigger* *fi*	*./script.sh 3 12*	*12 is bigger*
#!/bin/bash *If [[$1 > $2]]* *then* *echo $1 is bigger* *else* *echo $2 is bigger* *fi*	*./script.sh 4 4*	*4 is bigger*

Циклы в скриптах подразумевают многократное повторение действий. Типы и синтаксис циклов (как и ветвлений) похожи на другие языки программирования. Есть цикл типа for – идет по некоторому индексу и while – идет в зависимости от некоторого условия.

Так, смысл цикла for: **для** какого-то индекса **в** каком-то списке **выполняй** команду, **закончи**. Синтаксис на bash выглядит так:

for переменная **in** список

do

команда

done

Пример сложения первых десяти чисел с помощью данной конструкции:

```
#!/bin/bash
sum=0
for index in 1 2 3 4 5 6 7 8 9 10
do
let “sum=$sum+$index”
done
echo $sum
```

Суть цикла while заключается в следующем: **пока** выполняется некое условие **делай** такие команды, **закончи**. Синтаксис на bash выглядит так:

while команда условие

do

команда

done

Команда условие в этом случае то же самое, что и в ветвлении if. Цикл while может ни разу и не выполнится, если условие сразу же окажется ложным. Пример подсчета, насколько введенное число меньше 5 с помощью цикла while:

```
#!/bin/bash
dif=0
i=$1
while [[ $i < 5 ]]
do
let “dif=$dif+1”
let “i=$i+1”
done
if [[ $dif > 0 ]]
then
echo 5 over $1 on $dif
else
echo 5 is not over $1
fi
```

Для данного скрипта:

Команда вызова скрипта	*Результат*
./script.sh 3	*5 over 3 on 2*
./script.sh 9	*5 is not over 9*
./script.sh 5	*5 is not over 5*

Модуль 2. Юнит 5: Тест для самоконтроля 2

1. Какой командой следует запустить скрипт script1.sh, находящийся в каталоге /etc/dir1/, если работа пользователя в данный момент происходит в /home/Liza/dir2?
    - ./scrip1.sh
    - **/etc/dir1/script1.sh**
    - ../../../../script1.sh
    - run script1.sh

2. Если в скрипте содержится строка «echo $1 $3 $n», то она означает:
    - **Вывод в консоль первого и третьего параметров скрипта, затем значения переменной n**
    - Вывод в консоль первого, третьего и n-го параметров скрипта
    - Вывод в консоль параметров скрипта с первого по третий, затем значения переменной n
    - Вывод в консоль первого и третьего параметров скрипта, затем начало новой строки

3. В скриптах бывают следующие типы циклов:
    - If, while, for
    - Do, done, break
    - **For, while**
    - If, for

Модуль 2. Юнит 6: Контрольное задание 2

Задание: Расставьте перечисленные действия в верном порядке для успешного написания скрипта.

1. Открыть терминал
2. В нужном каталоге создать файл.
3. Открыть созданный файл в консольном текстовом редакторе.
4. Ввести построчно необходимые *shell* команды.
5. Сохранить и закрыть отредактированный файл.
6. Сделать файл исполняемым, добавив нужные права доступа. *(может стоять на 3м месте и остальное сдвигается вниз)*
7. Проверить скрипт, запустив его в терминале.

Модуль 2. Юнит 7: Памятка по регулярным выражениям

Регулярные выражения – это специальным образом записанные строки, символы, для обработки и изменения строк, для решения множества задач.

\	Начинаются буквенные спецсимволы, а также если нужно использовать спецсимвол в виде знака препинания.
^	Начало строки.
$	Конец строки.
*	Предыдущий символ может повторять 0 или больше раз.
+	Предыдущий символ должен повторять 1 или больше раз.
?	Предыдущий символ может встречаться 0 или 1 раз.
{n}	(n) раз нужно повторить предыдущий символ.
{N,n}	Предыдущий символ повторяется от N до n раз.
.	Любой символ кроме перевода строки.
[az]	Любой символ, указанный в скобках.
x|y	x или y.
[^az]	Любой символ, кроме тех, что в скобках.
[a-z]	Любой символ от a до z.
[^a-z]	Любой символ которого нет в диапазоне от a до z.
\b	Граница слова с пробелом.
\B	Обозначает что символ должен быть внутри слова.
\d	Означает, что символ – цифра.
\D	Нецифровой символ.
\n	Перевод строки.
\s	Пробел, табуляция и так далее.
\S	Любой символ кроме пробела.
\t	Табуляция.
\v	Вертикальная табуляция.
\w	Любой буквенный, включая подчеркивание.
\W	Любой буквенный, кроме подчеркивания.
\uXXX	Символ Unicdoe.

Примеры:

1) \d\d/\d\d/\d{4} – Дата в формате ДД/ММ/ГГГГ.
2) \d{4} – Последовательно из 5 цифр.

3) \b\w{3}\b – Слово в точности из 3 букв.

Модуль 3. УПРАВЛЕНИЕ ПРОЦЕССАМИ

Содержание модуля:

В данном модуле происходит работа с планированием задач в Linux. Объясняется понятие службы. Затрагиваются темы настройки автозапуска приложений, планировщики задач at и cron – создание, редактирование, удаление задач. Рассматриваются способы редактирования конфигурационных файлов. В модуле объясняется понятие системы инициализации systemd, отрабатывается создание задач с помощью systemd.

Модуль 3. Юнит 1: Текстовая лекция Управление службами

УПРАВЛЕНИЕ СЛУЖБАМИ

В среде ОС для управления службами, точками монтирования и т.п. применяется системный менеджер **systemd**. Менеджер **systemd**:

- обеспечивает параллельный запуск служб в процессе загрузки ОС,
- использует сокеты и активацию D-Bus для запускаемых служб,
- предлагает запуск демонов по необходимости,
- отслеживает запуск служб,
- поддерживает мгновенные снимки и восстановление состояния системы, монтирование и точки монтирования,
- а также внедряет основанную на зависимостях логику контроля процессов сложных транзакций.

Отличительной особенностью **systemd** является использование контрольных групп Linux, обеспечивающих иерархическую структуризацию служб: любая запущенная служба помещается в отдельную контрольную группу с уникальным идентификатором. Когда служба запускает другую зависимую службу, она автоматически включается в группу с тем же идентификатором. При этом непривилегированные службы не могут изменить свое положение в иерархии. При штатном завершении работы службы будут завершены и все зависимые от нее службы.

МЕНЕДЖЕР SYSTEMD

Менеджер **systemd** оперирует специально оформленными файлами конфигурации — юнитами (unit). Каждый юнит отвечает за конкретную службу (*.service), точку монтирования (*.mount), устройство (*.device), файл подкачки (*.swap), сокет (*.socket) и т. д.

Юниты в одном из каталогов:

- /usr/lib/systemd/system/ — юниты из установленных пакетов;
- /run/systemd/system/ — юниты, созданные в режиме рантайм. Эти юниты имеют более высокий приоритет, чем юниты из установленных пакетов;
- /etc/systemd/system/ — юниты, созданные и управляемые администратором. А эти юниты приоритетнее, чем юниты, созданные в режиме рантайм.

- Для отслеживания и контроля состояния менеджера **systemd** и юнитов используется утилита командной строки **systemctl**.

Для просмотра списка установленных юнитов нужно выполнить команду:

```
systemctl list-unit-files
```

Запущенные юниты можно посмотреть по команде:

```
systemctl list-units
```

Та же команда, но с параметром -t <тип_юнита> покажет запущенные юниты определенного типа:

```
systemctl list-units -t service
```

Основные параметры для использования с утилитой командной строки **systemctl** приведены в таблице ниже:

Параметр	Описание
systemctl start <юнит>	Незамедлительно запустить юнит
systemctl stop <юнит>	Незамедлительно остановить юнит
systemctl restart <юнит>	Перезапустить юнит
systemctl reload <юнит>	Перезагрузить настройки юнита
systemctl status	Вывести общую информацию о состоянии системы и список юнитов, которым соответствуют запущенные процессы. При запуске команды с именем юнита будет выведена информация о статусе данного юнита
systemctl cat <юнит>	Показать содержимое юнита
systemctl is-enabled <юнит>	Проверить включение юнита в автозапуск при загрузке системы
systemctl enable <юнит>	Добавить юнит в автозапуск при загрузке системы

systemctl disable <юнит>	Удалить юнит из автозапуска при загрузке системы
systemctl mask <юнит>	Маскировать юнит для исключения возможности его запуска
systemctl unmask <юнит>	Снять маску юнита
systemctl help <юнит>	Показать страницу руководства man юнита (при наличии поддержки данной функции для указанного юнита)
systemctl daemon-reload	Перезагрузить systemd для поиска новых или измененных юнитов
systemctl —failed	Показать список юнитов, которые не были запущены из-за ошибки

РАБОТА С ЮНИТАМИ

Конфигурационными файлами **systemd** при использовании менеджера **systemd** возможно как корректировать существующие юниты, так и создавать новые.

Юнит представляет собой ini-подобный файл, имя которого состоит из имени юнита и суффикса, определяющего тип юнита. В общем случае юнит-файл включает секции [Unit] и [Install], а также дополнительные секции, соответствующие конкретному типу юнита.

Секция [Unit] содержит описание юнита, а также информацию о зависимостях при запуске юнита:

- Description= — описание юнита;
- Wants= – зависимость требования запуска. Требование исходного юнита запустить юнит, указанный в параметре. При этом результат запуска юнита, указанного в параметре, не влияет на запуск исходного юнита. При отсутствии параметров After= и Before= юниты будут запущены одновременно;
- Requires= — зависимость требования запуска. Требование исходного юнита запустить юнит, указанный в параметре. При этом ошибка запуска юнита,

приведенного в параметре, приведет к ошибке запуска исходного юнита. При отсутствии параметров After= и Before= юниты будут запущены одновременно;

- After= — зависимость порядка запуска. Дополнительный, но не обязательный параметр к параметрам Wants= и Requires=, указывающий на необходимость запуска исходного юнита только после запуска юнита, указанного в параметре. При этом если данный параметр используется с параметром Wants=, то исходный юнит будет запущен вне зависимости от результата запуска юнита, указанного в параметре;
- Before= — аналогичен параметру After=, только определяет запуск исходного юнита до запуска юнита, указанного в параметре. Секция [Install] содержит информацию об установке юнита. Используется командами **systemctl enable** <юнит> и **systemctl disable** <юнит>.
- Alias= — список альтернативных имен юнита, разделенных пробелом. Имена должны иметь тот же суффикс, что и имя файла юнита. При использовании команды **systemctl enable** будут созданы символические ссылки из перечисленных имен на данный юнит.

Секция [Service] в юните службы содержит следующие параметры:

- Type= — определяет тип запуска службы:

a) simple — используется по умолчанию. Служба будет запущена незамедлительно. Процесс при этом не должен разветвляться. Не рекомендуется использовать данный тип, если другие службы зависят от очередности при запуске данной службы. Исключение – активация сокета;

b) forking — служба запускается однократно и процесс разветвляется с завершением родительского процесса. Рекомендуется использовать данный тип для запуска классических демонов. Потребуется также определить PIDFile, чтобы менеджер systemd мог отслеживать основной процесс;

c) oneshot — используется для скриптов, которые завершаются после выполнения одного задания;

d) notify — аналогичен типу simple, но дополнительно демон отправит менеджеру systemd сигнал о своей готовности;

e) dbus — сервис находится в состоянии готовности, когда определенное BusName появляется в системной шине DBus;

f) idle — менеджер systemd отложит выполнение службы до момента отправки всех заданий;

- PIDFile= —расположение pid-файла;

- WorkingDirectory= — рабочий каталог приложения;
- User= — пользователь, от имени которого будет запущена служба;
- Group= — группа, от имени которой будет запущена служба;
- OOMScoreAdjust= — приоритет завершения процесса при нехватке памяти, где 1000 – максимальное значение, означающее полный запрет на завершение процесса;
- ExecStop= — указывает на скрипт, который должен быть выполнен перед остановкой службы;
- ExecStart — указывает на команду, которая должна быть выполнена после запуска службы;
- RemainAfterExit — предписывает systemd считать процесс активным после его завершения.

Секция [Socket] в юните сокета определяет следующие параметры для управления сокетом:

- ExecStart= — правило запуска;
- ExecReload= — правило перезапуска;
- KillMode= — правило завершения;
- Restart= — правило перезапуска при возникновении ошибки.

ЦЕЛЕВОЕ СОСТОЯНИЕ

В **systemd** уровни запуска файлов реализованы в виде сгруппированных юнитов, представляющих целевое состояние (цель). Каждая цель имеет собственное имя вида <имя_состояния>.target и предназначена для конкретных задач. Одновременно могут быть активны несколько целей.

Цели могут наследовать все службы других целей, добавляя к ним свои.
В **systemd** также имеются цели, имитирующие общие уровни запуска SystemVinit, поэтому для переключения между целевыми юнитами можно использовать команду:

```
telinit RUNLEVEL
```

Для определения доступных целевых состояний используется команда:

```
systemctl list-unit-files --type=target
```

Для определения активных целевых состояний используется команда:

```
systemctl list-units --type=target
```

Чтобы перейти в целевое состояние, используется команда:

```
systemctl isolate <имя_состояния>.target
```

Она изменяет только текущий уровень выполнения, и ее действие не повлияет на последующие загрузки системы.

Целевого состояние по умолчанию, которое **systemd** использует сразу после загрузки системы, можно посмотреть по команде:

```
systemctl get-default
```

Целевое состояние по умолчанию задается символьной ссылкой:

```
/etc/systemd/system/default.target.
```

Для смены целевого состояния по умолчанию требуется перезаписать эту ссылку.

Примеры:

```
ln -sf /usr/lib/systemd/system/multi-user.target

/etc/systemd/system/default.target
ln -sf /usr/lib/systemd/system/graphical.target

/etc/systemd/system/default.target
```

Для просмотра дерева зависимостей юнитов от цели нужно выполнить команду:

```
systemctl list-dependencies <имя_состояния>.target
```

Модуль 3. Юнит 2: Тест для самоконтроля

1. Какая команда позволяет перезапустить службу?
    - System restart
    - Service restart
    - Servicectl restart
    - **Systemctl restart**

ПЛАНИРОВЩИК ЗАДАЧ

Cron — это программа-демон, которая выполняет задания по расписанию и позволяет делать это неоднократно. Это значит, что задание можно запустить в определенное время или через определенный промежуток времени.
Демон **at** запускает команду каждый конкретный промежуток времени. А **atd** работает с командами, которые должны запуститься однократно, но в конкретный момент.

В системах Linux многие задачи планируются для регулярного запуска:

- ротация журналов;
- обновление базы данных для программы locate;
- резервное копирование;
- сценарии обслуживания (например, удаление временных файлов).

При загрузке системы запускается демон `cron` и проверяет очередь заданий `at` и заданий пользователей в файлах `crontab`. При запуске демон cron сначала проверяет каталог `/var/spool/cron` на наличие файлов `crontab`. Файлы `crontab` имеют имена пользователей, соответствующие именам пользователей из `/etc/passwd`. Каждый пользователь может иметь только один файл `crontab`, но записей в файле может быть несколько.

Еще раз! Файлы `crontab` содержат инструкции для демона `cron`, который запустит задание(-я), описанное в файле `crontab`. Все файлы `crontab` из каталога `/var/spool/cron` загружаются в память, одновременно с ними загружаются файлы из `/etc/cron.d`. После этого демон `cron` загружает содержимое файла `/etc/crontab`. При стандартных настройках, содержимое `/etc/crontab` имеет следующий вид:

```
SHELL=/bin/bash

PATH=/sbin:/bin:/usr/sbin:/usr/bin

MAILTO=root

HOME=/

#  run-parts

01 * * * * root run-parts /etc/cron.hourly
```

```
02 4 * * * root run-parts /etc/cron.daily

22 4 * * 0 root run-parts /etc/cron.weekly

42 4 1 * * root run-parts /etc/cron.monthly
```

Информация файла указывает, что:

- содержимое каталога `/etc/cron.hourly` будет запускаться каждый час на первой минуте часа;
- содержимое каталога `/etc/cron.daily` будет запускаться каждый день на второй минуте четвертого часа;
- содержимое каталога `/etc/cron.weekly` будет запускаться каждое воскресенье на 22-й минуте 4-го часа;
- содержимое каталога `/etc/cron.monthly` будет запускаться каждый первый день месяца на 42-й минуте 4-го часа.
- `SHELL=/bin/bash` означает использование для запуска команд `/bin/bash`. Если переменная не указана, то значение будет взято из `/etc/passwd` для пользователя, являющегося владельцем файла.
- `HOME=/` — корневой каталог для пользователя (параметр необязательный). При необходимости доступа к специальным свойствам интерпретатора значения переменных `SHELL` и `HOME` можно изменить, независимо от того, что прописано в `/etc/passwd`.
- `MAILTO=root` регистрирует, кому нужно отсылать сообщение о результате работы команд.

Все содержимое из этих каталогов будет запускаться с правами доступа пользователя `root`, а файлы должны иметь права доступа на выполнение, поэтому перед размещением файлов в одном из этих каталогов необходимо убедиться, что сценарии не нанесут вред системе.

После того, как демон `cron` запущен и прочел содержимое всех файлов `crontab`, он бездействует, но просыпается каждую минуту и проверяет, не требуется ли запуск какой-либо команды в данную минуту или не появился ли новый файл `crontab`, который необходимо обработать. Демон `cron` определяет изменения по времени модификации файлов или каталогов, это свойство избавляет от необходимости перезапуска демона.

Как отмечалось выше, размещение файлов для `cron` в каталогах

- `/etc/cron.hourly`

- `/etc/cron.daily`
- `/etc/cron.weekly`
- `/etc/cron.monthly`

доступно только пользователю `root`. Для использования файлов `crontab` пользователями нужно применять команду `crontab`, она служит для создания, изменения и добавления файла для демона `cron`.

ПАМЯТКА ПО РАБОТЕ С CRONTAB

- **crontab -r** — удаление файла.
- **crontab -e** — редактирование файла.
- **crontab -u user_name file** — создание файла `crontab` из файла «file» для пользователя «user_name».
- **crontab -u** означает, чей `crontab` будет обработан. Если опция не задана, то будет обработан `crontab` того пользователя, который запустил команду `crontab`.
- **crontab -u user_name -l** — просмотр файла `crontab` пользователя «user_name».
- **crontab -u user_name -r** — удаление файла `crontab` пользователя «user_name».
- **crontab -u user_name -e** — редактирование файла `crontab` пользователя «user_name» используя редактор, заданный переменной окружения `VISUAL` или `EDITOR`.

Каждая команда в пользовательском файле `crontab` занимает одну строку и состоит из шести полей. Пользовательские файлы `crontab` находятся в каталоге `/var/spool/cron`. Общий формат команды: **минута час день_месяца месяц день_недели команда.**

Допустимые значения:

- минута: от 0 до 59;
- час: от 0 до 23;
- день_месяца: от 1 до 31;
- месяц: от 1 до 12 (можно три буквы из названия месяца, регистр не имеет значения от jan до dec);
- день_недели: от 0 до 6 (0 это воскресенье, можно писать от sun до sat).

Каждое из полей даты и времени может быть обозначено символом *. В этом случае оно будет соответствовать любому возможному значению. Для этих полей можно указывать диапазоны значений, разделенных дефисом, например:

```
* 5 4-10 0-3 * echo "HELLO"
```

Эта команда означает печать слова HELLO в 5:00 на 4, 5, 6, 7, 8, 9, 10 дни января, февраля, марта и апреля.

Для более детальных настроек регулярности можно использовать пошаговую запись, например:

```
* */2 * * sat echo "HELLO"
```

Эта команда означает печать HELLO каждый четный час каждой субботы.

Эту же команду можно написать двумя другими способами:

```
* 0,2,4,6,8,10,12,14,16,18,20,22 * * sat echo "HELLO"
* 0-23/2 * * sat echo "HELLO"
```

Бонусный пример без пояснения напоследок:

```
59 23 31 dec * echo "Happy new year"
```

Для отладки задания `cron` можно перенаправить результат в файл:

```
0-59 * * * * /home/user/mail 2>/tmp/tmp.cron
```

А вот пример работы сценария `mycrontest.sh`, который использовался ранее. Здесь показан вывод, который отправляется пользователю после запуска задания. Обратите внимание, он более компактный, чем тот, что отсылается функцией `cron`:

```
$ at -f mycrontest.sh -v 10:25

Sat Jul 7 10:25:00 2016

job 5 at Sat Jul 7 10:25:00 2016

From student@example.com Sat Jul 7 10:25:00 2016
```

Указание времени может быть достаточно сложным. Более подробную информацию можно найти на страницах справочника `man`, посвященных команде `at`.

Если файл `/etc/cron.allow` существует, любой пользователь, не являющийся суперпользователем `root`, должен быть указан в нем, чтобы иметь возможность использовать `crontab` и `cron`. Если такого файла нет, но есть `/etc/cron.deny`, пользователь, не являющийся суперпользователем `root` и попавший в список из этого файла, **не сможет** пользоваться `crontab` или функцией `cron`. Если нет ни того ни другого файла, то только суперпользователю будет позволено использовать эту команду. Пустой файл `/etc/cron.deny` (а по умолчанию он пустой) дает возможность всем пользователям работать с функцией `cron`.

ПЛАНИРОВАНИЕ ЗАДАЧ С ПОМОЩЬЮ УТИЛИТЫ AT

Напомним вам, что at — это демон, утилита. Она отвечает за запуск команды в заданный момент времени в будущем. Для её реализации нужно воспользоваться стандартным вводом. Команда будет запущена, как если бы она была введена в текущей оболочке без использования демона. `At` беспокоится даже о том, чтобы сохранить текущее окружение и воспроизвести определенные условия при вызове команды.

Время указывается в соответствии со следующими соглашениями: 16:12 или 4:12 pm означают 4:12 после полудня. Дата может быть задана в разных европейских и западных форматах, в том числе:

- ДД.ММ.ГГ — 27.07.12 или 27 июля 2012 года;
- ГГГГ-ММ-ДД — то же, что и 2012-07-27;
- ММ/ДД/ГГ — 07/27/12 или 07/27/2012, то есть 27 июля 2012 года;
- ММДДГГ — 072712 или 07272012 — все то же 27 июля 2012 года.

Без даты команда будет запущена, как только часы подойдут к указанному времени (в тот же день или на следующий день, если сегодня это время уже прошло). Можно также просто написать «today» или «tomorrow» — сегодня или завтра, соответственно.

```
$ at 09:00 27.07.16 <<END
> echo "Не забудь поздравить Иванова с днём рождения!" \
> | mail user@examle.com
> END
warning: commands will be executed using /bin/sh
job 31 at Fri Jul 27 09:00:00 2016
```

ЕЩЕ НЕСКОЛЬКО ПОЛЕЗНЫХ ФУНКЦИЙ

Запуск команды можно отложить на на заданный промежуток времени. Для этого стоит использовать альтернативный синтаксис `at now + «число» / «период»` . Значение «число» указывает число указанных единиц, которое должно пройти перед запуском программы, а «период» может быть в минутах, часах, днях или неделях.

Для отмены задачи, запланированной cron, нужно просто запустить `crontab -e` и удалить соответствующую строку в файле `crontab`. Для задач `at` это почти так же легко: надо запустить atrm-номер задачи. Номер задачи указывается командой `at` при ее планировании, а также ее можно найти с помощью команды `atq`, выводящей текущий список запланированных задач.

И, наверное, самое классное: возможность планировать асинхронные задачи! Это делается с помощью `anacron`. `Anacron` — это демон, дополняющий `cron` на компьютерах, которые не включены все время. Поскольку регулярные задачи обычно планируются на середину ночи, они не запустятся, если компьютер в это время выключен. Назначение `anacron` — запустить их, принимая во внимание периоды, в которые компьютер не работает.

Модуль 3. Юнит 4. Тест для самоконтроля 2

1. Что будет делать планировщик исходя из данной записи - */2 * * sat echo "HELLO"?
    - печать HELLO каждый час, каждую субботу
    - **печать HELLO каждый четный час, каждую субботу**
    - печать HELLO каждые полчаса, каждую субботу
    - печать HELLO каждую минуту

2. Какой командой можно посмотреть содержимое файла планировщика для текущего пользователя?
    - show crontab
    - **crontab –l**
    - cat /etc/crontab
    - cat crontab

3. Какой командой можно удалить файл crontab пользователя user1?
    - **crontab -u user1 –r**
    - del crontab
    - rm crontab
    - crontab user1 -r

Модуль 3. Юнит 5. Контрольное задание

Задание на подстановку слова из выпадающего списка (синим выделен нужный вариант)

Задание: Путем выбора из выпадающего списка необходимо вставить пропущенные слова.

При загрузке системы запускается демон [cron / at / crontab / /var/spool/cron / /etc/passwd] и проверяет очередь заданий [cron / at / crontab / /var/spool/cron / /etc/passwd] и заданий пользователей в файлах [cron / at / crontab / /var/spool/cron / /etc/passwd]. При запуске демон [cron / at / crontab / /var/spool/cron / /etc/passwd] сначала проверяет каталог [cron / at / crontab / /var/spool/cron / /etc/passwd] на наличие файлов [cron / at / crontab / /var/spool/cron / /etc/passwd], файлы [cron / at / crontab / /var/spool/cron / /etc/passwd] имеют имена пользователей, соответствующие именам пользователей из [cron / at / crontab / /var/spool/cron / /etc/passwd]. Каждый пользователь может иметь только один файл [cron / at / crontab / /var/spool/cron / /etc/passwd], записей в файле может быть несколько.

Модуль 4. Администрирование пользователей в Linux

Содержание модуля:

В данном модуле рассматривается управление пользователями через терминал. Создание, удаление, изменение информации для пользователей и пользовательских групп, редактирование конфигурационных файлов. Слушатели знакомятся с профилем пользователя по умолчанию. Также объясняется тема прав доступа, понятие суперпользователя.

Модуль 4. Юнит 1. Текстовая лекция Администрирование пользователей

АДМИНИСТРИРОВАНИЕ ПОЛЬЗОВАТЕЛЕЙ

Управление пользователями заключается в добавлении и удалении пользователей и предусматривает:

- добавление имен пользователей для возможности их работы в системе;
- определение их привилегий;
- создание и назначение рабочих каталогов;
- определение групп пользователей;
- удаление имен пользователей.

Каждый пользователь должен иметь уникальное регистрационное имя, дающее возможность идентифицировать его и избежать ситуации, когда один пользователь может стереть файлы другого. Кроме того, каждый пользователь должен иметь свой пароль для входа в систему.

Идентификатор пользователя. При авторизации Linux связывает входное имя пользователя с его идентификатором в системе UID (User ID).

UID — это положительное целое число, по которому система отслеживает пользователей. Оно уникально и однозначно идентифицирует учетную запись пользователя. Таким числом снабжены все процессы и все объекты файловой системы в. Linux. Оно используется для персонального учета действий пользователя и определения прав доступа к другим объектам системы.

Идентификатор группы. Он тоже связан с учетными записями, только теперь не единичных пользователей, а целых групп.

Группы пользователей применяются для организации одновременного доступа нескольких пользователей к некоторым ресурсам. У группы так же, как и у пользователя, есть имя и идентификационный номер GID (Group ID). В Linux пользователь должен принадлежать как минимум к одной группе — группе по умолчанию. При создании учетной записи пользователя обычно создается и группа, имя которой совпадает с входным именем, она будет группой по умолчанию для этого пользователя. Пользователь может входить более чем в одну группу, но в учетной записи указывается только номер группы по умолчанию.

Пользователь root. В Linux есть один пользователь, полномочия которого в системе принципиально отличаются от полномочий остальных пользователей — это пользователь с идентификатором «0».

Обычно учетная запись пользователя с UID=0 называется `root` (от англ. «корень»). Пользователю с таким UID разрешено выполнять любые действия в системе, а значит, любая ошибка или неправильное действие могут повредить систему, уничтожить данные и привести к другим печальным последствиям. Для повседневной работы категорически не рекомендуется регистрироваться в системе под именем `root` именно по этой причине. Работать в `root` следует только тогда, когда это действительно необходимо: при настройке и обновлении системы, восстановлении после сбоев.

Теперь поговорим об этом всем чуть подробнее!

ОДНОВРЕМЕННЫЙ ДОСТУП К СИСТЕМЕ

Каждый компьютер, на котором работает Linux, предоставляет возможность зарегистрироваться и получить доступ к системе одновременно нескольким пользователям. По умолчанию в любом Linux создается несколько привилегированных пользователей, которые владеют системными файлами и большинством процессов, запускаемых в системе. Главным из таких пользователей является `root` — это администратор, которому принадлежат, среди всех прочих, **права управлять остальными пользователями в системе**.

В ОС Linux очень удобно реализована поддержка пользователей. Для того, чтобы человек мог выполнять какие-то действия с системой, администратор должен зарегистрировать его в системе — выдать ему имя и пароль, создать для него каталог, в котором пользователь окажется сразу же после успешного входа в систему.

Кроме администратора есть еще несколько идентификаторов пользователей, которые автоматически создаются системой при установке: daemon (uid=1), bin (uid=2), sys (uid=3), adm (uid=4), lp, uucp и nobody. Эти системные аккаунты используются автоматически для разделения и безопасного выполнения системных задач — чтобы многие операции можно было запускать с полномочиями этих аккаунтов, а не суперпользовательскими.

Выжимка из файла `/etc/passwd`, список системных аккаунтов:

```
root:x:0:1:Super-User:/:/sbin/sh

daemon:x:1:1::/:

bin:x:2:2::/usr/bin:
```

```
sys:x:3:3::/:

adm:x:4:4:Admin:/var/adm:
```

С такими же целями создаются и несколько системных групп: `root` (или `wheel` в некоторых системах*), `c gid=0, other (gid=1), bin, sys, adm`. Все эти группы включают вышеописанных пользователей и суперпользователя.

Список служебных групп в системе из файла `/etc/group` :

```
root::0:root

other::1:

bin::2:root,bin,daemon

sys::3:root,bin,sys,adm

adm::4:root,adm,daemon

uucp::5:root,uucp

mail::6:root
```

Каждый зарегистрированный пользователь Linux имеет следующие параметры:

- идентификатор пользователя (UID);
- идентификатор группы (GID);
- пароль (password);
- срок действия пароля (expiry date);
- домашний каталог (home directory);

  При входе в систему пользователь попадает в свой домашний каталог. Обычно пользователь имеет права на создание и удаление, запись, чтение и выполнение любых файлов в своем домашнем каталоге. К каталогу пользователя может иметь доступ только он сам или при желании пользователи его группы.

- полное имя (full name или GECOS);

По символьному имени пользователя типа student не очень-то много можно разобрать. Для этих целей предусмотрена возможность сопоставлять с именем пользователя его реальное имя и фамилию. Это один из простейших вариантов заполнения поля.

На самом деле можно указать несколько блоков дополнительных сведений, разделяя их знаком запятой: полное имя; номер офиса, где работает пользователь; рабочий и домашний телефон. Таким образом, запись «Mr Student, 115, 12345678, 87654321 в этом поле будет обозначать пользователя, чье полное имя Mr Student, который находится в кабинете 115, с рабочим телефоном 12345678 и домашним телефоном 87654321.

- командная оболочка (shell).

Специальная программа, позволяющая пользователю вводить определенные команды для выполнения необходимых действий. Именно оболочка запускается в первую очередь, давая приглашение к вводу дальнейших команд.

Пароли — это отдельная интересная тема. Сейчас мы быстро пройдемся по тем особенностям хранения паролей, которые касаются работы с пользователями в системе.

Пример файла `/etc/shadow`:

```
root:f6ChQ5l9GCx1Q:6445::::::

daemon:*:6445::::::

bin:*:6445::::::

sys:*:6445::::::

adm:*:6445::::::

listen:!:::::::

nobody:*:6445::::::

student:f6ChQ5l9GCx1Q:11658::::::
```

Файл `shadow` представляет собой таблицу, каждая строка которой состоит из 9 полей, разделенных двоеточиями: регистрационное имя, хэшированный пароль, контрольные сроки в днях, среди которых:

- число дней с 1.01.70 г. до дня последнего изменения пароля;
- минимальное число дней действия пароля со дня его последнего изменения;
- максимальное число дней действия пароля;

- за сколько дней до устаревания пароля система начнет выдавать предупреждения;
- число дней со времени обязательной смены пароля до блокировки учетной записи;
- день блокировки учетной записи;
- последнее, девятое поле зарезервировано и не используется.

Таким образом, в `/etc/passwd` хранится большая часть информации, но зашифрованный пароль хранится в файле `/etc/shadow`. Пароли — ключевой момент администрирования пользователей Linux. К их подбору следует подходить с тщательностью и серьезностью, потому что неверно выбранный пароль может привести к непоправимым последствиям.

УТИЛИТЫ АДМИНИСТРИРОВАНИЯ УЧЕТНЫХ ЗАПИСЕЙ ПОЛЬЗОВАТЕЛЕЙ

При добавлении пользователя в файл `/etc/passwd` вносится учетная запись в форме:

```
login_name: encrypted_password: user_ ID: group_ ID: user_ information:

login_directory: login_shell
```

В этой записи поля разделены двоеточиями, а значения этих полей приведены в таблице:

Поле	Назначение
login_name	Регистрационное имя пользователя
encrypted_password	Указатель на теневой файл паролей (shadow)
user_ID	Уникальный номер, используемый ОС для идентификации пользователя. Для локальных пользователей не должен превышать 2499

group_ID	Уникальный номер или имя, используемые для идентификации первичной группы пользователя. Если пользователь является членом нескольких групп, он может в процессе работы менять группу, если это разрешено системным администратором.
user_information	Описание пользователя, например, его имя и должность
login_directory	Рабочий каталог пользователя (в котором он оказывается после входа в систему)
login_shell	Оболочка, используемая пользователем, после входа в систему (например, **/bin/bash**)

Также описание файла `/etc/passwd` приведено в `man passwd`.

В каждой системе Linux обязательно имеются команды администрирования пользователей: их добавления и удаления, а также изменения их свойств. Команда добавления называется `useradd`, а команда удаления — `userdel`. В большинстве систем имеется также скрипт `adduser`, который является надстройкой над командой `useradd` и позволяет производить добавление пользователей с большим комфортом за счет интерактивности. Ниже мы рассмотрим несколько примеров применения команд `useradd` и `userdel`.

Для добавления пользователя применяется команда `adduser` с именем добавляемого пользователя в качестве параметра, например:

```
#adduser User1
```

Команда `adduser` добавляет пользователя, создает домашний каталог, создает почтовый ящик, а также копирует файлы, имена которых начинаются с точки, из каталога `/etc/skel` в рабочий каталог пользователя. Он должен содержать все файлы-шаблоны, которые имеет каждый пользователь. Обычно это персональные конфигурационные файлы, такие как .profile, .cshrc и .login, для настройки оболочки.

Команда `adduser` представляет собой файл сценария bash, находящийся в каталоге `/usr/sbin`. Можно добавить запрос дополнительной информации о пользователе. Чтобы это сделать, необходимо воспользоваться командой `chfn` для изменения стандартных записей о пользователе. Описание команд приведено в `man adduser` и `man chfn`.

А если хочется рвать, метать и удалять пользователей, то вот инструкция ↴

Есть несколько степеней удаления пользователя:

- лишение пользователя возможности входа в систему;

  Полезно в случае его длительного перерыва в работе.

- удаление учетной записи;
- удаление пользователя и всех его файлов.

На время отсутствия пользователя можно заблокировать его запись с помощью команды:

```
#usermod -L user_name
```

При этом все пользовательские файлы и каталоги остаются нетронутыми, но войти в систему под его именем становится невозможно.

Для разблокировки записи необходимо выполнить команду:

```
#usermod -U user_name
```

Одним из вариантов лишения пользователя возможности входа в систему может быть смена имени пользователя. При этом вход под старым именем становится невозможным. Для этого необходимо выполнить команду:

```
#usermod -l new_user_name old_user_name
```

Удаление учетной записи пользователя производится либо путем непосредственного редактирования файла `/etc/passwd`, либо с помощью команды:

```
#deluser user_name
```

По умолчанию учетная запись удаляется без удаления домашнего каталога и файлов системы, принадлежащих удаляемому пользователю. Для удаления домашнего каталога может использоваться дополнительный параметр — `remove-home`, а для поиска и удаления всех файлов системы, принадлежащих удаляемому пользователю, — параметр `remove-all-files`.

Если же необходимо удалить и домашнюю директорию, то следует использовать ключ `-r`:

```
#userdel -r student
```

В Astra Linux для работы с группами служат команды `groupadd`, `groupmod` и `groupdel`. Группы перечислены в файле `/etc/group`. Каждая запись этого файла состоит из нескольких полей, разделенных двоеточиями. Это следующие поля:

- имя группы;
- пароль;

  Обычно это звездочка, которая позволяет входить в состав группы любому пользователю, для управления доступом можно добавить пароль;

- идентификатор группы, номер, которым система обозначает данную группу;
- пользователи, список пользователей, относящихся к данной группе.

Ниже приведен пример записи из файла `/etc/group`:

```
engines::100:chris,robert,valerie,aleina
```

Давайте разберемся, что это значит. Группа называется engines, пароля нет, идентификатор группы – 100, в группу входят пользователи chris, robert, valerie и aleina.

Составим несколько коротких памяток с командами, о которых шла речь в лекции.

Управление пользователями и группами:

- useradd добавляет нового пользователя;
- passwd устанавливает пароль пользователя;
- usermod изменяет параметры учетной записи пользователя;
- userdel удаляет учетную запись пользователя.

Команды управления группами:

- groupadd создает новую группу;
- gpasswd устанавливает пароль группы;
- groupmod изменяет параметры группы;
- groupdel удаляет группу.

Дополнительная информация располагается в файлах:

- /etc/default/useradd — свойства по умолчанию для новых пользователей;

- /etc/login.defs — настройки новых пользователей;
- /etc/skel — каталог, файлы из которого копируются в домашний каталог нового пользователя.

Модуль 4. Юнит 2. Тест для самоконтроля

1. Могут ли в одной системе существовать два пользователя с одинаковым UID?
    - Да, могут
    - **Нет, не могут**
    - Могут, если один из них имеет привилегии суперпользователя
    - Могут, если у них по несколько UID

   (Вопрос с множественным выбором):
2. С помощью каких команд можно добавить нового пользователя в систему?
    - ✓ **adduser**
    - ✓ **useradd**
    - ✓ passwd
    - ✓ usermod

3. Сколько полей отведено для каждой строки пароля в файле /etc/shadow?
    - 3
    - 5
    - **9**
    - 11

Модуль 4. Юнит 3. Учебное задание.

1. Задание на определение порядка элементов:

Задание: Составьте содержимое строки файла /etc/group в верном порядке:
Элементы: «1005» «studentgroup» «:» «:» «:» «Alex»
Верно составленная строка: studentgroup::1005:Alex

2. Задание на заполнение поля. Порядок может меняться (неважно, какой из верных ответов запишут первым, какой вторым)

Задание: Напишите 2 утилиты, с помощью которых можно добавить в систему нового пользователя.
Поле1: useradd Поле2: adduser

Модуль 4. Юнит 4. Контрольное задание.

Задание на множественный выбор:

Задание: Выберите, какие параметры имеет каждый пользователь в файле /etc/passwd в операционной системе Linux:
Имя пользователя Дата создания пользователя **Домашний каталог пользователя** Идентификатор создателя пользователя **Зашифрованный пароль** **Цифровой идентификатор пользователя (UID)** **Цифровой идентификатор группы пользователя (GID)** Хеш пароля пользователя **Полное имя пользователя** **Оболочка входа в систему**

Модуль 4. Юнит 5. Памятка Права доступа в ОС Linux.

В системе Linux могут работать одновременно или по очереди несколько или множество пользователей. У каждого из них собственное файловое пространство, доступ к которому других пользователей ограничен. Отсюда вытекает, что у любого файла в Linux должны быть специальные атрибуты – права на доступ. Эти атрибуты должны сообщать, кто имеет право работать с файлом.

rwx представление	Двоичное число	Восьмеричное число	Значение
---	000	0	Все запрещено
--x	001	1	Только исполнение
-w-	010	2	Только запись
-wx	011	3	Только запись и исполнение
r--	100	4	Только чтение
r-x	101	5	Чтение и исполнение
rw-	110	6	Чтение и запись
rwx	111	7	Все разрешено

rwx-представление назначается для трех категорий пользователей т.е. представляет собой три тройки прав. Первая тройка определяет права владельца файла. Вторая тройка определяет права группы владельца файла. Третья тройка для всех остальных пользователей. Также можно представить три тройки в виде двоичных чисел либо трех восьмеричных чисел.

Примеры:

rw-rw-rw- (все могут читать и изменять)

rwx------ (полный доступ имеет владелец файла)

rw-r--r--(все могут читать, владелец также изменять)

rwxr-xr-x (все могут читать и исполнять, владелец также изменять)

Модуль 5. Настройка сетевого подключения

Содержание модуля:

В модуле по настройке сетевого подключения слушатели узнают понятие сетевого интерфейса, сетевого подключения, виды виртуальных сетевых соединений (NAT, Bridge, Host only). Различия, особенности и недостатки данных видов соединений. Будет проводиться первичная настройка параметров сетевого подключения в ОС Linux через терминал. Будет рассмотрена о выполнение сетевых настроек путем редактирования конфигурационных файлов. Также затронуты основы работы с менеджерами пакетов. Apt и dpkg – применение, различия, расположение и структура файлов репозиториев в системе, подключение дополнительных репозиториев, поиск пакетов утилит, проверка зависимостей, проверка версии установленных пакетов, способы установки, обновление, удаление пакетов в ОС Linux. В модуле объясняется понятие службы в ОС Linux. Рассматриваются типы служб, сетевые сервисы, способы управления службами в ОС Linux. Слушатели научатся просматривать статусы состояний служб, познакомятся с понятием лог-файла. Также познакомятся с применением протокола SSH, управлением службой ssh в ОС Linux – установка, настройка конфигурации, запуск, просмотр статуса службы. Рассмотрят варианты подключения по ssh для Windows.

Модуль 5. Юнит 1. Конфигурирование сетевых подключений

РАБОТА С СЕТЕВЫМИ ИНТЕРФЕЙСАМИ

Естественно, что для того, чтобы работать с локальной сетью, необходимо иметь сетевую карту (плату) и подключение к сети. Заметим, что в каталоге /dev нет специального файла для сетевой карты. В Linux сетевые устройства создаются динамически и поэтому не требуют наличия там соответствующих файлов.

ПОЛУЧЕНИЕ СЕТЕВОГО АДРЕСА И УСТАНОВКА ПО

Поскольку вы собираетесь устанавливать машину с Linux в уже существующую сеть, то следующим вашим шагом при подключении к сети должно стать обращение к администратору сети за получением сетевого адреса. Точнее, вы должны получить следующую информацию:

IP-адрес вашего компьютера;

IP-адрес сети;

широковещательный IP-адрес;

имя домена, в который будет включен ваш компьютер;

маску подсети;

IP-адрес маршрутизатора (router);

IP-адрес сервера имен (DNS-сервера).

Раньше необходимо было согласовать с администратором ip-адрес и сетевое имя, которые выбирались для компьютера, чтобы избежать его совпадения с каким-то из уже включенных в сеть компьютеров. В настоящее время по умолчанию ip-адрес назначается автоматически от сервера DHCP, а имя компьютера назначается локальное по умолчанию.

НАСТРОЙКА СЕТЕВЫХ ИНТЕРФЕЙСОВ

Интерфейсом с точки зрения ОС является устройство, через которое система получает и передает сетевые пакеты.

Роль интерфейса локальной сети может выполнять одно (или несколько) из следующих устройств: Ethernet-карта, ISDN-адаптер или модем, подключенный к последовательному порту. Каждое устройство имеет свой IP-адрес. Для выхода в локальные сети используется, как правило, Ethernet-карта, что и будет предполагаться в настоящем разделе.

После подключения драйверов вы должны настроить те интерфейсы, которые предполагаете использовать. Настройка интерфейса заключается в присвоении IP-адресов сетевому устройству и установке нужных значений для других параметров сетевого подключения.

В настоящий момент в большинстве дистрибутивов для этого используется команда ip. Она имеет достаточно широкий функционал для управления сетевыми настройками, их просмотра и т.п. Для использования данного функционала предусмотрено множество опций ее использования (которые можно посмотреть, вызывав справку по команде).

Ранее наиболее часто использовалась утилита ifconfig (находится в пакете net-tools). Ее название происходит от английской фразы interface configuration (настройка интерфейса).

Рассмотрим несколько практических примеров настройки сетевых интерфейсов.

С помощью утилиты ip:

Так, чтобы получить информацию о сетевых интерфейсах в системе и о том, какие ip-адреса им назначены, можно использовать команду:

ip addr show

Включить сетевой интерфейс (например, eth1):

ip link set eth1 up

Отключить его:

ip link set eth0 down

Назначить данному интерфейсу ip-адрес:

ip addr add 192.168.1.15 dev eth1

И удалить назначенный адрес:

ip addr del 192.168.1.15/24 dev eth0

Можно заметить, что первое идущее слово после ip – опция, означающая уровень использования. Т.е. ip link управляет интерфейсами на физическом уровне, ip addr – на сетевом. Более подробно использующиеся опции можно найти в выводе справки о команде ip.

Также полезной может оказаться команда просмотра таблицы маршрутизации:

ip route show

Добавление статического маршрута:

ip route add 10.40.0.0/24 via 192.168.1.20 dev eth0

Удаление статического маршрута:

ip route del 10.40.0.0/24

Добавление шлюза по умолчанию:

ip route add default via 192.168.1.10

Далее рассмотрим другие возможные варианты настройки интерфейсов.

С помощью утилиты ifconfig:

НАСТРОЙКА ЛОКАЛЬНОГО ИНТЕРФЕЙСА LO

Этот интерфейс используется для связи программ IP-клиентов с IP-серверами, запущенными на той же машине, так что его необходимо настроить даже в том случае, если вы вообще не подключаете никаких сетевых устройств. Локальный интерфейс настраивается командой:

```
# /sbin/ifconfig lo 127.0.0.1
```

Теперь, чтобы проверить работоспособность протоколов TCP/IP на вашей машине, дайте команду:

```
# ping 127.0.0.1
```

НАСТРОЙКА ИНТЕРФЕЙСА ПЛАТЫ ETHERNET ЛОКАЛЬНОЙ СЕТИ (ETH0)

Для того чтобы ваш компьютер вошел в сеть с IP-адресом, полученным вами у администратора (пусть для примера это будет адрес 192.168.0.15), вы должны запустить команду ifconfig следующим образом:

```
# /sbin/ifconfig eth0 192.168.0.15 netmask 255.255.255.0 up
```

Если не указывать маску подсети, то по умолчанию устанавливается маска подсети 255.0.0.0.

В некоторых случаях необходимо бывает изменить адрес прерывания, используемого сетевой картой, порта ввода-вывода или типа соединения, используемого в сети. Это можно сделать, выполнив следующую команду:

```
# /sbin/ifconfig eth0 irq 5 io_addr 220 media 10baseT
```

Основную настройку сети можно выполнить, редактируя конфигурационный файл interfaces, который располагается в /etc/network/interfaces. Здесь вы можете задать IP-адрес сетевой карты (или использовать DHCP), настроить маршрутизацию, IP masquerading, установить маршрут по умолчанию и многое другое.

Если вы хотите использовать DHCP, вам необходимо написать следующее:

auto eth0

iface eth0 inet dhcp

А если хотите сконфигурировать вручную, например, задать шлюз по умолчанию:

auto eth0

iface eth0 inet static

address 192.168.0.7

netmask 255.255.255.0

gateway 192.168.0.254

Кстати, кроме шлюза опционально еще можно задать сеть и широковещательный адрес.

ТЕСТИРОВАНИЕ СЕТЕВОГО СОЕДИНЕНИЯ

Чтобы проверить, соединяется ли ваш компьютер с сетью, попробуйте дать команду ping, указав ей в качестве параметра IP-адрес одного из компьютеров сети. Пусть, например, вам известно (узнайте реальный номер и имя у администратора сети), что в сети есть компьютер с IP-адресом 192.168.0.2 и именем pc1. Тогда вы должны дать команду:

$ ping 192.168.0.2

или (тут вы одновременно проверяете и работу службы DNS):

$ ping pc1

Если соединение с сетью установлено, должны появиться и периодически обновляться строчки примерно такого вида:

64 bytes from 192.168.0.2: icmp_seq=0 ttl=32 time=1.2 ms

64 bytes from 192.168.0.2: icmp_seq=1 ttl=32 time=1.0 ms

Это означает, что сетевое соединение работает. Для того чтобы прервать тестирование сети, нажмите комбинацию клавиш Ctrl+C.

Полезный совет: иногда сетевые настройки могут примениться не сразу. Поэтому, если вы все сделали верно, но выход в интернет не происходит – попробуйте перезагрузить систему.

Модуль 5. Юнит 2. Тест для самоконтроля.

1. Что можно назвать MAC-адресом устройства?
    - **F4-D2-32-12-43-FD**
    - example@student
    - example.com
    - 103.17.10.123

(Вопрос с множественным выбором):

2. Что из перечисленного позволяет сделать команда ip?
    - **Удалить ip-адрес для интерфейса**
    - Отправить echo-запрос к другому хосту
    - **Показать таблицу маршрутизации**
    - Назначить имя хоста

3. Как проверить соединение компьютера с другими устройствами в сети?
    - trace
    - search
    - load
    - **ping**

Модуль 5. Юнит 3. Текстовая лекция Менеджер пакетов.

МЕНЕДЖЕР ПАКЕТОВ

Зачастую при работе с системой мы сталкиваемся с необходимостью устанавливать дополнительное ПО. И в случае с Windows это обычно связано с поиском необходимого ПО в интернете, дальнейшим скачиванием и его установкой. В Linux есть большие базы различного ПО, готового к скачиванию и адаптированного под тот или иной дистрибутив. Такое ПО обычно называют в Linux пакетами. Хранятся пакеты в репозиториях – специальных серверах в интернете, адаптированных для хранения различных пакетов для Linux. Когда какой-либо хост посылает запрос на скачивание какого-то пакета, то его запрос передается одному из нужных серверов, на которых находится репозиторий. Адреса таких серверов постоянные. Чтобы хост «знал», к какому серверу ему нужно обратиться для получения того или иного пакета, адреса серверов-репозиториев хранятся в специальном файле: /etc/apt/sources.list. Адреса репозиториев записаны здесь в виде строк с некоторыми параметрами. Строки следуют друг за другом в порядке их приоритета – от наиболее важных к наименее. В первую очередь система обращается к наиболее важному репозиторию, если там искомого пакета не найдется – то к следующему, и т.д., пока не дойдет до конца. Бывает, что репозиторием является не адрес сервера в интернете, а подключенный установочный образ системы. Особенно часто это используется в защищенный системах, в которых не предполагается доступ к интернету. Кстати, файл /etc/apt/sources.list можно редактировать – пользователь сам может добавлять в него строки репозиториев и удалять ненужные.

Но как именно происходит обращение к репозиторию? В этом пользователю помогает специальная утилита – менеджер пакетов в Linux. Менеджер пакетов позволяет устанавливать пакеты в ОС, настраивать зависимости (в тех случаях, когда, например, работа одного пакета зависит от наличия в системе другого), искать нужные пакеты, удалять их и т.д. Такая установка утилит в систему значительно упрощает данный процесс и экономит время пользователя. Использование менеджера пакетов позволяет не искать нужное ПО на различных сайтах и не задаваться вопросами совместимости найденного – чтобы что-то установить пользователь просто отправляет необходимому менеджеру пакетов запрос install с именем нужного пакета и менеджер пакетов делает оставшуюся работу сам, вплоть до запуска устанавливаемой службы.

Для разных дистрибутивов предусмотрены различные пакетные менеджеры. Так, в Debian подобных дистрибутивах обычно используются dpkg и apt. Dpkg в данном случае – базовая система управления пакетами в Debian. Он более низкоуровневый инструмент, чем apt. Apt же, по сути, намного расширяет возможности dpkg, а также является более популярным на данный момент. Приведем несколько примеров его использования.

Обновление баз данных пакетов:

apt update

Обновление системы:

apt upgrade

Найти пакет с названием, например, «utilite»:

apt search utilite

Показать информацию о пакете с названием «utilite»:

apt show utilite

Установить пакет с названием «utilite»:

apt install utilite

Удалить пакет с названием «utilite»:

apt remove utilite

Очистить кэш пакетов:

apt clean

Существует графический менеджер пакетов – Synaptic. Оснащен удобным графическим интерфейсом, но включает в себя функционал apt.

В Red Hat подобных дистрибутивах используются в основном rpm – более низкоуровневый инструмент, и yum – высокоуровневая надстройка.

Примеры использования yum:

Установить пакет с названием «utilite»:

yum install utilite

Удалить пакет с названием «utilite»:

yum remove utilite

Обновление пакета с названием «utilite»:

yum update utilite

Найти пакет с названием «utilite»:

yum list utilite

Получить информацию о пакете с названием «utilite»:

yum info utilite

В Arch Linux используется менеджер пакетов Pacman, в Linux Opensuse – Zypper, в Linux Gentoo – Portage.

Чтобы получить более подробную информацию по используемому менеджеру пакетов можно вызвать справку, где будут показаны возможные варианты использования.

Модуль 5. Юнит 4. Тест для самоконтроля 2

1. Для установки пакета с помощью менеджера пакета чаще всего используется опция
    - remove
    - list
    - show
    - **install**

2. Если вы хотите, чтобы добавленный в /etc/apt/ sources.list репозиторий проверялся в первую очередь, то необходимо
    - **добавить его в начало файла**
    - добавить его в конец файла
    - ввести параметр приоритета – 0
    - ничего не нужно, он будет проверен первым по умолчанию, т.к. он новый

3. С помощью менеджера пакетов не получится:
    - устанавливать пакеты в ОС
    - просматривать информацию о пакетах
    - **конфигурировать файлы настроек служб**
    - очищать кэш пакетов

Модуль 5. Юнит 5. Текстовая лекция Сетевые службы

СЕТЕВЫЕ СЛУЖБЫ

В операционной системе Linux есть так называемые службы – это приложения, работающие в фоновом режиме и запускающиеся обычно без вмешательства пользователя. Большинство служб запускаются сразу при старте ОС, еще до того как какой-либо пользователь авторизовался в системе. Есть несколько типов служб. Так, например, есть сервисы – они могут быть включены, отключены, управляться пользователем. Существуют службы, которые запускаются, выполняют функцию и выключаются. Есть так называемые демоны – они как раз постоянно запущены в фоновом режиме и не связаны с управляющим терминалом. Многие такие демоны для своего функционирования взаимодействуют с сетью, либо даже выполняют основной функционал посредством сетевого взаимодействия. Они и называются сетевыми службами. Рассмотрим подробнее несколько сетевых служб.

SSHD

sshd (или OpenSSH Daemon) – это служба, принимающая запросы по протоколу SSH на соединения от клиентов. (Для справки: протокол SSH – протокол удаленного администрирования операционных систем. Особенностью данного протокола является обеспечение безопасности соединения, которая обусловлена использованием различных алгоритмов шифрования. SSH использует клиент-серверную модель для аутентификации пользователей и получения удаленного доступа к системе. По умолчанию работает на 22-TCP порту.) Так, установленная в системе служба sshd запускается при старте системы. При том для каждого нового соединения создается отдельный экземпляр службы. Данный экземпляр будет отвечать за обмен ключами, шифрование, аутентификацию, выполнение команд и обмен данными в рамках «его» соединения. Служба OpenSSH по умолчанию поддерживает две версии протокола SSH – 1 и 2. Протокол 1 версии поддерживает только ключи RSA, протокол 2 версии – RSA и DSA. Независимо от протокола, каждый хост имеет собственный (обычно 2048-битный) идентифицирующий его ключ.

HTTPD

Служба httpd – основная программа веб-сервера Apache, работающая с протоколом передачи гипертекста http. httpd также по умолчанию является фоновым процессом, который по мере необходимости создает дочерние процессы и потоки обработки запросов. По умолчанию данной службой используются порты 80-TCP и 443-TCP (для безопасного https соединения). Данная служба принимает

запросы от клиентов, которые обращаются к веб-серверу на хосте, устанавливает с ними соединения.

DHCPD

Служба isc-dhcp-server по сути является службой dhcp сервера для Linux. DHCP-сервер – сервер, отвечающий за предоставление ip-адресов компьютерам и другим устройствам в сети. Клиент, настроенный на получение адреса по протоколу DHCP, посылает запрос к серверу (т.е. к нашей службе isc-dhcp-server), и сервер предоставляет свободный IP адрес клиенту во временное пользование. Служба также запускается автоматически в фоновом режиме и ждет запросов клиентов по умолчанию на 67, 68-UDP портах.

VSFTPD

vsftpd – это служба ftp-сервера по умолчанию для многих операционных систем. Ftp сервер по своей сути является сервером для обмена файлами. Служба vsftpd работает (как и многие другие) в фоновом режиме, ожидая и принимаю запросы клиентов. Запросы могут обрабатываться как от анонимных, так и от зарегистрированных на сервере пользователей. По умолчанию служба vsftpd работает на 20-TCP порту.

Модуль 5. Юнит 6. Тест для самоконтроля 3

1. В качестве отличительной особенности сетевых служб можно отметить
    - Работу в фоновом режиме
    - **Сетевые взаимодействия**
    - Запуск при загрузке ОС
    - Запуск при авторизации первого пользователя

2. Какая из перечисленных служб предназначена для удаленного администрирования системы?
    - vsftpd
    - isc-dhcp-server
    - postfix
    - **sshd**

3. Служба ssh запускается по умолчанию на
    - 20-TCP порту
    - **22-TCP порту**
    - 53-TCP порту
    - 68-UDP порту

Модуль 5. Юнит 7. Учебное задание

Задание с заполнением поля

Задание: Введите команду для проверки соединения с внутренним локальным интерфейсом lo.
Поле: **ping 127.0.0.1**

Модуль 5. Юнит 8 Контрольное задание

Задание на перетаскивание объектов

Задание: Расположите в верном порядке строки из файла /etc/network/interfaces для настройки статического адреса для интерфейса eth4
Строки: «address 10.40.240.3» «gateway 10.40.240.1» «auto eth4» «netmask 255.255.255.0» «iface eth4 inet static»
Верный порядок: auto eth4 iface eth4 inet static address 10.40.240.3 netmask 255.255.255.0 gateway 10.40.240.1

Модуль 5. Юнит 9. Промежуточный контроль.

1. Могут ли в одной системе существовать два пользователя с одинаковым UID?
- Да, могут
- **Нет, не могут**
- Могут, если один из них имеет привилегии суперпользователя
- Могут, если у них по несколько UID

2. Сколько полей отведено для каждой строки пароля в файле /etc/shadow?
- 3
- 5
- **9**
- 11

3. Как проверить соединение компьютера с другими устройствами в сети?
- **ping**
- search
- load
- telnet

4. С помощью какой команды можно добавить нового пользователя в систему?
- passwd
- usermod
- **useradd**
- **adduser**

5. Пользователь был создан с использованием команды **#useradd student**. В какой директории окажется student после того, как войдет в систему?
- /home
- /home/student
- **/**
- /student/home

6. С помощью какой команды можно посмотреть наличие и настойки сетевых интерфейсов?
- show run
- **ifconfig**
- ping
- ipconfig

7. Посредством редактирования какого файла можно изменять информацию о сетевых интерфейсах?

- /etc/nsswitch.conf
- /etc/shadow
- /etc/hosts
- **/etc/network/interfaces**

8. В каком файле можно увидеть все группы, в которых состоит пользователь?

- /etc/passwd
- **/etc/group**
- /etc/skel
- /etc/shadow

9. Какая команда позволяет посмотреть текущую таблицу маршрутизации?

- lspci | grep Ether
- ifconfig
- dmesg | grep eth
- **ip route show**

10. Какой параметр НЕ относится к настройке интерфейса при статической адресации?

- **route**
- address
- gateway
- netmask

11. Где хранится зашифрованный пароль пользователя?

- **/etc/shadow**
- /bin/passwd
- /etc/passwd
- /etc/ssh

12. Из какого каталога при создании пользователя данный копируются в домашнюю директории пользователя?

- /etc/home/
- /etc/default/
- /usr/
- **/etc/skel/**

Модуль 6. Работа с файловой системой

Содержание модуля:

В этом модуле слушателям предлагается работа с файловой системой в ОС Linux. Рассматривается структура ФС ext4, понятие файла и директории в Linux. Создание, редактирование, удаление объектов файловой системы. Частично повторяется материал по правам доступа и атрибутам файлов и директорий. Рассматривается назначение системных директорий в Linux. Также в данном модуле слушатели узнают команды для начала работы с графическими текстовыми редакторами – nano, vim. Изучат интерфейс, сочетания клавиш nano и, аналогично, vim. Смогут создавать и редактировать файлы с помощью консольных текстовых редакторов.

СОЗДАНИЕ И УДАЛЕНИЕ ФАЙЛОВ. ОТОБРАЖЕНИЕ ФАЙЛОВ НА ВНЕШНЮЮ ПАМЯТЬ

ОС выделяет внешнюю память при создании каждого нового файла. Файл в большинстве файловых систем состоит из **заголовка** и **памяти**.

В заголовке хранятся атрибуты файла, например, его длина, тип, ссылка на элементы файла во внешней памяти. Кроме создания и удаления файла, основные операции над ним — открытие и закрытие.

Открытие файла — это считывание в основную память его заголовка и, возможно, одного или нескольких соседних блоков. Оно должно быть выполнено перед выполнением операций чтения из файла или записи в файл.

Закрытие файла — это обратная операция: сброс всех копий блоков на внешнюю память и освобождение областей основной памяти, занятых открытым файлом.

ОС закрывает файлы процесса при его завершении, если процесс не сделал этого сам (последнее рекомендуется).

При отображении файлов на внешнюю память возникают проблемы, аналогичные проблемам распределения основной памяти, — фрагментация, возможность исчерпания внешней памяти или ее раздела (partition) – смежной области внешней памяти, имеющей определенное символьное обозначение.

СОЗДАНИЕ И УДАЛЕНИЕ ДИРЕКТОРИЙ

Директория (directory) — это каталог-справочник ссылок на группу файлов или других директорий, каждые из которых имеет в данной директории свое уникальное символьное имя. Иерархия директорий позволяет организовать поиск файла по его символьному пути (path).

РЕЗЕРВНОЕ КОПИРОВАНИЕ

Резервное копирование — это копирование (backup) файлов на устойчивые носители (флэш-память, компакт-диск и др.) с целью их последующего восстановления при сбое или при ошибке пользователя.

Все наиболее важные документы, директории, файловые системы должны регулярно копироваться на внешнюю память (желательно делать не одну копию, а несколько на разные носители). Это должно стать непреложным правилом для каждого пользователя. Возможности ОС позволяют выполнять такое копирование

автоматически, в определенное время, например, ночью, когда в офисе никого нет, но компьютеры локальной сети работают.

ФАЙЛОВАЯ СИСТЕМА

Файловая система — это часть операционной системы, которая обеспечивает пользователю удобный интерфейс при работе с данными, хранящимися на диске, и совместное использование файлов несколькими пользователями и процессами.

В широком смысле понятие "файловая система" включает:

- совокупность всех файлов на диске;
- наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске;
- комплекс системных программных средств, реализующих управление файлами, в частности: создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами.

ФАЙЛЫ УСТРОЙСТВ

Файлы устройств позволяют программам взаимодействовать с аппаратными средствами и периферийными устройствами системы.

При конфигурировании ядра к нему добавляются те модули, которые знают, как взаимодействовать с каждым из устройств системы. За всю работу по управлению конкретным устройством отвечает специальная программа, называемая драйвером устройства.

Жесткие ссылки — это скорее не тип файла, а его дополнительное имя. Оно создается добавлением ссылки, у каждого файла она как минимум одна — и обычно это имя, под которым файл был создан.

Ссылку невозможно отличить от имени файла, к которому она присоединена: в ОС Linux они идентичны. Linux подсчитывает количество ссылок, указывающих на каждый файл, и не освобождает блоки данных файла до тех пор, пока не удалит его последнюю ссылку.

Символические ссылки обеспечивают возможность указывать вместо путевого имени файла имя ссылки. Символическая ссылка содержит путевое имя файла, на который она ссылается.

Имена файлов могут состоять из любых символов, за исключением слэша и символа с кодом ноль. Максимальная длина имени файла определяется конкретной системой.

Для каждого файла определен владелец или группа владельцев и права доступа к нему.
Есть три типа прав доступа:

- чтение;
- запись;
- выполнение/поиск.

Изменить права доступа к файлу может только владелец и суперпользователь (root).

Для того чтобы создать символическую ссылку, используется команда ln с дополнительной опцией -s:

```
# ln -s имя_файла_или_каталога имя_символической_ссылки
```

Аналогично создается жесткая ссылка, используется команда ln без дополнительных опций:

```
# ln -s имя_файла_или_каталога имя_жесткой_ссылки
```

Файловая система связывает носитель информации, с одной стороны, и API для доступа к файлам, с другой. Когда прикладная программа обращается к файлу, она не имеет никакого представления о том, каким образом расположена информация в конкретном файле. Все, что знает программа — это имя файла, его размер и атрибуты. Эти данные она получает от драйвера файловой системы. Именно файловая система устанавливает, где и как будет записан файл на физическом носителе (например, жестком диске).

С точки зрения операционной системы, весь диск представляет из себя набор кластеров размером от 512 байт. Драйверы файловой системы организуют кластеры в файлы и каталоги, реально являющиеся файлами, содержащими список файлов в этом каталоге. Эти же драйверы отслеживают, какие из кластеров в настоящее время используются, какие свободны, какие помечены как неисправные.

Однако файловая система необязательно напрямую связана с физическим носителем информации. Существуют виртуальные и сетевые файловые системы, которые являются лишь способом доступа к файлам, находящимся на удалённом компьютере. Рассмотрим самые популярные файловые системы, использующиеся в Astra Linux.

ФАЙЛОВАЯ СИСТЕМА EXT2

Ext2, или вторая расширенная файловая система — файловая система для ядра Linux.

Она достаточно быстра для того, чтобы служить эталоном в тестах производительности файловых систем, но она не является журналируемой файловой системой — и это ее главный недостаток. В ext2 были сразу реализованы соответствующие стандарту POSIX списки контроля доступа ACL и расширенные атрибуты файлов. Развитием ext2 стала журналируемая файловая система ext3, полностью совместимая с ext2.

Атрибуты файлов ext2 хранятся не в каталогах, как это сделано в ряде простых файловых систем, а в специальных таблицах. В результате каталог имеет очень простую структуру, состоящую всего из двух частей: номера индексного дескриптора и имени файла. Система адресации этой файловой системы позволяет при максимальном размере блока 4 Кб иметь файлы размера до 2 терабайт.

ФАЙЛОВАЯ СИСТЕМА EXT3

Ext3, или третья расширенная файловая система — журналируемая файловая система, используемая в операционных системах на ядре Linux, является файловой системой по умолчанию во многих дистрибутивах. Основана на ФС ext2.

Основное отличие от ext2 состоит в том, что ext3 журналируема, то есть в ней предусмотрена запись некоторых данных, позволяющих восстановить файловую систему при сбоях в работе компьютера.

Стандартом предусмотрено три режима журналирования:

- writeback: в журнал записываются только метаданные файловой системы, то есть информация о ее изменении. Не может гарантировать целостности данных, но уже заметно сокращает время проверки по сравнению с ext2;
- ordered: то же, что и writeback, но запись данных в файл производится гарантированно до записи информации об изменении этого файла. Немного снижает производительность и не может гарантировать целостности данных, хотя и увеличивает вероятность их сохранности при дописывании в конец существующего файла;
- journal: полное журналирование как метаданных ФС, так и пользовательских данных. Самый медленный, но и самый безопасный режим; может гарантировать целостность данных при хранении журнала на отдельном разделе (а лучше — на отдельном жестком диске).

Файловая система ext3 может поддерживать файлы размером до 1 ТБ. С Linux-ядром 2.4 объем файловой системы ограничен максимальным размер блочного устройства, что составляет 2 терабайта. В Linux 2.6 для 32-разрядных процессоров максимальный размер блочных устройств составляет 16 ТБ, однако ext3 поддерживает только до 4 ТБ.

ФАЙЛОВАЯ СИСТЕМА EXT4

Ext4 — это 64-битная версия ext3, способная поддерживать больший размер файловой системы (1 эксбибайт).

Среди её возможностей — непрерывные области дискового пространства, задержка выделения пространства, онлайн-дефрагментация и прочие. Обеспечивается прямая совместимость с системой ext3 и ограниченная обратная совместимость при недоступной способности к непрерывным областям дискового пространства.

ФАЙЛОВАЯ СИСТЕМА NFS

Network File System (NFS) — протокол сетевого доступа к файловым системам. Основан на протоколе вызова удаленных процедур (ONC RPC, Open Network Computing Remote Procedure Call) и позволяет подключать (монтировать) удаленные файловые системы через сеть.

NFS абстрагирована от типов файловых систем как сервера, так и клиента, существует множество реализаций NFS-серверов и клиентов для различных операционных систем и аппаратных архитектур. В настоящее время используется наиболее зрелая версия NFS v.4, поддерживающая различные средства аутентификации (в частности, Kerberos и LIPKEY с использованием протокола RPCSEC_GSS) и списки контроля доступа (как POSIX, так и Windows-типов).

ОСНОВНЫЕ КАТАЛОГИ ОС LINUX

Имя	**Описание 1**
/	Корневой каталог
/bin	Наиболее важные команды и программы

/boot	Все, что необходимо для загрузки операционной системы, ядро Linux, загрузчик
/dev	Файлы устройств
/etc	Системные конфигурационные файлы
/home	Домашние каталоги пользователей
/lib	Общие библиотеки, модули ядра
/mnt	Каталог для монтирования локальных и удаленных файловых систем
/opt	Дополнительные программные пакеты
/proc	Информация, касающаяся ядра; управление процессами
/root	Домашний каталог пользователя root
/sbin	Системные команды
/tmp	Временные файлы
/usr	Иерархия вторичных программных файлов
/var	Переменные данные (например, регистрационные журналы); файлы спула

Основные команды для работы в системе:

- **Is** — список файлов и каталогов;
- **Is -al** — форматированный список со скрытыми каталогами и файлами;

- **cd dir** — сменить директорию на dir;
- **cd** — сменить на домашний каталог;
- **pwd** — показать текущий каталог;
- **mkdir dir** — создать каталог dir;
- **touch file** — создать file;
- **rm file** — удалить file;
- **rm -r dir** — удалить каталог dir;
- **cp file1 file2** — скопировать file1 в file2
- **cp -r dir1 dir2** — скопировать dir1 в dir2. Создаст каталог dir2, если он не существует;
- **mv file1 file2** — переименовать или переместить file1 в file2, если file2 — существующий каталог, то переместить file1 в каталог file2;
- **cat file** — вывести содержимое file;
- **more file** — вывести содержимое file постранично;
- **head file** — вывести первые 10 строк file;
- **tail file** — вывести последние 10 строк file;
- **date** — вывести текущую дату и время cal – вывести календарь на текущий месяц
- **uname -a**— показать информацию о ядре;
- **clear** — очистить экран;
- **wc file** — показывает количество строк, слов, символов в файле file.

Модуль 6. Юнит 2. Тест для самоконтроля

1. Какая команда позволяет создать файл file?
    - wc file
    - rm file
    - mkfile file
    - **touch file**

2. В каком каталоге хранятся файлы устройств?
    - /var
    - **/dev**
    - /mnt
    - /bin

3. Какая файловая система не является журналируемой?
    - ext4
    - **ext2**
    - ntfs
    - ext3

Модуль 6. Юнит 3. Контрольное задание

Задание на перетаскивание объектов. (В данной таблице они расположены в нужном соответствии, в задании их нужно перепутать)

Задание: Сопоставьте термин с его определением:

Термины	***Определения***
Файловая система	Это часть операционной системы, которая обеспечивает пользователю удобный интерфейс при работе с данными, хранящимися на диске, и совместное использование файлов несколькими пользователями и процессами.
Резервное копирование	это копирование (backup) файлов на устойчивые носители (флэш-память, компакт-диск и др.) с целью их последующего восстановления при сбое или при ошибке пользователя.
Директория	это каталог-справочник ссылок на группу файлов или других директорий, каждые из которых имеет в данной директории свое уникальное символьное имя.
Открытие файла	это считывание в основную память заголовка файла и, возможно, одного или нескольких соседних блоков.
Закрытие файла	Это сброс всех копий блоков файла на внешнюю память и освобождение областей основной памяти, занятых открытым файлом.
Файлы устройств	Это файлы, позволяющие программам взаимодействовать с аппаратными средствами и периферийными устройствами системы.
Жесткие ссылки	это скорее не тип файла, а его дополнительное имя.
Символические ссылки	Это «ярлыки», обеспечивающие возможность указывать вместо путевого имени файла имя ссылки.

Модуль 6. Юнит 4. Памятка Консольные текстовые редакторы

Горячие клавиши управления nano:

Ctrl+G	Вызов меню полной подсказки
Ctrl+X	Выход из программы
Ctrl+O	Запись текущего файла
Ctrl+W	Поиск текста в текущем файле
Alt+R	Замена текста в текущем файле
Ctrl+K	Удаление строки в позиции курсора с сохранение ее в буфере(вырезать)
Ctrl+U	Вставка содержимого буфера в строку в позиции курсора
Ctrl+P	Перемещение курсора на одну строку вверх
Ctrl+N	Перемещение курсора на одну строку вниз
Ctrl+F	Перемещение курсора на один символ вперед
Ctrl+B	Перемещение курсора на один символ назад
Ctrl+A	Перемещение курсора в начало текущей строки
Ctrl+E	Перемещение курсора в конец текущей строки
Alt+A	Выделение и помещение в буфер текста начиная с текущей позиции курсора
Ctrl+D	Удаление символа в позиции курсора
Ctrl+H	Удаление символа слева от курсора
Ctrl+I	Вставка символа табуляции
Ctrl+M	Вставка символа перевода строки в позиции курсора
Alt+G	Переход на указанный номер строки

Горячие клавиши управления vim:

Переключение между режимами:

i	Режим вставки
/	Режим поиска
v	Визуальный режим

Работа с файлами

:w	Запись файла
:w<Имя файла>	Записать файл и задать ему имя

:q!	Выйти из редактирования без сохранения изменений
:wq	Выйти и сохранить
:x	Выйти и сохранить короткий вариант

Работа с текстом

w	Перемещает на одно слово вперед
b	Перемещает на одно слово назад
^	Перемещает в начало строки
$	Перемещает в конец строки
d	Удаление(вырезать)
d + w	Удаление слова(вырезать)
d d	Удаление все строки целиком
x	Удалить лишнюю букву
y	Копирование
u	Отменить изменения
p	Вставка вырезанного текста
.	Повторить команду
/	Поиск

Модуль 7. Подготовка к программированию на Python в Linux

Содержание модуля:

Данный модуль посвящен подготовке слушателей к программированию на Python в ОС Linux. Рассматривается установка пакетов для работы с python в ОС Linux, рроверка версии интерпретатора, обновление версии. Также слушатели познакомятся с системой управления пакетами pip - установкой, обновлением, удалением пакетов для python с помощью pip. Рассмотрено начало работы с интерпретатором python, выполнение первых команд в консоли, импорт и использование функций установленных пакетов. Слушатели изучают понятие виртуальной среды, установку пакетов для работы с виртуальными средами python в ОС Linux, расположение файлов виртуальной среды, интерпретатора, отличия работы с системным и виртуальным интерпретаторами. В рамках модуля слушатели научатся создавать, активировать, декативировать виртуальные среды python. Рассмотрены установка, обновление, импорт, удаление пакетов в виртуальной среде.

ПРОГРАММИРОВАНИЕ НА PYTHON В LINUX

Python – один из самых простых и «понятных» языков программирования. Он имеет большую базу качественной документации и подойдет как для больших проектов с графикой, так и для небольших терминальных приложений. Python – интерпретируемый язык программирования, а это значит, что прежде, чем начать выполнять программы на нем, необходимо установить в систему интерпретатор (такой посредник, который переводит исходный код на языке python в машинный код). Так, чтобы проверить версию установленного в вашей системе python интерпретатора, необходимо ввести команду

python --version

либо, если вы уверены, что версия начинается с 3,

python3 --version

Если интерпретатора в системе нет, его нужно установить командой

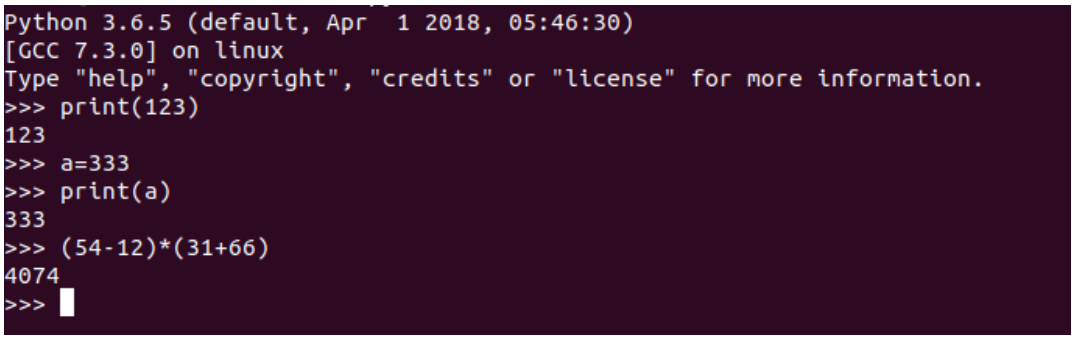
apt install python3

После чего, если все успешно установилось, можно попробовать вызвать работу интерпретатора командой

python3

При успешном запуске интерпретатора будет видна некоторая информация (например, версия python, дата и т.д.) и далее приветственная строка в виде трех символов больше «>>>». Данные символы являются приглашением для ввода команд. Если вы введете какую-то команду на python и нажмете Enter, она будет немедленно выполнена с выводом (если таковой есть) в терминал.

Пример:



Здесь мы выводим строку командой print, далее присваиваем переменной значение и выводим его на экран, затем выполняем арифметическую операцию. Кстати про переменные – при таком запуске интерпретатора они создаются только

для данной сессии. Если сейчас мы прекратим работу, потом запустим интерпретатор снова и попытаемся обратиться к переменной «a» - система не будет знать, что это за переменная:

```
Python 3.6.5 (default, Apr  1 2018, 05:46:30)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
>>>
```

Кстати, чтобы выйти из такого режима работы интерпретатора нужно ввести команду exit(), либо можно остановить процесс сочетанием ctrl+z.

Иногда бывает полезно, чтобы переменные все же запоминались не на одну сессию, а на некоторый период работы. Допустим, мы хотим создать программу, которую будем запускать в определенные моменты, и которая будет содержать какие-то переменные и оперировать с ними. Для этого нам понадобится создать скрипт на языке python.

Скрипты на python так же, как и на bash, являются последовательностью команд на языке программирования python, которые при выполнении отправляются на соответствующий интерпретатор – python. Записываются команды построчно, с форматированием, характерным для python – с соблюдением табуляции. Интерпретатор, как и в скриптах на bash, может указываться либо в начале файла скрипта после символов «#!», либо непосредственно при вызове файла на выполнение. И также необходимо сделать файл исполняемым.

Системный интерпретатор по умолчанию установится в /usr/bin, поэтому строка в начале скрипта должна выглядеть так (если используется системный интерпретатор):

#!/usr/bin/python3

По аналогии со скриптами на bash файл скрипта делается исполняемым (для всех пользователей) командой

chmod a+x <имя скрипта>

И запускается так:

Из той директории, где находится файл скрипта		*Из другой директории, где нет файла скрипта*	
С указанием интерпретатора в начале скрипта	*Без указания интерпретатора в начале скрипта*	*С указанием интерпретатора в начале скрипта*	*Без указания интерпретатора в начале скрипта*

# ./<имя скрипта>	# /usr/bin/python3 <имя скрипта>	# <полный путь к файлу скрипта>	# /usr/bin/python3 <полный путь к файлу скрипта>

Итак, приведем пример создания простейшего скрипта, выполняющего арифметическую операцию и затем выводящего строку «Hello».

Для начала создадим файл скрипта командой touch. Скрипт будет называться script.py. Далее отредактируем тело скрипта, например, с помощью редактора nano. Сделаем наш файл исполняемым и попробуем его выполнить. Последовательность команд с результатом:

```
viola@viola-VirtualBox:~$ touch script.py
viola@viola-VirtualBox:~$ nano script.py
viola@viola-VirtualBox:~$ sudo chmod a+x script.py
viola@viola-VirtualBox:~$ ./script.py
7
Hello
viola@viola-VirtualBox:~$
```

Содержимое скрипта в редакторе nano:

Данный скрипт был успешно выполнен.

Зачастую в процессе программирования мы сталкиваемся с установкой и импортом пакетов python – дополнительных библиотек, функции из которых мы могли бы использовать в своем коде. Но прежде, чем импортировать функцию или пакет в свой скрипт, необходимо установить пакет в систему. Для этого обычно

используют pip – специальный пакетный менеджер для языка python. Чтобы установить pip в свою систему нужно ввести команду

apt install python3-pip

После установки проверить версию можно командой

pip3 --version

Структура использования команды pip выглядит так:

pip3 команда опции пакет(ы)

Команда здесь – действие, которое необходимо выполнить. Это могут быть:

show	*показать информацию о пакете*
search	*найти пакет*
install	*установить пакет*
uninstall	*удалить пакет*
download	*скачать пакет и зависимости (без установки)*
list	*вывести список установленных пакетов*

Опции могут быть, например:

--upgrade – обновить пакет

--index-url URL – выполнить установку пакета, используя репозиторий по дресу URL

И т.д. Полный список опций можно просмотреть при вызове справки по команде: pip3 --help.

Пакеты же – это названия нужных нам пакетов python для установки и дальнейшего использования.

Например, если мы хотим установить пакет simplejson, введем команду

pip3 install simplejson

Пакет будет успешно установлен.

Иногда для комфортной работы бывает удобно создать виртуальную среду python – независимую среду, с собственным интерпретатором python, собственными импортами и автономной работой, не затрагивающей основной системный интерпретатор. Вышеперечисленными качествами и обусловлены плюсы использования виртуальных сред.

Разберемся, как можно создать такую среду в Linux. Для начала установим пакет python3-venv командой

apt install python3-venv

Далее, если установка прошла успешно, нужно создать саму виртуальную среду. Это можно сделать либо в готовой директории, созданной заранее, либо одной командой и создать папку и разместить в ней среду, это команда:

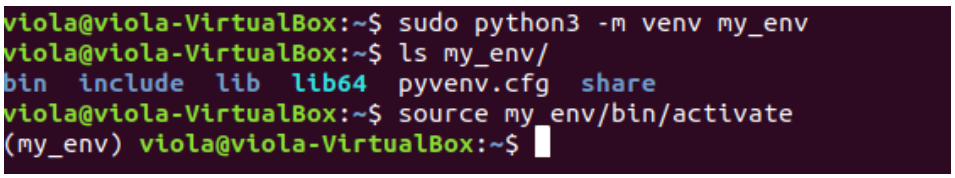
python3 –m venv <название папки среды>

Данная команда создаст необходимый каталог и поместит в него отдельный интерпретатор python, pip, нужные скрипты и библиотеки. Если после создания среды просмотреть содержимое папки, там окажутся папки и файлы:

bin include lib lib64 pyvenv.cfg share

Нам будет интересен каталог bin – в нем находится скрипт активации виртуальной среды. Так, чтобы активировать созданную виртуальную среду, нужно выполнить команду:
source < название папки среды>/bin/activate

После чего изменится приветственная строка – в начале появится название виртуальной среды в круглых скобках, означающее, что среда была успешно активирована. Если вы видите такой результат:



Можете смело начинать работу с своей виртуальной среде. Например, запустить интерпретатор python, выполнить какие-либо команды или установить пакеты. Все это будет происходить только в рамках данной среды и никак не коснется основной системы.

```
viola@viola-VirtualBox:~$ sudo python3 -m venv my_env
viola@viola-VirtualBox:~$ ls my_env/
bin  include  lib  lib64  pyvenv.cfg  share
viola@viola-VirtualBox:~$ source my_env/bin/activate
(my_env) viola@viola-VirtualBox:~$ python3
Python 3.6.9 (default, Jun 29 2022, 11:45:57)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print(5*3-12)
3
>>> exit()
(my_env) viola@viola-VirtualBox:~$
```

В виртуальной среде также можно создавать и запускать скрипты, притом технология их создания внутри среды ничем не отличается от приведенной выше (для основной системы). Для выхода из виртуальной среды можно использовать команду exit() либо сочетание клавиш ctrl+D. Для удаления виртуальной среды служит команда deactivate.

```
(my_env) viola@viola-VirtualBox:~$ deactivate
viola@viola-VirtualBox:~$
```

Правда каталог, в котором была создана среда, она не удалит – для этого придется использовать rm.

Модуль 7. Юнит 2. Тест для самоконтроля

1. *Активировать виртуальную среду python с названием virtual можно командой:*

    - *python3 –m venv virtual*
    - *pip3 --upgrade virtual*
    - ***source virtual/bin/activate***
    - *deactivate virtual*

2. *Основной системный интерпретатор python находится по умолчанию по пути*
    - **/usr/bin/python3**
    - /home/username/python/python3
    - /home/username/python3
    - /bin/python

3. *Как выглядит приветственная строка интерпретатора python?*
    - #!
    - **>>>**
    - <<<
    - #:

Модуль 7. Юнит 3. Контрольное задание

Задание на перетаскивание объектов (В данной таблице они в правильном порядке, их необходимо «перемешать»).

Задание 1: **Сопоставьте команды по управлению виртуальной средой python с их объяснением**

# source < название папки среды>/bin/activate	*Активировать созданную виртуальную среду*
# python3 –m venv <название папки среды>	*Создать виртуальную среду*
# chmod a+x <имя скрипта>	*Назначить права на исполнение скрипта для всех*