[1. Понятие операционной системы, выполняемые ей функции.](#_4ap8b0zg4hdg) **3**

[2. Порядок загрузки компьютера и операционной системы.](#_jj99ybnafix8) **3**

[3. История развития ОС.](#_3or8r4t6jy9k) **4**

[4. Семейство операционных систем UNIX. Философия UNIX.](#_y17ewh1nxiuw) **7**

[5. Сравнительная характеристика Linux и Windows.](#_qe20fttwhmdi) **10**

[6. Понятие системного вызова операционной системы.](#_ef90f25mn7bg) **11**

[7. Понятие виртуализации. Виртуальные машины.](#_z6bnlunt9bku) **13**

[8. Основные компоненты операционной системы. Разделение функций.](#_c85gyu7t2iyh) **13**

[9. Понятие файловой системы. Функции, виды, характеристики.](#_bsv1rru1su3d) **13**

[10. Физические и логические диски. Разбиение жесткого диска для установки Linux.](#_3rbihprk0jva) **14**

[11. Стандартная иерархия каталогов Linux.](#_52lfa6e90uh) **16**

[12. Понятие ядра операционной системы. Пространство ядра и пользовательское пространство.](#_ge6mfl45mb6z) **18**

[13. Командный интерпретатор операционной системы.](#_jo7g74vrbzzc) **19**

[14. Управление оперативной памятью в современных ОС. Виртуальная память.](#_df0z34eqitjn) **20**

[15. Работа ОС с внешними устройствами. Понятие драйвера устройства.](#_kr87gkie7qk1) **21**

[16. Многозадачность современных операционных систем.](#_bn689ur13jqx) **22**

[17. Графический и текстовый интерфейс взаимодействия с ОС.](#_9iqhwb4co1nm) **24**

[18. Источники установки программных приложений. Программные репозитории.](#_10tes6wnbn9q) **24**

[19. Способы установки программ в Linux.](#_wsmnsgocnq9w) **26**

[20. Процессы в Linux. Функции, организация, управление.](#_58sf4kpw66hd) **28**

[21. Жизненный цикл процесса операционной системы.](#_qwsqtfdox3tp) **30**

[22. Основные команды Linux для управления файлами.](#_mcd6m945kxjs) **31**

[23. Основные команды Linux для управления каталогами.](#_21fu642y15w2) **33**

[24. Основные команды Linux для управления файловыми системами.](#_g287v0aesq7u) **34**

[25. Межпроцессное взаимодействие. Проблема и способы решения.](#_hmidvdt7995f) **34**

[26. Понятие пакета в Linux. Менеджеры пакетов.](#_kbvle4onbkl) **36**

[27. Основные команды Linux для управления пакетами.](#_8x34egfkdlxf) **37**

[28. Основные команды Linux для управления процессами.](#_x8jr7q3odcly) **37**

[29. Типы файлов в Linux. Ссылки.](#_5orsc3zl28t) **38**

[30. Основные команды для управления текстовыми потоками.](#_qvwkpqqmdc6t) **39**

[31. Основные команды для просмотра и редактирования текстовых файлов.](#_s8dzdfbhhg64) **40**

[32. Регулярные выражения.](#_r35zfse6uvak) **40**

[33. Встроенные и внешние команды bash.](#_nwkssn8g0ia2) **42**

[34. Основные конфигурационные файлы Linux.](#_3e37fw9uzkpp) **43**

[35. Командный интерпретатор bash. Структура команды.](#_z33lo0ozwri5) **47**

[36. Bash. Переменные и типы.](#_u0zfch9tavfh) **48**

[37. Bash. Условия.](#_sw5d50gyc8e3) **50**

[38. Bash. Циклы.](#_fmnzcom6jszw) **51**

[39. Bash. Функции.](#_bl6o8xmr7xkk) **58**

[40. Переменные окружения в bash.](#_es674qbphimx) **62**

[41. Основные команды Linux для управления пользователями и группами.](#_1w64bn5rr9s7) **64**

[42. Суперпользователь root. Характеристика, особенности, функции, опасность.](#_g5c5g58e84t8) **64**

[43. Система прав доступа Linux. Структура и смысл прав доступа.](#_fv4mcxgdiuld) **65**

[44. Организация хранения паролей пользователей в Linux.](#_mpa5h1iz7qtu) **65**

[45. Основные команды Linux для управления правами доступа.](#_ljkldhrkvde) **66**

[46. Понятие компьютерных сетей. Общие принципы, организация.](#_86ua16uv1rg2) **67**

[47. Понятие сетевого ресурса. Классификация.](#_ps35x07t7pim) **69**

[48. Семейство сетевых протоколов TCP/IP.](#_dk2geu6wn8rc) **69**

[49. Адресация компьютеров в сети. IP, MAC адреса.](#_ih75fw5w9yn5) **73**

[50. Удаленный доступ к командной строке. Протокол SSH.](#_5hqwln1qne4u) **74**

### 

### 

### 1. Понятие операционной системы, выполняемые ей функции.

**Операционная система** - комплекс взаимосвязанных программ, который действует как интерфейс между приложениями и пользователями с одной стороны и аппаратурой компьютера с другой стороны.

**Функции ОС:**

**1)ОС как расширенная машина:**

* скрывает детали реализации
* предоставляет программам виртуальную машину

**2)ОС как менеджер ресурсов:**

* распределение ресурсов по времени
* распределение ресурсов по пространству

### 2. Порядок загрузки компьютера и операционной системы.

Порядок:

* Запуск системы, POST
* Загрузчик 1 уровня
* Загрузчик 2 уровня
* Загрузчик 3 уровня
* Загрузка ядра ОС

Первое устройство, которое запускается после кнопки включения компьютера, – блок питания. Если все питающие напряжения оказываются в норме – подается сигнал Power Good, свидетельствующий об успешном тестировании блока питания и разрешающий запуск компонентов системной платы. Далее чипсет формирует сигнал сброса центрального процессора, по которому очищаются регистры процессора, и он запускается.

Этапы процедуры POST:

1. Первоначальная инициализация основных системных компонентов
2. Детектирование оперативной памяти, копирование код BIOS в оперативную память
3. Первоначальная настройка чипсета
4. Поиск и инициализация видеоадаптера
5. Проверка контрольной суммы СMOS (Энергонезависимая память BIOS) и состояния батарейки
6. Тестирование процессора и оперативной памяти
7. Подключение клавиатуры, тестирование портов ввода-вывода и других устройств
8. Инициализация дисковых накопителей. Сведения об обнаруженных устройствах выводятся на экран
9. Поиск и инициализация устройств, имеющих собственную BIOS
10. Вызов программного прерывания BIOS INT 19h, который ищет загрузочный сектор на устройствах, указанных в списке загрузки

**Загрузчик 1 уровня:**

Master Boot Record- главная загрузочная запись. Содержит таблицу разделов и исполняемый код.

Главная задача – поиск активного системного раздела диска и передача управления его загрузочному сектору.

**Загрузчик 2 уровня:**

Передача управления исполняемому коду, записанному в загрузочном секторе Partition Boot Sector

**Загрузчик 3 уровня:**

Первоначально загрузчиком в Linux был LILO. Из-за неспособности понимать используемые файловые системы ему на смену пришел GRUB. Он считывает информацию из текстового файла Boot.ini, в котором записана информация об установленных ОС

**Загрузка ядра ОС:**

Загрузка ядра ОС и передача ему управления.

### 3. История развития ОС.

Первый настоящий цифровой компьютер был изобретен английским математиком Чарльзом Бэббиджем. Это была механическая машина без ОС

Первое поколение (1945-1955): электронные лампы

* Профессор Джон Атанасов и его аспирант Клиффорд Берри создали конструкцию, которая сейчас считается первым действующим цифровым компьютером. В ней использовалось 300 электронных ламп.
* Конрад Цузе в Берлине построил Z3 — компьютер, основанный на использовании электромеханических реле.
* В 1944 году группой ученых (включая Алана Тьюринга) в Блетшли Парк (Великобритания) был построен и запрограммирован «Колоссус», в Гарварде Говардом Айкеном построен «Марк I», а в университете штата Пеннсильвания Вильям Мочли и его аспирант Джон Преспер Эккерт построили «Эниак». Некоторые из этих машин были цифровыми, в других использовались электронные лампы, а работу части из них можно было программировать, но все они были весьма примитивно устроены и тратили многие секунды на производство даже простейших вычислений.

Все программирование велось исключительно на машинном языке

О языках программирования (даже об ассемблере) тогда еще ничего не было известно.

Об операционных системах вообще никто ничего не слышал.

Второе поколение (1955-1965): транзисторы и системы пакетной обработки

* Первые мейнфреймы
* Задание(программа) писалось на бумаге и переносилось на перфокарту
* Система пакетной обработки: сбор всех заданий и перенос их на магнитную ленту с помощью компьютера
* IBM 1401
* IBM 7094
* Компьютеры 2го поколения использовались для научных и технических вычислений
* Языки: Фортран и ассемблер
* Операционные системы: FMS, IBSYS

Третье поколение (1965-1980): интегральные схемы и многозадачность

* Фирма IBM попыталась решить эти проблемы разом, выпустив серию машин IBM System/360.
* В последующие годы корпорация IBM, используя более современные технологии, выпустила преемников 360-й серии, имеющих с ней обратную совместимость, эти серии известны под номерами 370, 4300, 3080 и 3090. Самыми последними потомками этого семейства стали машины zSeries.
* Семейство компьютеров IBM/360 стало первой основной серией, использующей малые интегральные схемы, дававшие преимущество в цене и качестве по сравнению с машинами второго поколения, собранными на отдельных транзисторах
* Операционная система OS/360
* Самым важным достижением явилась многозадачность.
* Желание сократить время ожидания ответа привело к разработке режима разделения времени — варианту многозадачности, при котором у каждого пользователя есть свой диалоговый терминал
* Первая универсальная система с режимом разделения времени CTSS была разработана на специально переделанном компьютере IBM 7094
* Универсальная общая компьютерная система MULTICS
* Несмотря на коммерческую неудачу, система MULTICS оказала существенное влияние на последующие операционные системы (особенно на UNIX и ее производные, на FreeBSD, Linux, IOS и Android)
* Мини-компьютеры: DEC машина PDP-1
* Кен Томпсон ,работавший над проектом MULTICS, чуть позже нашел мини-компьютер PDP-7, которым никто не пользовался, и решил написать упрощенную однопользовательскую версию системы MULTICS. Эта работа позже переросла в операционную систему UNIX:
* стандарт системы UNIX, названный POSIX
* клон системы UNIX, так называемую систему MINIX
* Желание получить свободно распространяемую версию MINIX (в отличие от образовательной) привело к тому, что финский студент Линус Торвальдс создал систему Linux

Четвертое поколение (с 1980 года по наши дни): персональные компьютеры

* Микрокомпьютеры
* Intel 8080 — первый универсальный 8-разрядный центральный процессор
* дисковую операционную систему, названную CP/M
* Операционная система DOS
* Видоизмененная система была переименована в MS-DOS
* изобретение графического интерфейса пользователя
* Mac OS X является операционной системой, построенной на основе UNIX, хотя и с весьма своеобразным интерфейсом
* Windows NT (NT означает New Technology)
* Windows 2000
* Windows 98 и Windows NT 4.0.
* Windows Me
* Windows XP под названием Vista
* Windows 7
* Windows 8
* Операционная система FreeBSD
* Пользовательский интерфейс, например Gnome или KDE

Пятое поколение (с 1990 года по наши дни): мобильные компьютеры

* Первый настоящий мобильный телефон появился в 1946 году, и тогда он весил около 40 кг
* Первый по-настоящему переносной телефон появился в 1970-х годах и при весе приблизительно 1 кг был воспринят весьма позитивно. Его ласково называли «кирпич».
* Первый настоящий смартфон появился только в середине 1990-х годов, когда Nokia выпустила свой N9000, представлявший собой комбинацию из двух отдельных устройств: телефона и КПК
* В 1997 году в компании Ericsson для ее изделия GS88 «Penelope» был придуман термин «смартфон»
* В первое десятилетие после своего появления большинство смартфонов работало под управлением Symbian OS. Эту операционную систему выбрали такие популярные бренды, как Samsung, Sony Ericsson, Motorola и Nokia.
* Операционные системы, например RIM Blackberry OS
* Операционная система Android, выпущенная компанией Google в 2008 году
* ОС писались преимущественно на языке программирования Java

### 4. Семейство операционных систем UNIX. Философия UNIX.

Семейство операционных систем UNIX

Проект системы UNIX (Uniplex Information and Computing Services) был создан сотрудниками лаборатории Bell Labs фирмы AT&T К. Томпсоном и Д. Ритчи более 20 лет назад. Разработанная ими ОС была реализована на ассемблере. Изначально сотрудник Bell Labs Б. Керниган назвал эту систему «UNICS». Однако скоро ее стали называть сокращенно «UNIX». В 1973 г. Д. Ритчи разработал язык программирования С (Си), и скоро ОС UNIX заново переписали на этом языке. После публикации Д. Ритчи и К. Томпсона в 1974 г. в журнале САСМ систему UNIX стали применять повсеместно. Главной проблемой ОС семейства UNIX является несовместимость различных версий. Попытки стандартизации версий UNIX закончились неудачей, так как наибольшее распространение получили две несовместимые версии этой системы: линия фирмы АТ&Т – UNIX System V и линия университета Berkeley – UNIX BSD. Многие фирмы, основываясь на этих версиях, разработали свои варианты UNIX: SunOS и Solaris фирмы Sun Microsystems, AIX фирмы IBM, UnixWare фирмы Novell и др. Одна из последних версий UNIX System V Release 4 собрала в себе лучшие черты линий UNIX System V и UNIX BSD, но данная разновидность системы является незавершенной, так как в ней отсутствуют системные утилиты, необходимые для успешного использования ОС.

**Общими чертами для любой ОС UNIX считаются:**

1) Многопользовательский режим со способом защиты данных от несанкционированного доступа;

2) Реализация мультипрограммной обработки в режиме разделения времени, которая основана на применении алгоритмов вытесняющей многозадачности; повышение уровня мультипрограммирования; 3) Унификация операций ввода-вывода на базе расширенного использования понятия «файл»;

4) иерархическая файловая система, которая образует единое дерево каталогов независимо от числа физических устройств, используемых для размещения файлов;

5) переносимость системы, которая осуществляется за счет написания ее основной части на языке С;

6) разнообразные средства взаимодействия процессов, например через сеть;

7) кэширование диска с целью уменьшения среднего времени доступа к файлам.

**Философия UNIX.**

[Дуг Макилрой](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%BA%D0%B8%D0%BB%D1%80%D0%BE%D0%B9,_%D0%94%D1%83%D0%B3%D0%BB%D0%B0%D1%81), изобретатель [каналов Unix](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B2%D0%B5%D0%B9%D0%B5%D1%80_(Unix)) и один из основателей традиции Unix, обобщил философию следующим образом:

«Философия Unix гласит:

Пишите [программы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), которые делают что-то одно и делают это хорошо.

Пишите программы, которые бы работали вместе.

Пишите программы, которые бы поддерживали [текстовые потоки](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D1%8B%D0%B5_%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%B8), поскольку это универсальный интерфейс».

Обычно эти высказывания сводятся к одному «Делайте что-то одно, но делайте это хорошо».

Из этих трёх принципов только третий является специфичным для Unix, хотя разработчики Unix чаще других акцентируют внимание на всех трёх принципах.

В 1994 году [Майк Ганцарз](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B0%D0%B9%D0%BA_%D0%93%D0%B0%D0%BD%D1%86%D0%B0%D1%80%D0%B7&action=edit&redlink=1) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Mike Gancarz) объединил свой опыт работы в Unix (он является членом команды по разработке системы [X Window System](https://ru.wikipedia.org/wiki/X_Window_System)) с высказываниями из прений, в которых он участвовал со своими приятелями программистами и людьми из других областей деятельности, так или иначе зависящих от Unix, для создания Философии Unix, которая сводится к 9 основным принципам:

* Красиво — небольшое.
* Пусть каждая программа делает что-то одно, но хорошо.
* Стройте прототип программы как можно раньше.
* Предпочитайте переносимость эффективности.
* Храните данные в простых текстовых файлах.
* Извлекайте пользу из уже существующих программных решений.
* Используйте [скриптовые языки](https://ru.wikipedia.org/wiki/%D0%A1%D1%86%D0%B5%D0%BD%D0%B0%D1%80%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) для уменьшения трудозатрат и улучшения переносимости.
* Избегайте пользовательских интерфейсов, ограничивающих возможности пользователя по взаимодействию с системой.
* Делайте каждую программу «фильтром».

Менее важные 10 принципов не снискали всеобщего признания в качестве частей философии Unix, и в некоторых случаях являлись предметом горячих споров ([монолитное ядро](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BD%D0%BE%D0%BB%D0%B8%D1%82%D0%BD%D0%BE%D0%B5_%D1%8F%D0%B4%D1%80%D0%BE) против [микроядра](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D1%8F%D0%B4%D1%80%D0%BE)):

Позвольте пользователю настраивать окружение.

Делайте ядра операционной системы маленькими и легковесными.

Используйте нижний регистр и придерживайтесь кратких названий.

Не храните тексты программ в виде распечаток («Спасите деревья!»).

Не сообщайте пользователю об очевидном («Молчание — золото»).

Разбивайте сложные задачи на несколько простых, выполняемых параллельно («Мыслите „параллельно“»).

Объединённые части целого есть нечто большее, чем просто их сумма.

Ищите [90-процентное решение](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D0%BE%D0%BD_%D0%9F%D0%B0%D1%80%D0%B5%D1%82%D0%BE).

Если можно не добавлять новую функциональность, не добавляйте её («[Чем хуже, тем лучше](https://ru.wikipedia.org/wiki/%D0%A7%D0%B5%D0%BC_%D1%85%D1%83%D0%B6%D0%B5,_%D1%82%D0%B5%D0%BC_%D0%BB%D1%83%D1%87%D1%88%D0%B5)»).

Мыслите иерархически.

### 5. Сравнительная характеристика Linux и Windows.

|  | **Windows** | **Linux** |
| --- | --- | --- |
| 1. *распространение, стоимость* | Закрытый код, предустановленный на компьютере вариант или коробочный вид,  Примерно 3000 руб. | Открытый код, свободное распространение,  Бесплатно |
| 1. *установка* | В течение 15 минут | В течение 15 минут |
| 1. *файловые системы* | NTFS и FAT32.  Не различает другие файловые системы | Более ста файловых систем  Распознает файловые системы Windows |
| 1. *подключение к Интернету* | Автоматическое по протоколу TCP IP | Автоматическое по протоколу TCP IP |
| 1. *работа в Интернете* | Для скачивания требует установку программ-клиентов FTP, SSH, Samba.  IE уязвим, проникновение многих вирусов | Все предустановленно, Выбор браузера не имеет значения. Лучший Mozila Firefox |
| 1. *пользовательское ПО* | Скудный набор программ после инсталляции: блокнот, Paint, калькулятор, Internet Explorer и проигрыватель (не проигрывает без установки кодеков) | Open Office, Gimp, Mediaплееры, программы для записи дисков и более 100 различных программ |
| 1. *установка ПО* | Ведется при помощи файла Setup.exe | Устанавливается при помощи пакетов |
| 1. *антивирусная защита* | Защищен частично, требуется дополнительная покупка и установка антивирусов  Имеется встроенная защита от вредоносного кода «Защита от вирусов», не надежная и взламывается даже от несерьезной вирусной атаке. | Изначально предусмотрены средства защиты в ядре операционной системы. |

**Вывод:** :Делайте сами

### 6. Понятие системного вызова операционной системы.

Системный вызов позволяет приложению обратиться к операционной системе с просьбой выполнить то или иное действие, оформленное как процедура (или набор процедур) кодового сегмента ОС. Реализация системных вызовов должна удовлетворять следующим требованиям:

* обеспечивать переключение в привилегированный режим;
* обладать высокой скоростью вызова процедур ОС;
* обеспечивать по-возможности единообразное обращение к системным вызовам для всех аппаратных платформ, на которых работает ОС;
* допускать расширение набора системных вызовов;
* обеспечивать контроль со стороны ОС за корректным использованием системных вызовов.

Требование по обеспечению переключения в привилегированный режим для большинства аппаратных платформ может быть выполнено только с помощью механизма программных прерываний.

В большинстве ОС системные вызовы обслуживаются по централизованной схеме, основанной на существовании диспетчера системных вызовов. Диспетчер системных вызовов обычно представляет собой простую программу, которая сохраняет содержимое регистров процессора в системном стеке (процессор переходит к работе в привилегированном режиме), проверяет, попадает ли запрошенный номер вызова в поддерживаемый ОС диапазон и передает управление процедуре ОС, адрес которой задан в таблице адресов системных вызовов. Процедура реализации системного вызова извлекает из системного стека аргументы и выполняет заданное действие. После завершения работы системного вызова управление возвращается диспетчеру. Диспетчер восстанавливает регистры процессора, помещает в определенный регистр код возврата и выполняет процедуру возврата из прерывания, которая восстанавливает непривилегированный режим работы процессора.

Для приложения системный вызов ничем не отличается от вызова обычной библиотечной функции языка С, связанной с объектным кодом приложения и выполняющейся в пользовательском режиме. Для ускорения выполнения некоторых простых системных вызовов, которым не требуется привилегированный режим, необходимая работа полностью выполняется с помощью библиотечной функции. При этом для стороннего наблюдателя системные вызовы и библиотечные функции выглядят одинаково. Так, программист при работе с функциями прикладного программного интерфейса API (например, Win32) имеет дело с библиотечными функциями, часть из которых используется для работы с системными вызовами, а часть – нет. Такой табличный способ организации системных вызовов принят практически во всех операционных системах. Он позволяет легко модифицировать состав системных вызовов, просто добавив в таблицу новый адрес и расширив диапазон допустимых номеров вызовов.

Операционная система может выполнять системные вызовы в *синхронном* или *асинхронном* режимах. Синхронный системный вызов означает, что процесс, сделавший такой вызов, приостанавливается до тех пор, пока системный вызов не выполнит всю требующуюся работу. После этого планировщик переводит процесс в состояние готовности и при очередном выполнении процесс гарантированно может воспользоваться результатами завершившегося к этому времени системного вызова. **Синхронные системные вызовы называются еще блокирующими, так как вызвавший системное действие процесс блокируется до его завершения.**

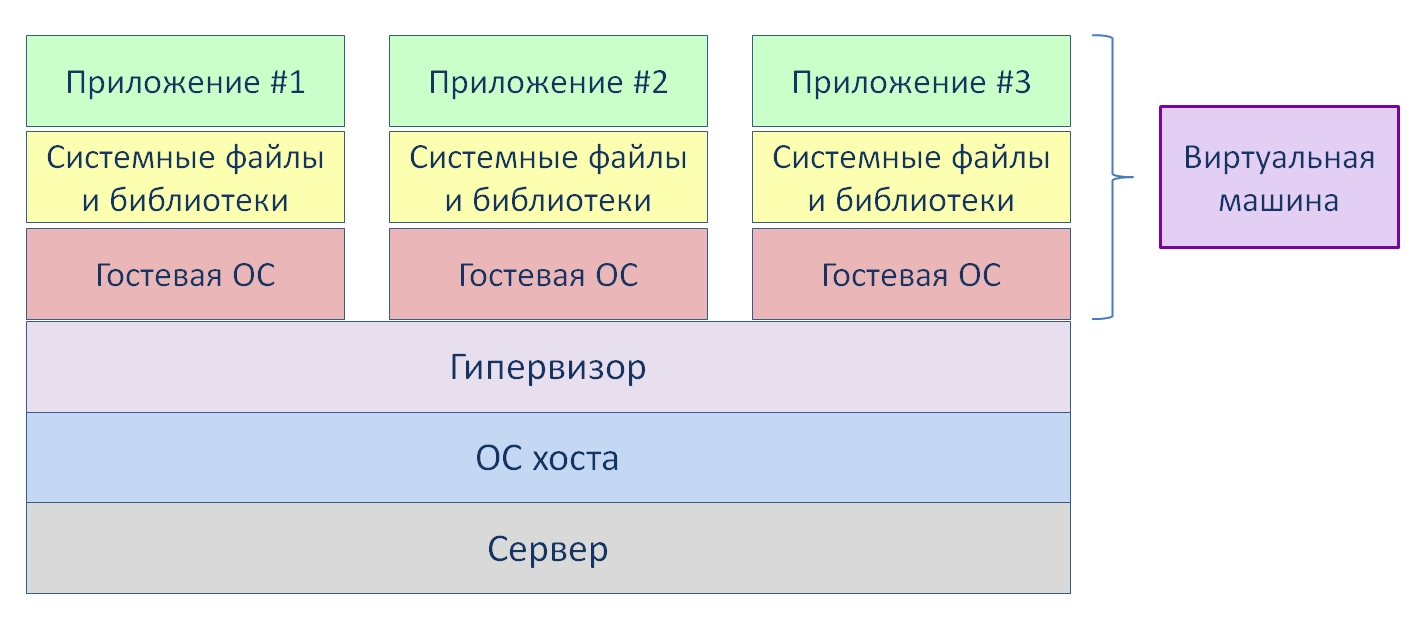
Асинхронный системный вызов не приводит к переводу процесса в режим ожидания, и после выполнения некоторых начальных системных действий управление возвращается прикладному процессу.

Большинство системных вызовов в операционных системах являются синхронными, так как этот режим избавляет приложение от работы по выяснению момента появления результата вызова. Вместе с тем в новых версиях операционных систем количество асинхронных системных вызовов постепенно увеличивается, что дает больше свободы разработчикам сложных приложений. Асинхронные системные вызовы широко используются в операционных системах на основе микроядерного подхода. При этом в пользовательском режиме работает часть ОС, и приложениям необходимо иметь полную свободу в организации своей работы, которую и обеспечивает асинхронный режим обслуживания системных вызовов микроядром.

### 7. Понятие виртуализации. Виртуальные машины.

**Виртуализация-** это сокрытие настоящей реализации какого-либо процесса или объекта от истинного представления для того, кто им пользуется. Продуктом виртуализации является нечто удобное для использования и не имеет значение как объект устроен в действительности. запуск одной или нескольких систем в рамках другой операционной системы на одной вычислительной машине.

**Виртуальные машины -** это программные абстракции, запускаемые на платформе реальных аппаратно-программных систем.



### 8. Основные компоненты операционной системы. Разделение функций.

К ним относятся *ядро системы*, обеспечивающая взаимодействие всех компонент, *загрузчик программ, подсистемы*, обеспечивающие диалог с человеком, *оконная система*, *интерпретатор команд*, а также *файловая система*. Именно системные компоненты определяют свойство ОС.

**Разделение функции.** Разделение функций было сделано для того чтобы бы избежать возможности изменения исполняемой программы или данной одной программы в памяти компьютера другой программы. Реализация разделений функции была одобрена разработчиками процессов, предложивших архитектуры с двумя режимами работы «реальными» и «защищенными»

### 9. Понятие файловой системы. Функции, виды, характеристики.

Файловая системы – порядок определяющий хранение и изменение на носителях данных в компьютере или другом электронном оборудовании. Она определяет формат содержимого, а также способ физического хранения информации.

**Функции файловой системы:**

**-** создание, чтение и удаление

- поиск файлов

- назначение и изменение атрибутов

- размещение и упорядочение

- определение структуры файла

- защита файлов при системном сбое

- определение максимального поддерживаемого объема данных на носителе информации

**Виды файловой системы:**

*NTFS*

*FAT32*

*EXT3*

*EXT4*

*UDF*

*NFS+*

*ExFAT*

***Характеристики файловой системы:***

часть операционной системы, которая связана непосредственно с размещением, удалением, перемещением электронной информации на определенном носителе, а также безопасностью ее дальнейшего использования в будущем. Именно этот ресурс также применим в случаях, когда требуется восстановление утерянной информации по причине программного сбоя, как такового. То есть это основной инструмент работы с электронными файлами.

### 10. Физические и логические диски. Разбиение жесткого диска для установки Linux.

Когда компьютер выключен, вся информация хранится на жестком диске. При включении компьютер загружается со своего жесткого диска.

Физические диски – это физическое реально существующее устройство, например:

* жесткий диск внутри компьютера (ноутбука),
* внешний жесткий диск, который подключается через порт USB,

Жесткий диск можно разбить на один, два или более **логических дисков**. Логический диск не существует физически, так как является частью дискового пространства внутри реального физического устройства.

Если на компьютере работает несколько пользователей, то удобно, чтобы у каждого из них был свой логический диск на жестком диске. Тогда каждый пользователь на таком компьютере хранит свою информацию на своем логическом диске.

Для установки Windows достаточно просто выбрать раздел диска. Система сама его отформатирует и установит туда все файлы. Но Linux позволяет намного гибче настроить расположение системы и ее файлов. Во время установки можно разместить различные папки с системными или пользовательскими файлами на разных разделах.

Эта функция повышает надежность системы, а также удобство ее использования.

У каждого из разделов своя задача. Разбивка диска для linux между несколькими разделами изолирует их один от другого. Если на одном разделе, например, домашнем закончится место, то система все еще сможет нормально работать, потому что это никак не касается корневого раздела.

В стандартной разбивке диска для Linux используется четыре раздела

* **/** - корень, основной раздел для файловой системы; ес
* Главный раздел системы. На нем будут находиться все системные файлы, и к нему будут подключаться другие разделы. Сюда же устанавливаются все программы. Требуется большое количество места (приблизительно 50 Гигабайт) и стабильная файловая система (ext4, reiserfs,  btrfs).
* **/boot** - файлы загрузчика;

Разбивка жесткого диска linux начинается с создания раздела /boot. На нем размещаются файлы настройки и модули загрузчика, которые считываются при старте Grub, а также ядро и initrd образ. Нужна самая быстрая и простая файловая система (ext2). Задача этого раздела сводится к тому, чтобы как можно быстрее выдать файлы во время загрузки. Места много не требуется (достаточно 300 Мегабайт).

* **/home** - раздел для файлов пользователя;

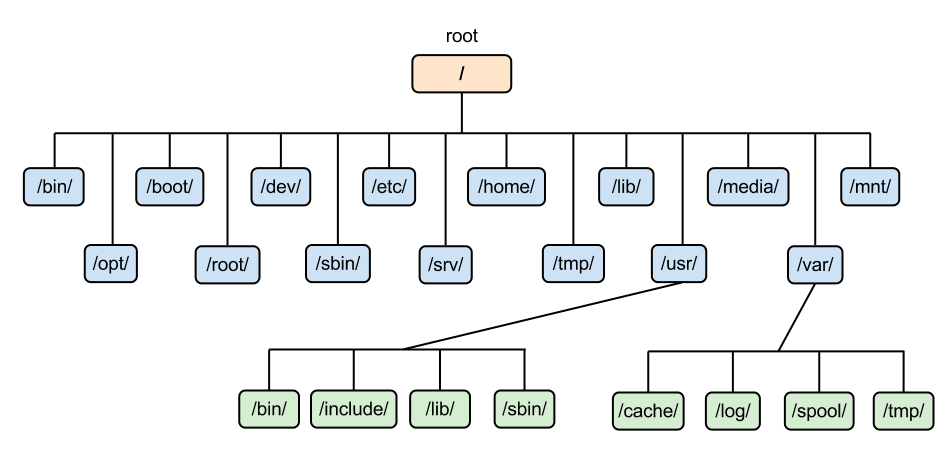
Раздел для файлов пользователя: загрузки, документы, видео, музыка, настройки программ. Потребуется все остальное место, которое не используется другими разделами. Нужна быстрая и стабильная файловая система (ext4, btrfs, xfs).

* **swap** - раздел подкачки, для выгрузки страниц из оперативной памяти, если она будет переполнена.

Это раздел подкачки, куда будут отправляться неиспользуемые страницы памяти, если она переполнена. Также сюда записывается все содержимое памяти когда компьютер уходит в режим сна или гибернации. Размер должен быть равным количеству оперативной памяти. Файловая система специальная – swap.

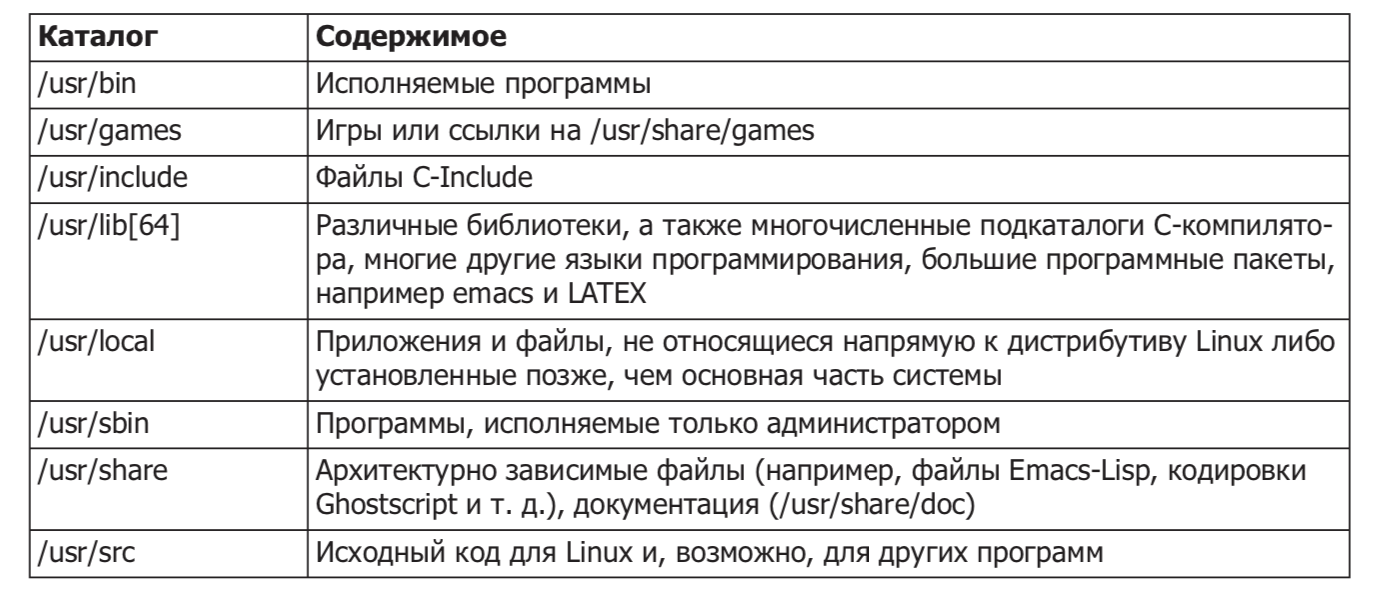
### 11. Стандартная иерархия каталогов Linux.

Иерархию каталогов Linux напоминает дерево. В Linux используется Стандарт иерархии файловой системы (Filesystem Hierarchy Standart, FHS) Дерево каталогов в Linux начинается с корневого каталога /. В нем, как правило, ничего нет, только каталоги: /bin, /boot, /dev, /etc, /home, /proc, /root, /sbin, /srv, /tmp, /usr, /var.



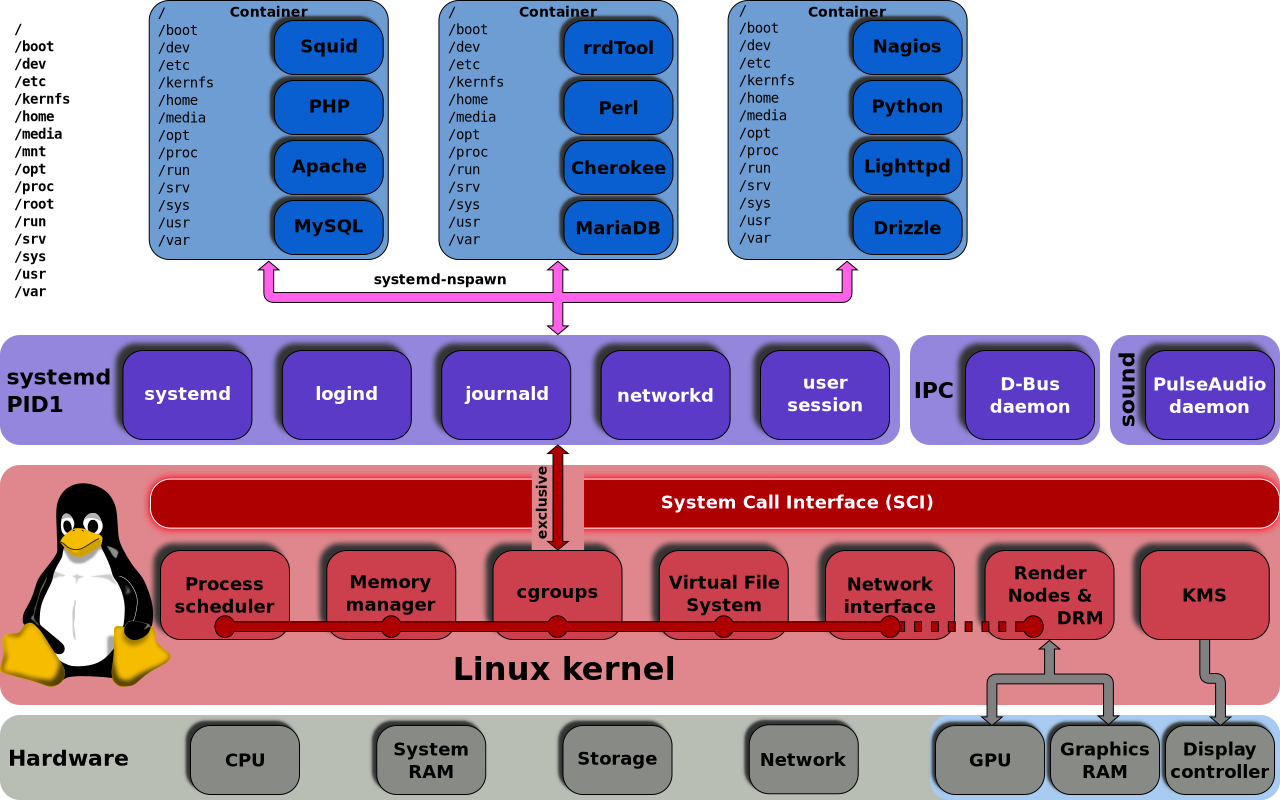
Теперь рассмотрим их.

* /bin – содержит элементарные команды Linux, предназначенные для управления системы. Эти команды могут выполнять все пользователи.
* /boot – в ней находятся файлы, необходимые для загрузки системы (Например: GRUB). В большинстве дистрибутивов тут располагается и ядро.
* /dev - все подключенные флешки, клавиатуры, микрофоны, камеры. Это просто файлы-устройства в каталоге.
* /etc - содержатся конфигурационные файлы всех программ, установленных в системе.
* /home – содержит домашние каталоги всех обычных пользователей Linux.
* /root – каталог администратора.
* /proc - здесь содержится вся информация о запущенных процессах в реальном времени. По сути, это псевдофайловая система, содержащая подробную информацию о каждом процессе, его Pid, имя исполняемого файла, параметры запуска, доступ к оперативной памяти и так далее.
* /sbin – команды, предназначенные для управления системой. Все программы могут выполняться только суперпользователем.
* /srv – хранятся данные о серверных процессах (в некоторых дистрибутивах)
* /tmp – содержит временные файлы
* /var – содержит изменяющиеся файлы. Наиболее важными являются lock, log и mail. Log - здесь содержатся большинство файлов логов всех программ, установленных в операционной системе. Lock - здесь находятся файлы блокировок. Эти файлы означают, что определенный ресурс, файл или устройство занят и не может быть использован другим процессом. Mail -в эту папку почтовый сервер складывает все полученные или отправленные электронные письма, здесь же могут находиться его логи и файлы конфигурации.
* /usr – это самый большой каталог с большим количеством функций. Тут наиболее большая структура каталогов Linux. Здесь находятся исполняемые файлы, исходники программ, различные ресурсы приложений, картинки, музыку и документацию.



### 12. Понятие ядра операционной системы. Пространство ядра и пользовательское пространство.

Ядро операционной системы (Kernel) - часть операционной системы: постоянно находящаяся в оперативной памяти, управляющая всей операционной системой, содержащая: драйверы устройств, подпрограммы управления памятью, планировщик заданий, реализующая системные вызовы и т.п.



* Пространство ядра - адресное пространство виртуальной памяти операционной системы, которое резервируется для работы ядра операционной системы, его расширений и, возможно, некоторых драйверов устройств.
* Пользовательское пространство — адресное пространство виртуальной памяти операционной системы, отводимое для пользовательских программ.

### 13. Командный интерпретатор операционной системы.

Операционная система представляет собой программу, выполняющую системные вызовы. Редакторы, компиляторы, ассемблеры, компоновщики, утилиты и интерпретаторы команд по определению не являются частью операционной системы при всей своей важности и приносимой пользе. Рискуя внести некоторую путаницу, в этом разделе мы коротко рассмотрим и командный интерпретатор UNIX, называемый оболочкой. Например: для UNIX оболочкой можно назвать Bash.

**Командный интерпретатор** (оболочка) – это пользовательский интерфейс, через который человек работает с системой Linux. Оболочка является мощным языком программирования, с помощью которого можно автоматизировать процессы. Альтернативные оболочки: zsh, fish

### 14. Управление оперативной памятью в современных ОС. Виртуальная память.

Память является важнейшим ресурсом, требующим тщательного управления со стороны мультипрограммной операционной системы. Распределению подлежит вся оперативная память, не занятая операционной системой. Обычно ОС располагается в самых младших адресах, однако может занимать и самые старшие адреса.

Функциями ОС по управлению памятью в мультипрограммной системе являются:

* отслеживание свободной и занятой памяти;
* выделение памяти процессам и освобождение памяти по завершении процессов;
* вытеснение кодов и данных процессов из оперативной памяти на диск (полное или частичное), когда размеры основной памяти не достаточны для размещения в ней всех процессов, и возвращение их в оперативную память, когда в ней освобождается место;
* настройка адресов программы на конкретную область физической памяти.

Помимо первоначального выделения памяти процессам при их создании ОС должна также заниматься динамическим распределением памяти, то есть выполнять запросы приложений на выделение им дополнительной памяти во время выполнения. После того как приложение перестает нуждаться в дополнительной памяти, оно может возвратить ее системе. Выделение памяти случайной длины в случайные моменты времени из общего пула памяти приводит к фрагментации и, вследствие этого, к неэффективному ее использованию. Дефрагментация памяти тоже является функцией операционной системы.

Во время работы операционной системы ей часто приходится создавать новые служебные информационные структуры, такие как описатели процессов и потоков, различные таблицы распределения ресурсов, буферы, используемые процессами для обмена данными, синхронизирующие объекты и т. п. Все эти системные объекты требуют памяти»» В некоторых ОС заранее (во время установки) резервируется некоторый фиксированный объем памяти для системных нужд. В других же ОС используется более гибкий подход, при котором память для системных целей выделяется динамически. В таком случае разные подсистемы ОС при создании своих таблиц, объектов, структур и т. п. обращаются к подсистеме управления памятью с запросами.

Виртуальная память позволяет запускать программы, размер которых превышает объем установленной на машине физической памяти, за счет быстрого перемещения фрагментов адресного пространства между оперативной памятью и диском.

### 15. Работа ОС с внешними устройствами. Понятие драйвера устройства.

Существует два способа работы с внешними устройствами:

* **Синхронная** – процессор подает запрос внешнему устройству с помощью **драйвера** и ожидает завершения выполнения запроса. Системы с синхронной организацией работы ВУ **неэффективны** с точки зрения использования времени работы центрального процессора. Процессор часто «ожидает» выполнения запроса. Лучше всего **подходит для однопрограммных специализированных вычислительных систем**.
* **Асинхронная**

1. Программа инициирует прерывание “обращение к системе”, тем самым передается заказ на выполнение обмена, (параметры заказа могут быть переданы через специальные регистры, стек и т.п.) Происходит обработка прерывания (при этом программа (процесс) находится в ожидании). При обработке прерывания конкретному драйверу устройства передается заказ на выполнение обмена (который поступает в очередь).
2. После завершения обработки прерывания “обращение к системе” программа продолжает свое выполнение до завершения обмена (не всегда).
3. Выполнение программы приостанавливается по причине возникновения прерывания – завершение обмена с конкретным устройством. После обработки прерывания выполнение будет продолжено.

**Асинхронная схема** сглаживает системный дисбаланс между скоростью выполнения машинных команд и скоростью доступа к ВУ, она более **эффективна**, но **требует наличия развитого аппарата прерываний**. Такая схема используется чаще всего.

**Драйвер** — компьютерное программное обеспечение, с помощью которого другое программное обеспечение (напр. операционная система) получает доступ к аппаратному обеспечению некоторого устройства. Обычно с ОС поставляются драйверы для ключевых компонентов аппаратного обеспечения, без которых система не сможет работать. Для некоторых устройств (таких, как видеокарта или принтер) могут потребоваться специальные драйверы, обычно предоставляемые производителем устройства. *Драйвер не обязан взаимодействовать с аппаратными устройствами, он может их только имитировать.*

### 16. Многозадачность современных операционных систем.

**Многозада́чность** — свойство операционной системы или среды выполнения обеспечивать возможность параллельной (или псевдопараллельной) обработки нескольких задач. Истинная многозадачность операционной системы возможна только в распределённых вычислительных системах.

Существует 2 типа многозадачности:

* **Процессная многозадачность** (основанная на процессах — одновременно выполняющихся программах). Здесь программа — наименьший элемент управляемого кода, которым может управлять планировщик операционной системы. Более известна большинству пользователей (работа в текстовом редакторе и прослушивание музыки).
* **Поточная многозадачность** (основанная на потоках). Наименьший элемент управляемого кода — поток (одна программа может выполнять 2 и более задачи одновременно).

**Многопоточность** — специализированная форма многозадачности, реализующая ее с помощью нескольких потоков.

Многозадачность обладает следующими особенностями:

* Каждая задача имеет свой приоритет, в соответствии с которым получает процессорное время и память
* Система организует очереди задач так, чтобы все задачи получили ресурсы, в зависимости от приоритетов и стратегии системы
* Система организует обработку прерываний, по которым задачи могут активироваться, деактивироваться и удаляться
* По окончании положенного кванта времени ядро временно переводит задачу из состояния выполнения в состояние готовности, отдавая ресурсы другим задачам. При нехватке памяти страницы невыполняющихся задач могут быть вытеснены на диск (своппинг), а потом, через определённое системой время, восстанавливаться в памяти
* Система обеспечивает защиту адресного пространства задачи от несанкционированного вмешательства других задач
* Система обеспечивает защиту адресного пространства своего ядра от несанкционированного вмешательства задач
* Система распознаёт сбои и зависания отдельных задач и прекращает их
* Система решает конфликты доступа к ресурсам и устройствам, не допуская тупиковых ситуаций общего зависания от ожидания заблокированных ресурсов
* Система гарантирует каждой задаче, что рано или поздно она будет активирована
* Система обрабатывает запросы реального времени
* Система обеспечивает коммуникацию между процессами

### 17. Графический и текстовый интерфейс взаимодействия с ОС.

**Текстовый пользовательский интерфейс** (TUI ИЛИ CUI)— разновидность интерфейса пользователя, использующая при вводе-выводе и представлении информации исключительно набор буквенно-цифровых символов и символов псевдографики.

В простейшем случае текстовый интерфейс использует интерфейс командной строки, однако многие программы с помощью интерактивных элементов создают более дружественный интерфейс, приближающийся по удобству к графическому.

Программы с **текстовым интерфейсом** могут реализовывать оконный интерфейс, чему особенно способствует применение псевдографических символов => **текстовом интерфейсе** могут быть реализованы все базовые элементы интерфейса, используемые и в графическом интерфейсе (кнопки, меню)

На программном уровне для ввода и вывода информации консольные программы используют стандартные устройства ввода-вывода (**stdin, stdout, stderr**)

**Графический интерфейс пользователя** (GUI) — разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.), представленные пользователю на дисплее, исполнены в виде графических изображений. Также называется графической оболочкой управления.

В отличие от TUI строки, в GUI пользователь имеет произвольный доступ (с помощью устройств ввода — клавиатуры, мыши, джойстика и т. п.) ко всем видимым экранным объектам (элементам интерфейса) и осуществляет непосредственное манипулирование ими. Чаще всего элементы интерфейса в GUI реализованы на основе метафор и отображают их назначение и свойства, что облегчает понимание и освоение программ неподготовленными пользователями.

### 18. Источники установки программных приложений. Программные репозитории.

**Установка программного обеспечения, инсталляция** — процесс установки программного обеспечения на компьютер конечного пользователя. Выполняется особой программой (пакетным менеджером), присутствующей в операционной системе (например, RPM, APT или dpkg в Linux, Установщик Windows в Microsoft Windows), или же входящим в состав самого программного обеспечения средством установки. В операционной системе GNU очень распространено использование системы GNU toolchain и её аналогов для компиляции программного обеспечения непосредственно перед установкой.

Источники остановки программных приложений зависят от используемой ОС:

**Семейство Windows**

Наиболее популярным форматом для семейства Windows NT является установочный пакет MSI, который устанавливается посредством Установщика Windows. Компании, производящие средства для создания инсталляторов: InstallShield (InstallShield Wizard), Macrovision (InstallAnywhere), Wise Solutions, Inc., SetupBuilder. Большинство из этих средств могут создавать как пакеты MSI, так и свои собственные пакеты.

Бесплатными альтернативами являются NSIS, Clickteam Install Creator, Inno Setup, Install Simple, а также инструментальные средства от Microsoft (WiX).

**Семейство UNIX**

Большинство дистрибутивов операционных систем на базе GNU, Linux и BSD имеет встроенные системы управления пакетами, с помощью которых можно устанавливать как необходимые компоненты операционной системы, так и стороннее программное обеспечение, зачастую даже если оно использует собственный установщик, которым это не предусмотрено.

**Семейство macOS**

macOS также использует систему управления пакетами AppStore, есть утилита для командной строки Homebrew, которая позволяет ставить пакеты так же, как это делается в дистрибутивах UNIX. Некоторые коммерческие приложения для macOS используют отдельный установщик, например, Installer VISE или Stuffit InstallerMaker. Приложения, которые не нуждаются в установке дополнительных компонентов системы, могут быть установлены посредством простого копирования файлов приложения в нужное место хранилища (HDD/SSD). macOS также включает отдельное приложение для обновления ОС Software Update.

**Программные репозитории** (Personal Package Archive / PPA) – это набор файлов, которые содержат информацию о различном программном обеспечении, его версиях и некоторых других деталях, таких, как контрольная сумма.

Репозитории отличаются в зависимости от дистрибутива, его версии, сборки.

Репозитории можно менять руками, так же их может быть несколько. Например, дистрибутив Ubuntu имеет сразу 4 репозитория:

* **Основное** - бесплатное программное обеспечение с открытым исходным кодом, поддерживаемое Canonical.
* **Universe** - поддерживаемое сообществом бесплатное программное обеспечение с открытым исходным кодом.
* **Restricted** - проприетарные драйверы для устройств.
* **Multiverse** - Программное обеспечение ограничено авторским правом или правовыми вопросами.

### 19. Способы установки программ в Linux.

Способ 1: Менеджер пакетов и официальные хранилища:

Debian/Ubuntu/Linux Mint: Менеджер **apt** (Пример установки Mozilla Firefox)

* + 1. Открываем консоль(терминал) **CTRL + ALT + T**;
    2. Удостоверьтесь что все установленные пакеты обновлены “**sudo apt-get update**”;
       1. Так как все действия через утилиту **apt** производятся от имени суперпользователя, поэтому надо будет указать пароль от этой учётной записи.
    3. После успешного обновления пишем “**sudo apt-get install firefox**”
    4. Ожидаем; (*Примечание: для успешной установки будет нужен доступ к интернету*)

В последней версии Ubuntu писать **apt-get** не обязательно. Достаточно будет написать **apt** и далее **install**.

Примеры других популярных приложений:

**sudo apt install vlc** — видеоплеер.  
 **sudo apt install gnome-music** — музыкальный плеер.  
 **sudo apt install gimp** — графический редактор.  
 **sudo apt install gparted**— ПО управления разделами жёсткого диска.

Способ 2: Менеджер пакетов и пользовательские хранилища:

Помимо официальных хранилищ различных приложений существуют ещё и пользовательские. Если вы не имеете адреса репозитория, сначала следует его отыскать. Проще всего это сделать через специальный сайт, а вся процедура выглядит так:

Переходим на сайт Launchpad “**https://launchpad.net**”;

* + 1. В строке поиска вводим название софта. Для удобства можно дописать **PPA**, что означает пользовательское хранилище;
    2. В результатах отыщите подходящий вариант и нажмите на соответствующую ссылку;
    3. Ознакомьтесь с возможными пакетами и выберите подходящий;
    4. Перейдите на страницу ПО;
    5. Попав на страницу **PPA**, внизу вы увидите команды, с помощью которых и производится инсталляция;

Debian/Ubuntu/Linux Mint: (Пример добавления Chromium)

* + 1. Запускаем консоль(терминал);
    2. Отыскиваем ссылку на репозиторий. Пишем “**sudo add-apt-repository ppa:saiarcot895/chromium-dev”;**
    3. Подтверждаем действие, введя пароль;
    4. По окончании процедуры обновите системные библиотеки “**sudo apt-get update”;**
    5. Используем команду “**sudo apt install chromium-browser”**;
    6. Подтверждаем действие “**Y”**;
    7. Ожидаем; (*Примечание: для успешной установки будет нужен доступ к интернету*)

Способ 3: Установка DEB-пакетов:

Формат файлов DEB используется для распространения программного обеспечения и является стандартным типом данных операционных систем на базе Debian. В таких дистрибутивах по умолчанию установлены инструменты инсталляции софта такого формата как через графическую оболочку, так и через «Терминал».

**Способ 1: Использование браузера:**

1. Запустите браузер из меню или панели задач и перейдите на необходимый сайт, где следует отыскать рекомендованный пакет формата DEB. Кликните на соответствующую кнопку для начала скачивания.
2. После появления всплывающего окна отметьте маркером пункт «Открыть в», выберите там «Установка приложений (по умолчанию)», а затем щелкните на «ОК».
3. Запустится окно инсталлятора, в котором следует нажать на «Установить».
4. Введите свой пароль, чтобы подтвердить начало установки.
5. Ожидайте завершения распаковки и добавления всех нужных файлов.

**Способ 2: Стандартный установщик приложений:**

1. Запустите «Менеджер пакетов» и воспользуйтесь панелью навигации слева, чтобы перейти в папку хранения софта.
2. Нажмите правой кнопкой мыши по программе и выберите пункт «Открыть в Установка приложений».
3. Проведите процедуру инсталляции аналогичную той, которую мы рассмотрели в предыдущем способе.

**Способ 3: Утилита GDebi:**

1. Сначала разберёмся, как сделать это через «Терминал». Откройте меню и запустите консоль либо щелкните ПКМ по рабочему столу и выберите соответствующий пункт.
2. Введите команду ”**sudo apt install gdebi”** и нажмите на Enter;
3. Впишите пароль для учетной записи;
4. Подтвердите операцию “**Y**”

### 20. Процессы в Linux. Функции, организация, управление.

Процессы - это по сути работающие программы, которые могут быть запущены как самой системой, так и любым пользователем. Данные программы могут работать как в фоновом режиме и быть невидимыми, так и на переднем плане и иметь своеобразный видимый интерфейс. Для выполнения каждого процесса приходится выделять память и процессорное время.

Выполнение процессов в системе UNIX осуществляется на двух уровнях: уровне пользователя и уровне ядра. Когда процесс производит обращение к операционной системе, режим выполнения процесса переключается с режима задачи (пользовательского) на режим ядра.

Основные различия между этими двумя режимами:

1. В режиме задачи процессы имеют доступ только к своим собственным инструкциям и данным, но не к инструкциям и данным ядра;
2. Некоторые машинные команды являются привилегированными и вызывают возникновение ошибок при попытке их использования в режиме задачи

К процессам часто обращаются, когда компьютер содержит вирус или другое вредоносное ПО. Взглянув на процессы можно найти аномальное поведение системы и найти вредоносное ПО. Поэтому даже обычному пользователю необходимо знать основы работы с процессами. Прежде, чем мы рассмотрим как управлять процессами, перечислим основные параметры, характерные для каждого процесса:

* **PID** - (process ID) идентификатор каждого процесса.
* **PPID** - (parent process ID) идентификатор родительского процесса. Процесс может порождать и другие процессы.
* **UID, GID** - реальные идентификаторы пользователя и его группы, запустившего данный процесс.
* **EUID, EGID** - эффективные идентификаторы пользователя и его группы.
* **Priority/Nice** - приоритет и относительный приоритет.
* **STAT** - состояние процесса.приорит

***Как же можно увидеть все процессы?***

* Список всех процессов ”**ps aux”**.
* Отображение всех процессов, включая и PPID **”ps -ef”**.
* Отображение дерева процессов, то есть схематическое отображение от какого процесса был порожден конкретный процесс ”**ps auxf**”.
* Отображение дерева процессов, включая PPID ”**ps ajxf**”.
* Более короткий вывод дерева процессов ”**pstree**”.
* Отображение процессов конкретного пользователя ”**ps -U пользователь**”.

***Управление процессами в Linux***

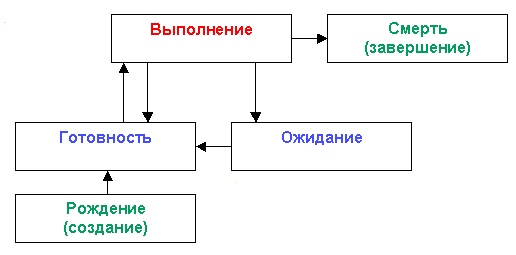
В Linux есть очень большое количество утилит для решения различных задач по управлению процессами. Это и такие многофункциональные решения, как **htop**, **top**, а также простые утилиты - **ps**, **kill**, **killall**, **who** и т д. Мы остановимся на работе с программой **htop** и ее аналогами в форме утилит в стиле GNU, одна утилита - одна функция.

Давайте установим **htop**, если она у вас еще не установлена. В Ubuntu это делается так “**sudo apt install htop”**.

***Изменение приоритета процессов***

Приоритет процесса Linux означает, насколько больше процессорного времени будет отдано этому процессу по сравнению с другими. Так мы можем очень тонко настроить какая программа будет работать быстрее, а какая медленнее. Значение приоритета может колебаться от 19 (минимальный приоритет) до -20 - максимальный приоритет процесса Linux. Причём, уменьшать приоритет можно с правами обычного пользователя, но чтобы его увеличить нужны права суперпользователя. Например, команда **nice**. С помощью нее вы можете указать приоритет для запускаемого процесса “**nice -n 10 apt-get upgrade**”.

### 21. Жизненный цикл процесса операционной системы.



Процессы в системе UNIX могут находиться в различных логических состояниях и переходить из состояния в состояние в соответствии с установленными правилами перехода, при этом информация о состоянии сохраняется в таблице процессов и в адресном пространстве процесса.

Полный набор состояний процесса содержится в следующем перечне:

1. Процесс выполняется в режиме задачи.
2. Процесс выполняется в режиме ядра.
3. Процесс не выполняется, но готов к запуску под управлением ядра.
4. Процесс приостановлен и находится в оперативной памяти.
5. Процесс готов к запуску, но программа подкачки (нулевой процесс) должна еще загрузить процесс в оперативную память, прежде чем он будет запущен под управлением ядра.
6. Процесс приостановлен, и программа подкачки выгрузила его во внешнюю память, чтобы в оперативной памяти освободить место для других процессов.
7. Процесс возвращён из привилегированного режима (режима ядра) в непривилегированный (режим задачи), ядро резервирует его и переключает контекст на другой процесс.
8. Процесс вновь создан и находится в переходном состоянии; процесс существует, но не готов к выполнению, хотя и не приостановлен. Это состояние является начальным состоянием всех процессов, кроме нулевого.
9. Процесс вызывает системную функцию **exit** и прекращает существование. Однако, после него осталась запись, содержащая код выхода, и некоторая хронометрическая статистика, собираемая родительским процессом. Это состояние является последним состоянием процесса.

### 22. Основные команды Linux для управления файлами.

1. **ls опции /путь/к/папке**

Опции команды указывают как именно и в каком виде нужно выводить информацию на экран, а путь, это та папка, которую нужно отобразить. Рассмотрим основные опции утилиты:

* **-a** - отображать все файлы, включая скрытые, это те, перед именем которых стоит точка;
* **-A** - не отображать ссылку на текущую папку и корневую папку . и ..;
* **--author** - выводить создателя файла в режиме подробного списка;
* **-b** - выводить Escape последовательности вместо непечатаемых символов;
* **--block-size** - выводить размер каталога или файла в определенной единице измерения, например, мегабайтах, гигабайтах или килобайтах;
* **-B** - не выводить резервные копии, их имена начинаются с ~;
* **-c** - сортировать файлы по времени модификации или создания, сначала будут выведены новые файлы;
* **-C** - выводить колонками;
* **--color** - включить цветной режим вывода, автоматически активирована во многих дистрибутивах;
* **-d**- выводить только директории, без их содержимого, полезно при рекурсивном выводе;
* **-D** - использовать режим вывода, совместимый с Emacs;
* **-f** - не сортировать;
* **-F** - показывать тип объекта, к каждому объекту будет добавлен один из специализированных символов \*/=>@|;
* **--full-time** - показывать подробную информацию, плюс вся информация о времени в формате ISO;
* **-g** - показывать подробную информацию, но кроме владельца файла;
* **--group-directories-first** - сначала отображать директории, а уже потом файлы;
* **-G** - не выводить имена групп;
* **-h** - выводить размеры папок в удобном для чтения формате;
* **-H** - открывать символические ссылки при рекурсивном использовании;
* **--hide** - не отображать файлы, которые начинаются с указанного символа;
* **-i** - отображать номер индекса inode, в которой хранится этот файл;
* **-l** - выводить подробный список, в котором будет отображаться владелец, группа, дата создания, размер и другие параметры;
* **-L** - для символических ссылок отображать информацию о файле, на который они ссылаются;
* **-m** - разделять элементы списка запятой;
* **-n** - выводить UID и GID вместо имени и группы пользователя;
* **-N** - выводить имена как есть, не обрабатывать контролирующие последовательности;
* **-Q** - брать имена папок и файлов в кавычки;
* **-r** - обратный порядок сортировки;
* **-R** - рекурсивно отображать содержимое поддиректорий;
* **-s** - выводить размер файла в блоках;
* **-S** - сортировать по размеру, сначала большие;
* **-t**- сортировать по времени последней модификации;
* **-u** - сортировать по времени последнего доступа;
* **-U** - не сортировать;
* **-X** - сортировать по алфавиту;
* **-Z** - отображать информацию о расширениях SELinux;
* **-1** - отображать один файл на одну строку.

1. **cd опции папка\_назначения**

Опций всего две, это -P и -L. Они влияют на то, как будут обрабатываться символы возврата в предыдущую директорию и символические ссылки:

* **-P** - позволяет следовать по символическим ссылкам перед тем, как будут обработаны все переходы "..";
* **-L** - переходит по символическим ссылкам только после того, как были обработаны "..";
* **-e** - если папку, в которую нужно перейти не удалось найти - выдает ошибку.

Дальше нужно указать директорию, в которую следует перейти. Если этого не сделать, а вызвать cd без параметров, то рабочей папкой будет выбран ваш домашний каталог.

1. **rm**

удаление файла. Не запрашивает предупреждение!

rm –r – удаление папки.

1. **rmdir**

удаление пустой папки.

1. **mv**

перемещает файл в новое место. Она также может использоваться для переименования файлов. Например, **mv file newfile** переименует файл file в newfile.

Чтобы переместить файл в другую папку нужно указать путь к ней после названия файла.

1. **cp**

Копирование файлов. Работает как mv, только исходный файл также остается на своем месте.

cp –r – рекурсивно копирует всю папку со всеми файлами и вложенными папками в новое место.

1. **mkdir**

создает новую папку.

1. **ln**

позволяет создавать жесткие и символические ссылки на файлы или папки. Для создания символической ссылки используется опция -s. Для создания жесткой ссылки никаких опций задавать не нужно.

1. **chmod**

позволяет изменить права доступа к файлам. Например, **chmod +x script.sh** добавляет флаг исполняемости к файлу script.sh:  chmod +x  script.sh

1. **touch**

создать пустой файл.

### 23. Основные команды Linux для управления каталогами.

* mkdir\rmdir – позволяет создать или удалить директорию
* pwd – позволяет узнать текущую директорию
* cd – позволяет переместиться в указанную директорию
* ls – позволяет просмотреть (в определенном порядке) содержимое директории
* find – позволяет найти конкретные файлы (согласно определенному принципу) в директории
* du – показывает размер директории
* cp (-r или –a) – копирование каталогов (неявно, с помощью указанных в скобочках опций)
* mv – перемещение каталогов

### 24. Основные команды Linux для управления файловыми системами.

* **badblocks** - если у вас старый жесткий диск и на нем накопилось много битых блоков, вы можете с помощью этой утилиты пометить их все на уровне файловой системы, чтобы больше не использовать.
* **e2label** - позволяет изменить метку раздела с файловой системой ext.
* **fsck** - проверка файловой системы linux и исправление найденных ошибок
* **mkfs** - позволяет создать файловую систему Linux.
* **resize2fs** - изменить размер раздела с файловой системой
* **tune2fs** - позволяет изменить файловую систему Linux, настроить ее параметры.

### 25. Межпроцессное взаимодействие. Проблема и способы решения.

***Межпроцессное взаимодействие*** (от англ. Inter-Process Communication или сокр. IPC) - набор способов обмена данными между множеством потоков в одном или более процессах. Ситуации, когда приходится процессам взаимодействовать:

* Передача информации от одного процесса другому
* Контроль над деятельностью процессов (например: когда они борются за один ресурс)
* Согласование действий процессов (например: когда один процесс поставляет данные, а другой их выводит на печать. Если согласованности не будет, то второй процесс может начать печать раньше, чем поступят данные).

***Проблема***: Ситуации, в которых два (и более) процесса считывают или записывают данные одновременно и конечный результат зависит от того, какой из них был первым, называются **состояниями состязания**. Часть программы, в которой есть обращение к совместно используемым данным, называется **критической областью** или критической секцией.

***Решения проблемы:***

1. Избежание нахождения двух процессов в одной критической области. Для этого необходимо выполнение четырёх условий (Условия избегания состязания и эффективной работы процессов):

* Два процесса не должны одновременно находиться в критических областях.
* В программе не должно быть предположений о скорости или количестве CPU.
* Процесс, находящийся вне критической области, не может блокировать другие процессы.
* Невозможна ситуация, в которой процесс вечно ждет попадания в критическую область.

1. **Взаимное исключение** с целью избежать вмешательства в критическую область одного процесса при нахождении там другого и связанных с этим проблем. Всего существует три метода взаимного исключения:

* **Запрещение всех прерываний** при входе процесса в критическую область и разрешение прерываний по выходе из области (поскольку CPU переключается с одного процесса на другой только по прерыванию, отключение прерываний исключает передачу CPU другому процессу).
* **Переменные блокировки**: рассмотрим одну совместно используемую переменную блокировки, изначально равную 0. Если процесс хочет попасть в критическую область, он предварительно считывает значение переменной блокировки. Если переменная равна 0, процесс изменяет ее на 1 и входит в критическую область. Если же переменная равна 1, то процесс ждет, пока ее значение сменится на 0. Таким образом, 0 означает, что ни одного процесса в критической области нет, а 1 означает, что какой-либо процесс находится в критической области.
* **Строгое чередование:** создается переменная turn, принимающая значения 0 или 1. Каждому из двух процессов присваивается определенный идентификатор (0 или 1 соответственно). Если значения переменной и идентификатора совпадают, то процесс входит в критическую область, а после выхода из нее меняет значение (например с 0 на единицу). Если же значения не совпадают, то процесс попадает в цикл, в котором проверяет значение переменной до тех пор, пока оно не будет совпадать с его идентификатором (после заходит в область).

1. Программное решение взаимного исключения без участия строго чередования – ***Алгоритм Петерсона***
2. Взаимное исключение с участием аппаратного обеспечения – ***команда TSL***

### 26. Понятие пакета в Linux. Менеджеры пакетов.

Под пакетами в **Linux**подразумевается программное обеспечение (ПО), которое необходимо установить на компьютер. Пакетсодержит собранную программу, информацию о том, какие требуется совершить действия для ее установки, информацию о зависимостях, архив встроенных файлов, а также, возможно, много других данных **(в зависимости от вида пакета)**. За установку (удаление, обновление) пакетов отвечают **менеджеры пакетов**. Пакеты бывают бинарными (исполняемые файлы) и включающие исходные коды программ.

**Система управления пакетами** (Менеджер пакетов) - набор программного обеспечения, позволяющего управлять процессом установки, удаления, настройки и обновления различных компонентов программного обеспечения. Системы управления пакетами активно используются в различных дистрибутивах операционной системы Linux и других UNIX-подобных операционных системах. Наиболее популярные системы управления:

* Node.js Package Manager (npm)
* Red Hat Package Manager (RPM) (Дополнительные утилиты: YUM, DNF)
* Dpkg (Для работы в Debian) (Дополнительные утилиты: APT, APM, Synaptic)
* Pacman (Arch Linux)
* Portage (Gentoo)
* Entropy (Sabayon)
* Image Packaging System (IPS)
* Zypper (Opensuse) и другие

### 27. Основные команды Linux для управления пакетами.

**apt (Debian/Ubuntu/Mint)**

1. apt install имя\_пакета. Установить нужный пакет.
2. apt-add-repository адрес\_репозитария. Добавить сторонний репозитарий.
3. apt update. Обновить сведения о пакетах.
4. apt upgrade . Обновить все пакеты до самых свежих (выполнять после apt update).
5. apt remove имя\_пакета . Удалить ненужный пакет.
6. apt purge имя\_пакета . Удалить ненужный пакет со всеми зависимостями, если хотите освободить больше места.
7. apt autoremove . Удалить все ненужные зависимости, бесхозные пакеты и прочий мусор.
8. dpkg. Средство обработки запросов.
9. dpkg –l | grep gnome. *Поиск установленных пакетов*, удовлетворяющих критерию поиска

### 28. Основные команды Linux для управления процессами.

1. kill. Эта команда служит для принудительного завершения процессов. Нужно ввести kill PID\_процесса. PID процесса можно узнать, введя top.
2. xkill. Ещё одна команда для завершения процессов. Введите её, затем щёлкните по тому окну, которое нужно закрыть.
3. killall. Убивает процессы c определённым именем. К примеру, killall firefox.
4. top. Отображает перечень запущенных процессов, сортируя в зависимости от потребления ресурсов CPU. Своего рода терминальный «Системный монитор».
5. ps. Предназначена для вывода информации о выполняемых процессах. Данная команда имеет много параметров, о которых вы можете прочитать в руководстве (man ps).
6. nice. Команда nice выполняет указанную команду с пониженным приоритетом, коэффициент понижения указывается в диапазоне 1..19 (по умолчанию он равен 10). Суперпользователь может повышать приоритет команды, для этого нужно указать отрицательный коэффициент, например --10. Если указать коэффициент больше 19, то он будет рассматриваться как 19.
7. jobs. Команда jobs выводит список процессов, которые выполняются в фоновом режиме
8. fg. fg - переводит процесс в нормальные режим ("на передний план" - foreground)
9. bg. bg – перевод процесса в фоновый режим. Запустить программу в фоновом режиме можно с помощью конструкции <команда> &

### 29. Типы файлов в Linux. Ссылки.

**ТИПЫ ФАЙЛОВ В LINUX**

Файлы в операционной системе Linux можно поделить на **три основных типа**:

1. Обычные файлы, для хранения информации
2. Специальные файлы - для устройств и туннелей
3. Директории

**1. ОБЫЧНЫЕ ФАЙЛЫ**

Это файлы, с которыми мы привыкли работать каждый день, они могут содержать текст, исполняемые инструкции для программ, изображения или другую информацию. Это самый распространенный тип файлов, которые вы можете найти в системе Linux. Рассмотрим небольшой список относящихся сюда файлов:

1. Текстовые файлы
2. Исполняемые файлы
3. Файлы изображений
4. Файлы архивов
5. Файлы библиотек программ
6. И другие подобные типы

**2. СПЕЦИАЛЬНЫЕ ФАЙЛЫ**

Предназначены для обмена информации с ядром, работы с устройствами или общения между программами. Такие файлы могут тоже быть нескольких типов, в зависимости от назначения.

**Блочные файлы** - это файлы устройств, которые обеспечивают буферизованный доступ к аппаратным компонентам. Такие файлы могут передать большой блок данных за небольшой один раз.

**Символьные файлы** обеспечивают не буферизованный доступ к аппаратным компонентам и ядру. Поскольку у них нет буфера, они позволяют передавать только по одному символу за один раз. А в остальном, это такие же файлы устройств, как и блочные файлы.

**Символические ССЫЛКИ** - это файлы, которые указывают на другие файлы в системе по их имени. Они могут указывать как на обычные файлы, таки на каталоги или другие типы файлов в linux. По сути, это те же ярлыки Windows.

В Linux еще есть жесткие ссылки, но они не имеют отношения к типу файлов, потому что реализованы на уровне файловой системы и считаются обычными файлами. Поскольку они указывают на одно и то же место на диске, это два разных файла, с одинаковым содержимым.

**Туннели и именованные туннели** - это файлы, позволяющие настроить связь между двумя процессами перенаправив вывод одного процесса на вход другого. Именованные туннели используются для связи между двумя процессами и работают так же как и обычные туннели. Обозначаются такие типы файлов linux буквой p (pipe).

**Файлы сокетов** - это файлы, обеспечивающие прямую связь между процессами, они могут передавать информацию между процессами, запущенными в разных средах или даже разных машинах. Это значит, что с помощью сокетов программы могут обмениваться данными даже по сети. По сути, сокет работает так же как туннели, но только в обе стороны.

**3. КАТАЛОГИ**

*Это специальные файлы, которые позволяют объединять другие файлы и каталоги в группы для более простой навигации и поиска*. Естественно, они могут содержать как обычные, так и специальные файлы, одним словом любые типы файлов ос linux. В системе Linux, файлы организуются в папки начиная от корня (/)

### 30. Основные команды для управления текстовыми потоками.

Поток номер 0 (stdin) зарезервирован для чтения команд пользователя или входных данных. При интерактивном запуске программы по умолчанию нацелен на чтение с устройства текстового интерфейса пользователя (клавиатуры). Командная оболочка UNIX (и оболочки других систем) позволяют изменять цель этого потока с помощью символа «<». Системные программы (демоны и т. п.), как правило, не пользуются этим потоком.

Поток номер 1 (stdout) зарезервирован для вывода данных, как правило (хотя и не обязательно) текстовых. При интерактивном запуске программы по умолчанию нацелен на запись на устройство отображения (монитор). Командная оболочка UNIX (и оболочки других систем) позволяют перенаправить этот поток с помощью символа «>». Средства для выполнения программ в фоновом режиме (например, nohup) обычно переназначают этот поток в файл.

Поток номер 2 (stderr) зарезервирован для вывода диагностических и отладочных сообщений в текстовом виде. Чаще всего цель этого потока совпадает с stdout, однако, в отличие от него, цель потока stderr не меняется при «>» и создании конвейеров («|»). То есть, отладочные сообщения процесса, вывод которого перенаправлен, всё равно попадут пользователю. Командная оболочка UNIX позволяет изменять цель этого потока с помощью конструкции «2>». Например, для подавления вывода этого потока нередко пишется «2>/dev/null».

### 31. Основные команды для просмотра и редактирования текстовых файлов.

Просмотр файлов:

1. cat адрес\_файла. Выводит все содержимое файла в стандартный вывод.(Пример: **cat /etc/passwd**)

2. tac адрес\_файла. То же, что и cat, но в обратном порядке.

3. less опции файл. Вывод содержимого файла. Удобно для больших файлов. Выход – q.

4. head адрес\_файла. Вывод по умолчанию первых 10 строк. (Пример: **head /etc/passwd**).

5. tail адрес\_файла. Вывод по умолчанию последних 10 строк.

6. more опции адрес\_файла. Чтение определенных строк. В опциях: -5 (вывод 5 строк), +5 (вывод с 5ой строки). (Пример: **more +5 less.txt**)

Редактирование файлов:

1. nano путь/к/файлу. (Пример: **nano /etc/stud.** Или: **sudo** **nano /etc/stud**)

### 32. Регулярные выражения.

Регулярные выражения - это инструмент для поиска текста по шаблону, обработки и изменения строк, который можно применять для решения множества задач, таких, как:

* Проверка ввода текста;
* Поиск и замена текста в файле;
* Пакетное переименование файлов;
* Взаимодействие  с сервисами, таким как Apache;
* Проверка строки на соответствие шаблону.

Регулярные выражения состоят из шаблонов, вернее сказать задают шаблон поиска. Шаблон состоит из правил поиска, которые составляются из символов и метасимволов.

Правила:

* | - разделяет допустимые варианты (логическое ИЛИ). (Пример: «gray|grey» соответствует gray или grey).
* () - определения области действия и приоритета операторов. (Пример: «gray|grey» и «gr(a|e)y» являются разными образцами, но они оба описывают множество, содержащее gray и grey).
* {} - определяет, сколько раз предшествующее выражение может встречаться. (Пример: {n} - ровно n повторений)
* ? - то же самое, что и {0,1}. (Пример: «colou?r» соответствует и color, и colour).
* \* - любое число повторений или их отсутствие. ( Пример: «go\*gle» соответствует, gogle, gooооgle…)
* + - хотя бы 1 раз ({1,}). (Пример: «go+gle» соответствует google, goooogl... но не gogle)

Метасимволы(символы, которые не соответствуют своему реальному значению):

В принципе можно посмотреть на правила…

!!Чтобы получить просто символ, необходимо предварить (экранировать) метасимвол (. \* + \ ? [ ] { } ) обратным слешем. Например, \. или \[

Пример:

**regex@regexp:$ cat text1**

**1 apple**

**2 pear**

**3 banana**

**regex@regexp:$ grep "[2p]" text1 # поиск строк, где есть 2 или p**

**1 apple**

**2 pear**

### 33. Встроенные и внешние команды bash.

Встроенные (внутренние) команды:

1. echo (или printf)

2. exit

3. cd

4. и тд и тп. Посмотреть список можно командой **help** или **help | less**

Внешние команды:

1. echo

2. cat

3. ls

4. rm (удалефуоние)

5. mkdir

6. chmod

7. иные

!!Узнать, встроенная команда или внешняя, можно: type -a команда (например, ls)

Пример:

**regex@regexp:$ type -a cd uname : ls uname**

**cd is a shell builtin**

**uname is /bin/uname**

**: is a shell builtin**

**ls is aliased to `ls --color=auto'**

**ls is /bin/ls**

**l is a function**

**l ()**

**{**

**ls --color=auto**

**}**

### 34. Основные конфигурационные файлы Linux.

Операционная система Linux в отличие от Windows не имеет общего реестра для хранения настроек системы, все настройки хранятся в конфигурационных файлах. Большинство этих файлов размещено в папке /etc/.

Настройки большинства системных и сторонних программ находятся в этих файлах, это могут быть настройки графического сервера, менеджера входа, системных служб, веб-сервера, системы инициализации.

1. /ETC/ADJTIME

Этот конфигурационный файл отвечает за настройку формата системного времени и читается службой systemd-timedated. Время может быть представлено в двух вариантах: LOCAL - время текущего часового пояса и UTC - время по Гринвичу. Вы можете вручную менять значение или воспользоваться утилитой timedatectl.

2. /ETC/BASH.BASHRC

Этот файл принадлежит командной оболочке bash. Это не совсем конфигурационный файл - а скрипт, его содержимое выполняется при запуске каждого экземпляра bash для настройки оболочки. Точно так же выполняется содержимое файла ~/.bashrc для каждого пользователя.

3. /ETC/CRONTAB

Crontab - файл [настройки планировщика cron](https://losst.ru/kak-dobavit-komandu-v-cron). Здесь записываются все задания, которые должен выполнить планировщик, а также время и периодичность. Этот файл не принято редактировать напрямую. Для этого используется утилита **crontab -e**.

4. /ETC/ENVIRONMENT

Здесь содержатся[переменные окружения](https://losst.ru/peremennye-okruzheniya-v-linux), которые будут загружены для каждого сеанса терминала, независимо от того запущен он на локальной машине или по ssh. Файл читается скриптами Bash во время инициализации оболочки.

5. /ETC/FSTAB

Наверное, все уже знают файл /etc/fstab. Здесь выполняется настройка монтирования файловых систем во время загрузки. В современных системах он читается **systemd** и все записи на ходу транслируются в юнит-файлы, с помощью которых уже выполняется монтирование. Смотрите также: [автоматическое монтирование fstab](https://losst.ru/avtomaticheskoe-montirovanie-fstab-i-systemd).

6. /ETC/GROUP

В этом файле хранятся все [группы пользователей](https://losst.ru/gruppy-polzovatelej-linux), которые есть в системе. С помощью него вы можете посмотреть список групп, их идентификаторы или добавить новые. Но добавлять группы с помощью редактирования файла не принято, для этого есть утилита **usermod**.

### 

7. /ETC/HOSTNAME

В этом файле содержится имя хоста, файл будет прочитан во время загрузки системы и указанное имя компьютера установится в системе. Вы будете его видеть в приглашении ввода терминала или в информации о системе.

8. /ETC/HOSTS

Файл /etc/hosts позволяет задавать псевдонимы для различных сетевых узлов. Таким образом, компьютер не обращается к DNS для получения IP домена, а берет его из hosts. Это позволяет, например, заблокировать доступ к нежелательным сайтам просто перенаправив их на localhost или же получить доступ к сайту по ip, которому еще не присвоен домен.

9. /ETC/HOSTS.ALLOW И /ETC/HOSTS.DENY

С помощью этих двоих файлов можно настраивать права доступа ко всем локальным службам. Например, вы можете разрешить доступ к службе apache только с локального компьютера. Это очень сильно повысит безопасность системы, если ваш компьютер подключен к публичной сети.

10. /ETC/ISSUE И /ETC/ISSUE.NET

Баннер, который будет выводиться при входе в командную оболочку локально или по SSH. Обычно там выводится версия ядра и дистрибутива Linux, но вы можете заменить эту информацию по своему усмотрению.

11. /ETC/LD.SO.CONF

В этом файле содержатся пути к папкам, в которых компоновщик linux ld.so будет искать динамические библиотеки во время запуска программ. Папки /lib64, /lib, /usr/lib64 и /usr/lib будут проверены автоматически.

12. /ETC/LOCALTIME

Это символическая ссылка, которая указывает на файл часового пояса в папке /usr/share/zoneinfo/. Редактировать файл не нужно, а для изменения настроек нужно создать символическую ссылку на другую временную зону.

13. /ETC/LOGIN.DEFS

Файл /etc/login.defs отвечает за настройку поведения утилиты управления пользователями и параметры входа в систему. Вы можете настроить какой минимальный и максимальный id нужно выдавать, что делать с папкой пользователя при удалении и многое другое, количество попыток входа и таймаут, а также многое другое.

14. /ETC/MIME.TYPES

В этом файле содержатся общесистемные правила преобразования расширений файлов в понятные системе MIME типы данных. Затем уже система выбирает, чем открыть тот или иной тип данных.

15. /ETC/MODPROBE.D/

Папка /etc/modprobe содержит конфигурационные файлы со списками модулей ядра, которые не нужно загружать при старте системы, псевдонимами для существующих модулей, а также позволяет задавать настройки для модулей.

16. /ETC/MODULES-LOAD.D/

Папка /etc/modules-load.d/ содержит файлы со списками модулей, которые должны быть загружены при запуске системы. Имя файла не важно, но он должен иметь расширение .conf.

17. /ETC/NSSWITCH.CONF

Этот файл задает настройки порядка разрешения имен в системе для всех программ, написанных на Си или С++. Например, нужно сначала просматривать локальную сеть и систему, или сразу же отправлять запрос к DNS.

18. /ETC/NTP.CONF

Файл ntp.conf отвечает за настройку службы синхронизации времени - ntpd. В файле указаны адреса ntp серверов, с которых служба будет получать время, а также общие настройки.

19. /ETC/OS-RELEASE

Отображает очень подробную информацию об установленном дистрибутиве:

20. /ETC/PASSWD

Файл содержит список всех зарегистрированных в системе пользователей, а также дополнительные настройки для них, например, оболочку, дату смены пароля и дату отключения аккаунта, кроме самого пароля. Напрямую файл лучше не редактировать, а использовать утилиту для [управления пользователями](https://losst.ru/kak-udalit-polzovatelya-linux) adduser или deluser.

### 

21. /ETC/PROFILE

Файл /etc/profile, точно так же как и /etc/environment загружается и выполняется при запуске любой командной оболочки в системе. Но в отличие от environment, это скрипт, а значит, он может задавать не только переменные, но и выполнять различные команды для инициализации оболочки.

22. /ETC/RESOLV.CONF

В этом файле содержатся IP адреса DNS серверов, которые будет использовать компьютер. В большинстве дистрибутивов вы можете редактировать файл вручную или же использовать специальные утилиты.

23. /ETC/SDDM.CONF

Это конфигурационный файл Linux для настройки менеджера входа sddm, для других менеджеров входа будут свои файлы настройки. Здесь можно изменить максимальный и минимальный ID пользователя, который может войти в систему, например, чтобы разрешить авторизацию root, изменить тему, добавить вход без пароля и многое другое.

24. /ETC/SHADOW

Раньше пароли пользователя содержались в файле /etc/passwd, но поскольку к нему мог получить доступ любой пользователь, это было небезопасно, несмотря на то, что пароли зашифрованы. Поэтому все пароли были вынесены в /etc/shadow. Вы можете [изменить пароль пользователя](https://losst.ru/kak-smenit-parol-v-linux).

25. /ETC/SUDOERS

/etc/sudoers - это файл настройки[прав доступа к утилите sudo](https://losst.ru/nastrojka-sudo-v-linux). Эта утилита позволяет выполнять команды от имени других пользователей, в том числе от имени суперпользователя. Но использовать ее могут только те пользователи, которые прописаны в этом файле.

26. /ETC/SYSCTL.CONF

Этот файл отвечает за [настройку параметров ядра](https://losst.ru/nastrojka-yadra-linux) во время выполнения. Тут вы можете задать все параметры из подсистемы /sys/ и они будут сохранены после перезагрузки.

27. /ETC/VCONSOLE.CONF

У этого файла только одна цель - задать кодировку, раскладку клавиатуры и шрифт по умолчанию для всех виртуальных консолей, запускаемых на машине.

28. /BOOT/GRUB/GRUB.CFG

Этот конфигурационный файл Linux находится не в /etc из-за своего особого предназначения. Здесь содержатся все настройки загрузчика, пункты меню и другие параметры, поэтому он должен быть доступен еще до того как была подключена корневая файловая система.

### 35. Командный интерпретатор bash. Структура команды.

Консоль и терминал обрабатывают команды с помощью программной оболочки.   
Программная оболочка - интерпретатор команд, он распознает команды введенные в командной строке, и запускает программы для выполнения команды.   
В [Ubuntu](https://help.ubuntu.ru/wiki/ubuntu) по умолчанию используется оболочка [bash](https://help.ubuntu.ru/wiki/bash), он распознает команды на языке bash.

Что необходимо знать с самого начала

1. Любой bash-скрипт должен начинаться со строки:  
#!/bin/bash  
в этой строке после #! указывается путь к bash-интерпретатору, поэтому если он у вас установлен в другом месте(где, вы можете узнать набрав whereis bash) поменяйте её на ваш путь.  
2. Комментарии начинаются с символа # (кроме первой строки).  
3. В bash переменные не имеют типа

Может использоваться интерактивно:

user@host:dir$ echo test test

user@host:dir$

Можно писать сценарии:

user@host: nano test.sh

#!/bin/sh

echo test

**Команды**

Команды - это предопределенный набор букв, цифр, символов, которые можно ввести в командной строке и выполнить нажав энтер.

*Команды делятся на два вида:*

-команды встроенные в программную оболочку (например history)

-команды управляющие программами, установленными в системе

Команды для управления программами строятся по такой схеме:

название\_программы -ключ значение

**Название програмы** - это название исполняемого файла из каталогов записанных в переменной $PATH (/bin, /sbin, /usr/bin, /usr/sbin, /usr/local/bin, /usr/local/sbin и др.) или полный путь к исполняемому файлу (/opt/deadbeef/bin/deadbeef)   
**Ключ** - пишется после названия программы, например -h, у каждой программы свой набор ключей, они перечислены в справке к программе, ключи используются для указания какие настройки использовать или какое действие выполнить   
**Значение** - адрес, цифры, текст, спецсимволы (\*, ~, \, &, « », \_ ), переменные ($HOME, $USER, $PATH)

Выполнить команды можно следующим образом:

* набрать команду в командной строке и нажать Enter
* скопировать команду из инструкции и вставить ее в командную строку, затем нажать Enter
* создать скрипт и выполнить двойным нажатием мыши (создать текстовый файл, в первой строке написать #!/bin/bash, ниже написать команды в столбик, сохранить, в свойствах файла разрешить выполнение, нажать два раза по файлу для выполнения всех перечисленных команд)

### 36. Bash. Переменные и типы.

В отличие от большинства других языков программирования, Bash не производит разделения переменных по "типам". По сути, переменные Bash являются строковыми переменными, но, в зависимости от контекста, Bash допускает целочисленную арифметику с переменными. Определяющим фактором здесь служит содержимое переменных.

*Существуют два типа переменных которые можно*

*использовать в скриптах*

• Переменные среды

• Пользовательские переменные

**1)Некоторые переменные среды**

$# - общее количество параметров переданных

скрипту

$\* все аргументы переданные скрипту выводятся в

строку

$@ тоже самое что и предыдущий, но параметры выводятся в столбик

$! – PID последнего запущенного в фоне процесса

$$ - PID самого скрипта

**2)Пользовательские Переменные**

• В дополнение к переменным среды скрипты

позволяют задавать и использовать в сценарии

собственные переменные Подобные переменные

хранят значение до тех пор пока не завершится

выполнение сценария

• Как и в случае с системными переменными к

пользовательским переменным можно обращаться

используя знак доллара

**3)Подстановка команд**

•Одна из самых полезных возможностей

скриптов это возможность извлекать

информацию из вывода команд и назначать

её переменным что позволяет использовать

эту информацию где угодно в файле

сценария

1) mydir=’pwd’

2)mydir=$(pwd)

**4)Математические операции**

•Для выполнения математических операций в

файле скрипта можно использовать

конструкцию вида $((a+b))

var1=$((5+6))

echo $var1

### 37. Bash. Условия.

**Управляющая конструкция if-then**

В некоторых сценариях требуется управлять

потоком исполнения команд. Например, если некое

значение больше пяти нужно выполнить одно

действие в противном случае другое

• Подобное применимо в очень многих ситуациях и

здесь нам поможет управляющая конструкция if-then

В наиболее простом виде она выглядит так:

if команда

then

Команда

else

Команды

fi

Пример скрипта:

#!/bin/bash

User=anotherUser

if grep $User /etc/passwd

then

echo “The user $User exists”

else

echo “The user $User doesn’t exist”

fi

**Сравнение строк**

* + str1=str2 – проверяет строки на равенство, возвращает истину, если они идентичны
  + str1!=str2 – истина, если они не идентичны
  + str1< str2 истина, если 1 меньше, чем 2
  + -n str1 – истина, если длина строки больше нуля
  + -z str1 – истина, если длина строки равна нулю

**Проверки файлов**

* -d <file> – проверяет, существует ли файл и является ли он директорией
* -е <file> - существует ли файл
* -f <file> существует и является ли он файлом
* -r <file> - существует и доступен ли он для чтения
* -s <file> - существует и не является ли он пустым
* -w <file> - существует и доступен ли он для записи
* -x <file> - существует и является ли он исполняемым
* file1 -nt file2 – проверяет, новее ли первый файл в сравнении со вторым
* file1 -ot file2 – проверяет, первый файл старше ли второго
* -O <file> - существует и является ли его владельцем текущий пользователь
* -G <file> - существует и соответствует ли его идентификатор группы идентификатору группы текущего пользователя

Пример программы, которая проверяет наличие существования директории в переменной mydir, если такая директория существует, то выводится ее содержимое, иначе выводится, что такой директории нет:  
#!/bin/bash

Mydir=/home/likegeeks

if [ -d $Mydir ]

then

echo “The $Mydir directory exists”

cd $Mydir

ls

else

echo “The $Mydir directory doesn’t exist”

fi

### 38. Bash. Циклы.

Циклы for

Вот какова базовая структура таких циклов:

for var in list

do

команды

done

В каждой итерации цикла в переменную var будет записываться следующее значение из списка list. В первом проходе цикла, таким образом, будет задействовано первое значение из списка. Во втором — второе, и так далее — до тех пор, пока цикл не дойдёт до последнего элемента.

**Перебор простых значений**

Пожалуй, самый простой пример цикла for в bash-скриптах — это перебор списка простых значений:

#!/bin/bash

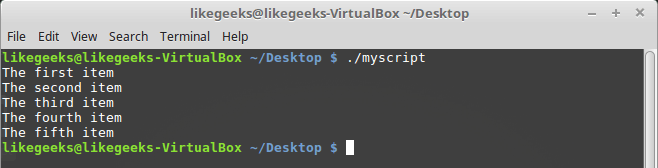
for var in first second third fourth fifth

do

echo The  $var item

done

Результат работы скрипта:



**Перебор сложных значений**

В списке, использованном при инициализации цикла for, могут содержаться не только простые строки, состоящие из одного слова, но и целые фразы, в которые входят несколько слов и знаков препинания. Например, всё это может выглядеть так:

#!/bin/bash

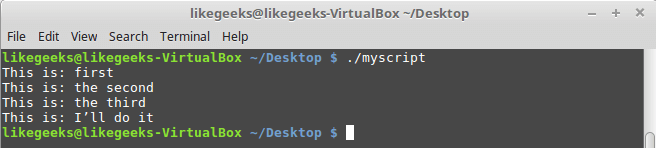
for var in first "the second" "the third" "I’ll do it"

do

echo "This is: $var"

done

Результат:



**Обход файлов, содержащихся в директории**

Один из самых распространённых вариантов использования циклов for в bash-скриптах заключается в обходе файлов, находящихся в некоей директории, и в обработке этих файлов.  
Например, вот как можно вывести список файлов и папок:

#!/bin/bash

for file in /home/likegeeks/\*

do

if [ -d "$file" ]

then

echo "$file is a directory"

elif [ -f "$file" ]

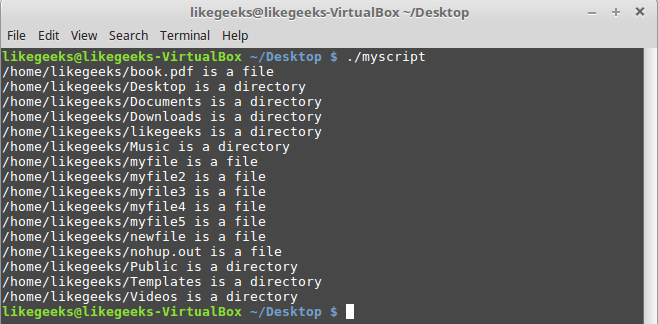
then

echo "$file is a file"

fi

done

Результат:



Обратите внимание на то, как мы инициализируем цикл, а именно — на подстановочный знак «\*» в конце адреса папки. Этот символ можно воспринимать как шаблон, означающий: «все файлы с любыми именами». он позволяет организовать автоматическую подстановку имён файлов, которые соответствуют шаблону.  
  
При проверке условия в операторе if, мы заключаем имя переменной в кавычки. Сделано это потому что имя файла или папки может содержать пробелы.

**Циклы for в стиле C**

В bash-скриптах можно использовать циклы for, описание которых выглядит очень похожим на циклы в стиле C, правда, без некоторых отличий тут не обошлось. Схема цикла при подобном подходе выглядит так:

for (( начальное значение переменной ; условие окончания цикла; изменение переменной ))

На bash это можно написать так:

for (( a = 1*; a < 10; a++ ))*

Пример:

#!/bin/bash

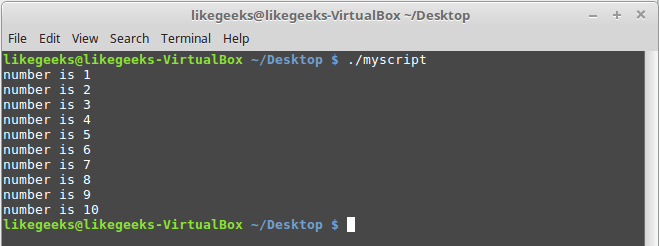
for (( i=1; i <= 10; i++ ))

do

echo "number is $i"

done

Этот код выведет список чисел от 1 до 10.



**Цикл while**

Конструкция for — не единственный способ организации циклов в bash-скриптах. Здесь можно пользоваться и циклами while. В таком цикле можно задать команду проверки некоего условия и выполнять тело цикла до тех пор, пока проверяемое условие возвращает ноль, или сигнал успешного завершения некоей операции. Когда условие цикла вернёт ненулевое значение, что означает ошибку, цикл остановится.  
  
Вот схема организации циклов while  
while команда проверки условия  
do  
другие команды  
done

Пример:

#!/bin/bash

var1=5

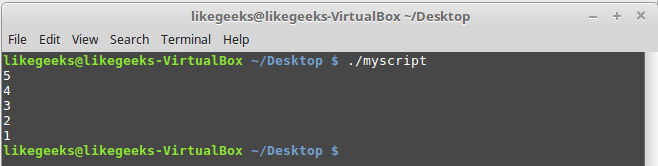
while [ $var1 -gt 0 ]

do

echo $var1

var1=$[ $var1 - 1 ]

done

На входе в цикл проверяется, больше ли нуля переменная $var1. Если это так, выполняется тело цикла, в котором из значения переменной вычитается единица. Так происходит в каждой итерации, при этом мы выводим в консоль значение переменной до его модификации. Как только $var1 примет значение 0, цикл прекращается.  
  


**Вложенные циклы**

В теле цикла можно использовать любые команды, в том числе — запускать другие циклы. Такие конструкции называют вложенными циклами:

#!/bin/bash

for (( a = 1; a <= 3; a++ ))

do

echo "Start $a:"

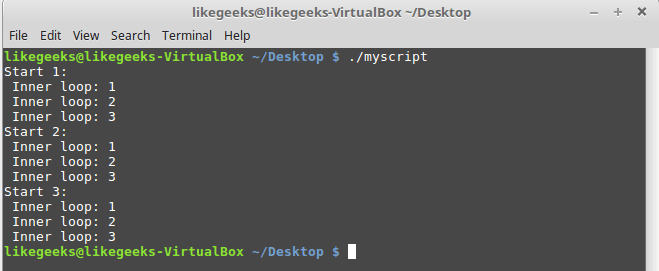
for (( b = 1; b <= 3; b++ ))

do

echo " Inner loop: $b"

done

done

Ниже показано то, что выведет этот скрипт. Как видно, сначала выполняется первая итерация внешнего цикла, потом — три итерации внутреннего, после его завершения снова в дело вступает внешний цикл, потом опять — внутренний.  
  
  
*Вложенные циклы*

Управление циклами

Возможно, после входа в цикл, нужно будет остановить его при достижении переменной цикла определённого значения, которое не соответствует изначально заданному условию окончания цикла. В подобных случаях пригодятся следующие две команды:  
 =break

=continue

Команда break

Эта команда позволяет прервать выполнение цикла. Её можно использовать и для циклов for, и для циклов while:

#!/bin/bash

for var1 in 1 2 3 4 5 6 7 8 9 10

do

if [ $var1 -eq 5 ]

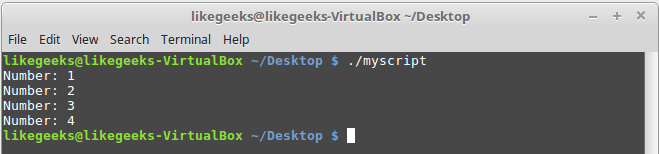
then

break

fi

echo "Number: $var1"

done

Такой цикл, в обычных условиях, пройдется по всему списку значений из списка. Однако, в нашем случае, его выполнение будет прервано, когда переменная $var1 будет равна 5.  
  
  
*Досрочный выход из цикла for*  
  
Вот — то же самое, но уже для цикла while:

#!/bin/bash

var1=1

while [ $var1 -lt 10 ]

do

if [ $var1 -eq 5 ]

then

break

fi

echo "Iteration: $var1"

var1=$(( $var1 + 1 ))

done

Команда break, исполненная, когда значение $var1 станет равно 5, прерывает цикл. В консоль выведется то же самое, что и в предыдущем примере.

Команда continue

Когда в теле цикла встречается эта команда, текущая итерация завершается досрочно и начинается следующая, при этом выхода из цикла не происходит. Посмотрим на команду continue в цикле for:

#!/bin/bash

for (( var1 = 1; var1 < 15; var1++ ))

do

if [ $var1 -gt 5 ] && [ $var1 -lt 10 ]

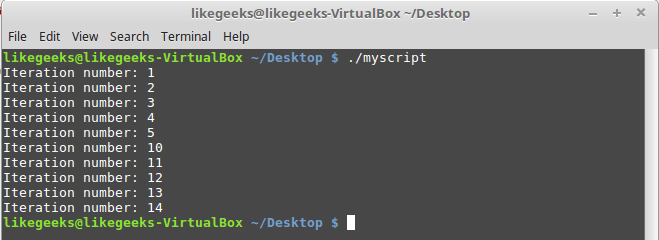
then

continue

fi

echo "Iteration number: $var1"

done

Когда условие внутри цикла выполняется, то есть, когда $var1 больше 5 и меньше 10, оболочка исполняет команду continue. Это приводит к пропуску оставшихся в теле цикла команд и переходу к следующей итерации.  
  


### 39. Bash. Функции.

* **Функции bash** — это именованные блоки кода, которые можно повторно использовать в скриптах.
* Функцию можно объявить:

functionName {

}

или

functionName() {

}

* Для вызова функции достаточно указать её имя.
* Функцию можно вызывать столько раз, сколько нужно.

!Внимание: попытавшись использовать функцию до её объявления, вы столкнётесь с ошибкой.

* **Имена для функций должны быть уникальными**, иначе проблем не избежать. Если вы переопределите ранее объявленную функцию, новая функция будет вызываться вместо старой без каких-либо уведомлений или сообщений об ошибках.
* **Команда return** позволяет задавать возвращаемый функцией целочисленный код завершения (значение хранится в переменной $?). Максимальное число, которое может вернуть команда return — 255.

Есть два способа работы с тем, что является результатом вызова функции. Приведем их примеры на след. скриптах:

**(1).**

#!/bin/bash

function way1 {

val=5

return $(( $val + 10 ))

}

way1

echo "$?"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Выведется число 5.

**(2).**

#!/bin/bash

function way2 {

val=5

echo $(( $val + 10 ))

}

vivod=$( way2 )

echo "$vivod"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Выведется число 5.

И функция way1 и функция way2 добавляют 10 к числу, которое содержится в переменной $val, значение которой мы задали вначале функции. Затем они возвращают результат, используя команду return.

В **(1)** способе то, что возвратила функция, выводится командой echo с использованием переменной $? (хранит код возврата).

!Но: если вы выполните любую другую команду до извлечения из переменной $? значения, возвращённого функцией, это значение будет утеряно.

А во **(2)** способе результат функции записывается в переменную $vivod, которая затем выводится командой echo. Такой подход позволяет обойти ограничения команды return и возвращать из функции любые данные.

* В функциях существует два вида переменных: **глобальные** и **локальные**.
* **По умолчанию** все объявленные в скриптах переменные **глобальны**. Так, к переменным, объявленным за пределами функций, можно без проблем обращаться из функций, а к переменным, объявленным в функциях, можно обращаться и в основном коде скрипта после вызова функций.
* Переменные, которые объявляют и используют внутри функции, могут быть объявлены **локальными**. Для того, чтобы это сделать, используется ключевое слово **local** перед именем переменной. Если за пределами функции есть переменная с таким же именем, это на неё не повлияет. Ключевое слово **local** позволяет отделить переменные, используемые внутри функции, от остальных переменных.
* Функцию можно вызвать как **без аргументов** (как в предыдущих примерах), так и **с аргументами**. Аргументы передают функции, записывая их после её имени. Функции могут использовать стандартные позиционные параметры, в которые записывается то, что передаётся им при вызове. Например, имя функции хранится в параметре $0, первый переданный ей аргумент — в $1, второй — в $2, и так далее. Количество переданных функции аргументов можно узнать, обратившись к переменной $#.

!Внимание: функция не может напрямую работать с параметрами, которые переданы скрипту при его запуске из командной строки. Вместо этого, если в функции планируется использовать параметры, переданные скрипту при вызове из командной строки, надо передать их ей при вызове.

Пример:

#!/bin/bash

function srzn {

if [ $# -eq 0 ]; then

sum=0

for ar in “$@”

do

sum=$(( $sum + $ar ))

done

sred=$(( $sum / 2 ))

echo “Srednee znachenie peredannyh fynkcii parametrov = $sred”

else

echo “Parametry ne naideny”

fi

}

srzn 1 2 3 4 5 // используются параметры, переданные функции

srzn

srzn $1 $2 $3 // используются параметры, переданные скрипту при вызове из командной строки

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Запуск скрипта (пусть называется srednee.sh): ./srednee.sh 20 10 6

Что выведет: Srednee znachenie peredannyh fynkcii parametrov = 3

Parametry ne naideny

Srednee znachenie peredannyh fynkcii parametrov = 12

### 40. Переменные окружения в bash.

* Когда вы работаете в bash, ваше **окружение состоит из многих параметров**, таких как вид приглашения, домашняя директория, рабочая директория, имя интерпретатора, открытые вами файлы, определенные вами функции и так далее.
* Ваше окружение включает множество **переменных**, которые могут определяться как интерпретатором, так и вами.
* В bash также можно использовать **переменные командной оболочки**, которые можно экспортировать в ваше окружение для использования другими процессами, запущенными в командном интерпретаторе, или другими командными интерпретаторами, которые вы можете вызывать из текущего интерпретатора.
* Как переменные окружения, так и переменные командной оболочки имеют имена. Для ссылки на **значение переменной** необходимо поставить перед ее именем знак доллара ('**$**').
* **Основные** переменные окружения bash:

| **Имя** | **Функция** |
| --- | --- |
| **USER** | Имя пользователя, вошедшего в систему |
| **UID** | Числовой идентификатор пользователя, вошедшего в систему |
| **HOME** | Домашняя директория пользователя |
| **PWD** | Текущая рабочая директория |
| **SHELL** | Имя командного интерпретатора |
| **$** | Идентификатор процесса (PID) запущенного интерпретатора bash -(или другого процесса) |
| **PPID** | Идентификатор процесса, породившего данный процесс (т. е. идентификатор родительского процесса) |
| **?** | Код завершения последней команды |

* Вы можете создать или задать значение переменной командной оболочки, поставив знак равенства (**=**) сразу же после ее имени. Если переменная существует, то она получит новое значение.
* Имена переменных чувствительны к **регистру**.
* При обсуждении использования кавычек мы упоминали, что можно использовать как одинарные, таки двойные кавычки. Между ними существует важное отличие. Интерпретатор подставляет значения переменных командной оболочки, если они заключены в двойные кавычки ($quot;), но не делает этого при использовании одинарных кавычек (').
* **Команда env**, запущенная без каких-либо опций или параметров, выводит текущие значения переменных окружения.
* Вы можете использовать **команду unset** для сброса переменной и удаления ее из списка переменных командной оболочки. Если переменная была экспортирована в окружение, то она также будет удалена и из окружения.
* При помощи **команды set** вы можете управлять многими аспектами работы bash (или других интерпретаторов). Если вы используете команду **set** без каких-либо опций, то она выводит список всех переменных командной оболочки и их значения (если они существуют).
* **Команда exec** запускает другую команду, которая замещает текущий интерпретатор.

### 41. Основные команды Linux для управления пользователями и группами.

* **useradd** или **adduser** - добавить нового пользователя.
* **passwd** - задать пароль для пользователя.
* **usermod** - изменить параметры учетной записи пользователя.
* **userdel** или **deluser** - удалить учетную запись пользователя.
* **groupadd** - добавляет новую группу.
* **gpasswd** - устанавливает пароль группы.
* **groupmod** - изменение параметров группы.
* **groupdel** - удаление группы.

### 42. Суперпользователь root. Характеристика, особенности, функции, опасность.

Чтобы запускать административные задачи на Linux, у вас должны быть права суперпользователя (права root).

**Характеристика:**

**root**  или **суперпользователь** — это специальный аккаунт в UNIX-подобных системах с идентификатором (UID, User IDentifier) 0, владелец которого имеет право на выполнение **всех** без исключения операций.(в отличии от обычного пользователя).

**Особенности:** пользователь root имеет права на выполнение абсолютно всех команд и др.

**Функции:**

* Выполнение различных действий от имени суперпользователя.
* Администрирование системы.

**Опасность:**

1. Безоговорочное выполнение всех действий от имени root, что может привести к неисправным последствиям. (Вы безо всяких предупреждений со стороны системы сможете выполнить любую операцию, в частности, удалить системные файлы, сделав при этом систему неработоспособной.)
2. Будьте осторожны при выполнении административных задач -- вы можете испортить вашу систему!
3. Как-то ограничить свободу действий root невозможно.

### 43. Система прав доступа Linux. Структура и смысл прав доступа.

Для каждого объекта в файловой системе Linux существует набор прав доступа, определяющий взаимодействие пользователя с этим объектом. (это могут быть файлы, каталоги и др.)

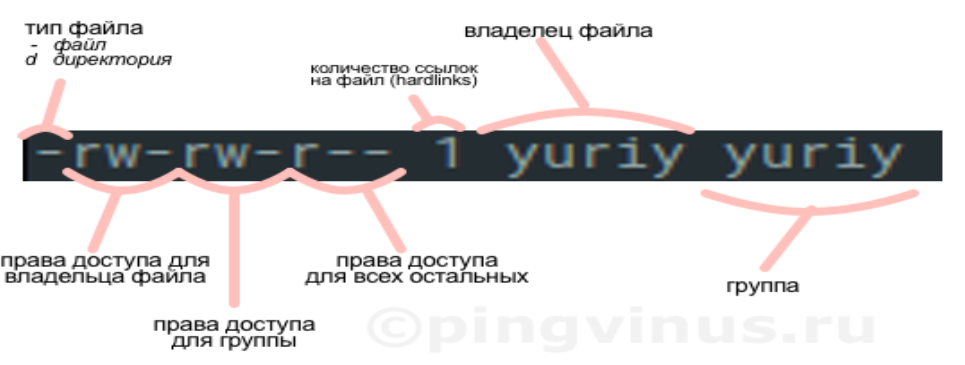
У каждого объекта в Linux имеется владелец, следовательно, права доступа применяются относительно владельца файла. Они состоят из набора 3 групп по три атрибута:

● чтение( r ), запись( w ), выполнение( x ) для владельца;

● чтение, запись, выполнение для группы владельца;

● чтение, запись, выполнение для всех остальных.





Права доступа позволяют разграничить полномочия пользователей, работающих в системе. В частности, права доступа позволяют отдельным пользователям иметь «личные» файлы и каталоги.

### 44. Организация хранения паролей пользователей в Linux.

Вся информация о пользователе обычно хранится в файлах /etc/passwd и /etc/group.

/etc/passwd – этот файл содержит информацию о пользователях. Запись для каждого пользователя занимает одну строку:

root: 12345$passwd: 0 : 0 : root : /root : /bin/bash





имя пользователя – имя, используемое пользователем на все приглашения типа login при аутентификации в системе.

зашифрованный пароль – обычно хешированный по необратимому алгоритму MD5 пароль пользователя или символ '!', в случаях, когда интерактивный вход пользователя в систему запрещен.

UID – числовой идентификатор пользователя. Система использует его для распределения прав файлам и процессам.

GID – числовой идентификатор группы. Имена групп расположены в файле /etc/group. Система использует его для распределения прав файлам и процессам.

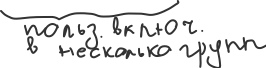
Настоящее имя пользователя – используется в административных целях, а также командами типа finger (получение информации о пользователе через сеть).

Домашний каталог – полный путь к домашнему каталогу пользователя.

Оболочка – командная оболочка, которую использует пользователь при сеансе. Для нормальной работы она должна быть указана в файле регистрации оболочек /etc/shells.

/etc/group – этот файл содержит информацию о группах, к которым принадлежат пользователи:

Project: 12$75w: 100: root, bin, daemon



Имя группы – имя, применяемое для удобства использования таких программ, как newgrp .

Шифрованный пароль – используется при смене группы командой newgrp. Пароль для групп может отсутствовать.

GID – числовой идентификатор группы. Система использует его для распределения прав файлам и процессам.

Пользователи, включенные в несколько групп – В этом поле через запятую отображаются те пользователи, у которых по умолчанию (в файле /etc/passwd ) назначена другая группа.

### 45. Основные команды Linux для управления правами доступа.

Для изменения прав используется команда chmod, в качестве параметра может принимать символьное и числовое представление атрибутов.

Символьный вид:

chmod |u/g/o/a| |+/-/= | |r/w/x/X/u/g/o| filename

где: u,g,o,a – установка прав для пользователя, группы, остальных пользователей, всех групп прав доступа соответственно.

+,-,= – добавить, удалить, установить разрешение соответственно.

r,w,x,X,u,g,o – право чтения, записи, выполнения, выполнения если есть такое право еще у какой либо из групп доступа, такие же как у владельца, такие же как у группы, такие же как у остальных пользователей.

filename - Имя файла, у которого изменяются права.

Для использования числового вида необходимо представить права доступа к файлу в виде 3-х двоичных групп. Так например:

rwx r-x r-- будет выглядеть как: 111 101 100

Теперь каждую двоичную группу перевести в 8-ричное число:

111 – 7, 101 – 5, 100 – 4, то есть права определяются как сумма цифр 4 (чтение), 2 (запись) и 1 (исполнение).

Чтобы задать файлу такие права необходимо выполнить команду:

chmod 754 filename.

Команда chown – позволяет сменить владельца файла. Синтаксис команды:

chown username:groupname filename,

где username – имя пользователя – нового владельца файла; groupname – имя группы – нового владельца файла; filename – имя файла, у которого сменяется владелец.

Команда chgrp используется для изменения владельца-группы файла. Синтаксис команды:

chgrp groupname filename

где: groupname – имя группы, которой будет принадлежать файл filename – имя изменяемого файла

Использовать команды chown и chmod может только пользователь- владелец файла и root , а команду chgrp - пользователь-владелец файла, группа- владелец файла и root .

Существует особое право под названием ***sticky bit*** используется в основном для каталогов, из такого каталога пользователь может удалить только те файлы, владельцем которых он является. Примером может служить каталог /tmp, в который запись открыта для всех пользователей, но нежелательно удаление чужих файлов. Установка атрибута производится следующим образом :

chmod +t filename

### 46. Понятие компьютерных сетей. Общие принципы, организация.

Компьютерная сеть — совокупность компьютеров, соединенных с помощью каналов связи и средств коммутации в единую систему для обмена сообщениями и доступа пользователей к программным, техническим, информационным и организационным ресурсам сети. Сети предоставляют пользователям возможность не только быстрого обмена информацией, но и совместной работы на принтерах и других периферийных устройствах, и даже одновременной обработки документов.

Все многообразие компьютерных сетей можно классифицировать по группе признаков:

- территориальная распространенность;

- ведомственная принадлежность;

- скорость передачи информации;

- тип среды передачи;

- способ организации сети.

По территориальной распространенности сети могут быть *локальными, глобальными, региональными и городскими.*

***Городская сеть*** (MAN - Metropolitan Area NetWork) - сеть, которая обслуживает информационные потребности большого города.

***Региональные*** - расположенные на территории города или области.

Сети, в пределах одного здания, объединяющие от 2 до 300 компьютеров, которые принадлежат обычно одной организации (или одной семье), называются ***локальными*** вычислительными сетями.

***Глобальные*** сети - WAN (Worldwide Area Network) объединяют сотни, тысячи узлов во многих странах мира.

По принадлежности различают ведомственные и государственные сети. Ведомственные принадлежат одной организации и располагаются на ее территории. Государственные сети используются в государственных структурах.

По скорости передачи информации компьютерные сети делятся на:

низкоскоростные (до 10 Мбит/с),

среднескоростные (до 100 Мбит/с),

высокоскоростные (свыше 100 Мбит/с);

По типу среды передачи разделяются на сети коаксиальные, на витой паре, оптоволоконные, с передачей информации по радиоканалам, в инфракрасном диапазоне. Компьютеры могут соединяться кабелями, образуя paj-личную топологию сети (звездная, шинная, кольцевая и др.).

По способу организации сети подразделяются на *реальные и искусственные.*

***Искусственные сети*** (псевдосети) позволяют связывать компьютеры вместе через последовательные или параллельные порты и не нуждаются в дополнительных устройствах.

***Реальные сети*** позволяют связывать компьютеры с помощью специальных устройств коммутации и физической среда передачи данных.

### 47. Понятие сетевого ресурса. Классификация.

Сетевой ресурс - это устройство или часть информации, к которой может быть осуществлён удалённый доступ с другого [компьютера](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), обычно через [локальную компьютерную сеть](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%BA%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C) или посредством корпоративного [интернета](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82), как если бы ресурс находился на локальной машине.

Ресурсы бывают трех типов :

1. Аппаратные
2. Программные
3. Информационные

**Аппаратные ресурсы.** Означает, что определенные ресурсы компьютера, например, диск, принтер или дисковод CDROM устанавливаются таким образом, чтобы все машины в сети могли пользоваться ими.

**Программные ресурсы.** Помещение программ на общий диск, взамен хранения копий на каждом персональном компьютере. Это касается программ, которые занимают много места и с которыми работают многие пользователи данной сети. Например, программы 1С: Предприятие, КонсюльтантПлюс, Гарант, АудитЭксперт.

**Информационные ресурсы.** Сеть позволяет пользоваться общей информацией совместно с другими пользователями сети. В зависимости от того, каким образом установлена сеть, существуют два способа доступа к информации. Прямой способ - передать файл с одного компьютера на другой. Косвенный способ - отправить файл на некоторый промежуточный пункт, где информация будет находиться до тех пор, пока ее не заберет адресат. Примером разделения информационного ресурса является использование Интернета.

### 48. Семейство сетевых протоколов TCP/IP.

| Прикладной уровень | Прикладной уровень |
| --- | --- |
| Представительский уровень |
| Сеансовый уровень |
| Транспортный уровень | Транспортный уровень |
| Сетевой уровень | Сетевой уровень |
| Канальный уровень | Уровень доступа к сети |
| Физический уровень |

TCP/IP – это набор протоколов, который задает стандарты связи между компьютерами и содержит подробные соглашения о маршрутизации и межсетевом взаимодействии.

**ARP**

ARP (Address Resolution Protocol) — протокол сетевого уровня, предназначенный для преобразования IP-адресов (адресов сетевого уровня) в MAC-адреса (адреса канального уровня) в сетях TCP/IP. ARP-таблица отображает IP и MAC подключенных к серверу сетевых устройств.

Если с помощью ARP-таблицы не удается сразу осуществить преобразование адресов, то IP-пакет ставится в очередь, а необходимая для преобразования информация получается с помощью запросов и ответов протокола ARP, после чего IP-пакет передается по назначению. Если в сети нет машины с искомым IP-адресом, то ARP-ответа не будет и не будет записи в ARP-таблице. Протокол IP будет уничтожать IP-пакеты, направляемые по этому адресу. Протоколы верхнего уровня не могут отличить случай повреждения сети Ethernet от случая отсутствия машины с искомым IP-адресом.

**IP**

Internet Protocol — маршрутизируемый протокол сетевого уровня стека TCP/IP.

IP объединяет сегменты сети в единую сеть, обеспечивая доставку пакетов данных между любыми узлами сети через произвольное число промежуточных узлов (маршрутизаторов). IP не гарантирует надёжной доставки пакета до адресата. Гарантию безошибочной доставки пакетов дают некоторые протоколы более высокого уровня.

В современной сети Интернет используется IP четвёртой версии, также известный как IPv4. В протоколе IP этой версии каждому узлу сети ставится в соответствие IP-адрес длиной 4 октета (4 байта). При этом компьютеры в подсетях объединяются общими начальными битами адреса. Количество этих бит, общее для данной подсети, называется маской подсети.

Pv6 (Internet Protocol version 6) — новая версия интернет протокола (IP), призванная решить проблемы, с которыми столкнулась предыдущая версия (IPv4) при её использовании в Интернете, за счёт использования длины адреса 128 бит вместо 32. После того как адресное пространство в IPv4 закончится, два стека протоколов — IPv6 и IPv4 — будут использоваться параллельно с постепенным увеличением доли трафика.

**UDP**

Протокол UDP (User Datagram Protocol - протокол пользовательских обеспечивает ненадежную доставку датаграмм и не поддерживает соединений из конца в конец.

Размер каждого записанного сообщения будет совпадать с размером каждого прочитанного. Протокол UDP никогда не объединяет несколько сообщений в одно и не делит одно сообщение на части.

**TCP**

Протокол TCP предоставляет транспортные услуги, отличающиеся от услуг UDP. Вместо ненадежной доставки датаграмм без установления соединений, он обеспечивает гарантированную доставку с установлением соединений в виде байтовых потоков.

Канал является дуплексным. Один прикладной процесс пишет данные в TCP-порт, они проходят по сети, и другой прикладной процесс читает их из своего TCP-порта.

Протокол TCP разбивает поток байт на пакеты; он не сохраняет границ между записями. Не существует зависимости между числом и размером записываемых сообщений с одной стороны и числом и размером считываемых сообщений с другой стороны.

Протокол TCP требует, чтобы все отправленные данные были подтверждены принявшей их стороной. Он использует таймауты и повторные передачи для обеспечения надежной доставки.

**TELNET**

TELNET (сокр. от англ. teletype network) — сетевой протокол для реализации текстового терминального интерфейса по сети.

Протокол TELNET позволяет обслуживающей машине рассматривать все удаленные терминалы как стандартные "сетевые виртуальные терминалы" строчного типа, работающие в коде ASCII, а также обеспечивает возможность согласования более сложных функций (например, локальный или удаленный эхо-контроль, страничный режим, высота и ширина экрана и т.д.) TELNET работает на базе протокола TCP.

**FTP**

FTP (англ. File Transfer Protocol) — протокол передачи файлов по сети. С его помощью можно подключаться к FTP-серверам, просматривать содержимое их каталогов и загружать файлы с сервера или на сервер.

Для доступа к серверу или удаленному компьютеру по протоколу FTP используют специальные программы, которые называются FTP-клиентами.

**SMTP**

SMTP (Simple Mail Transfer Protocol — простой протокол передачи почты) — это широко используемый сетевой протокол, предназначенный для передачи электронной почты в сетях TCP/IP.

SMTP-сервер выполняет две функции:

- Проверяет правильность настроек и выдает разрешение компьютеру, пытающемуся отправить email-сообщение;

- Отправляет исходящее сообщение на указанный адрес и удостоверяется в успешной доставке сообщения. Если его невозможно доставить, отправителю направляется сообщение об этом.

Основные параметры, которые передаются серверу соответствующими командами:

- MAIL FROM — электронный адрес отправителя;

- RCPT TO — электронный адрес получателя;

- DATA — заголовок и тело письма.

### 49. Адресация компьютеров в сети. IP, MAC адреса.

Для осуществления обмена сообщениями между компьютера каждый компьютер в сети должен иметь свой уникальный адрес. На практике получили распространение три способа адресации узлов.

**Аппаратные адреса**. Эти адреса назначаются производителями коммуникационного оборудования и являются уникальными. Аппаратные адреса используется в сети небольшого размера. При замене оборудования изменяются также адреса компьютеров.

**Символьные адреса или имена.** Эти адреса назначаются людьми. Символьный адрес легко использовать как в небольших, так и в крупных сетях. Для работы в больших сетях символьное имя может иметь сложную иерархическую структуру, например org.fa.ru. Еще символьные адреса называют DNS-именем т.к. для последующей ассоциации символов и IP-адреса используется DNS-сервера (Domain name servers). В качестве примеров можно привести DNS от CloudFlare 1.1.1.1 и Google Publc DNS: 8.8.8.8.

**Числовые составные адреса**. Из-за переменного формата и потенциально большой длины, их передача по сети не очень экономична. В больших сетях в качестве адресов узлов используют числовые составные адреса фиксированного и компактного форматов. В них поддерживается двухуровневая иерархия, адрес делится на старшую часть - номер подсети и младшую - номер узла.

**IP-адрес** является основным видом адресации в Internet. Он обозначает не только компьютер, но и сегмент сети, в котором находится данный компьютер. IP-адрес - представляет собой 32-разрядное двоичное число, которое разбивается на четыре восьмиразрядных поля, называемых октетами. TCP/IP представляет эти двоичные октеты их десятичными эквивалентами

**MAC-адрес**, который также называют физическим адресом, Ethernet-адресом, присваивается каждому сетевому адаптеру при его производстве. Его размер - 6 байт.

Этот сетевой адрес является уникальным, - фирмам-производителям выделены списки адресов, в рамках которых они обязаны выпускать карты. Адрес записывается в виде шести групп шестнадцатеричных цифр по две в каждой (шестнадцатеричная запись байта). Первые три байта называются префиксом (определяет 224 различных комбинаций). MAC-адреса используются на физическом и канальном уровнях

### 50. Удаленный доступ к командной строке. Протокол SSH.

Для взаимодействия с удаленными UNIX-подобными машинами наиболее часто применяется протокол SSH — сетевой протокол прикладного уровня, позволяющий производить удалённое управление операционной системой и туннелирование TCP-соединений (например, для передачи файлов). Схож по функциональности с протоколом Telnet но, в отличие от него, шифрует весь трафик, включая и передаваемые пароли. SSH допускает выбор различных алгоритмов шифрования. SSH-клиенты и SSH-серверы доступны для большинства ОС.

По-умолчанию, в большинстве UNIX-подобных ОС ssh-клиенты уже установлены из коробки. В Windows необходимо установить дополнительно SSH-клиент (например, Putty), но в некоторых реализациях Windows 10 есть встроенный ssh-клиент, доступный из под CMD (появился совсем недавно).

SSH-сервер по-умолчанию может не предоставляться, но он доступен в стандартных репозиториях ubuntu/debian-подобных дистрибутивов.

Установить его можно с помощью команды:

sudo apt install openssh-server

При работе с SSH возможно 2 способа авторизации на удаленной машине:

1. С помощью пароля
2. С помощью закрытого ключа

Для авторизации с помощью ключа необходимо его сгенерировать с помощью команды

ssh-keygen

Синтаксис команды для подключения к удаленной машине:

ssh логин@ip\_адрес

Если же порт по-умолчанию не **22**, то его также необходимо указать:

ssh -p порт логин@ip\_адрес

Если же авторизация пользователя происходит по связке открытого/закрытого ключа, то также необходимо передать файл ключа с помощью -i

ssh -p порт -i файл\_ключа логин@ip\_адрес

**Также есть возможность передачи файлов по протоколу ssh.**

Для этого есть специальная команда scp:

scp локальный\_файл логин@ip\_адрес:/директория\_куда\_копировать/

Если надо передать папку (но лучше ее сжать в .zip, передать и на сервере раззиповать, быстрее будет), то необходимо указать флаг -r

scp -r локальная папка логин@ip\_адрес:/директория\_куда\_копировать/

Можно пойти дальше и копировать данные с одного сервера на другой с помощью хост-машины:

scp -r логин1@ip1:/home/чтото логин2@ip2:/home/чтототам