Подготовка к экзамену!!!

1. Система контроля версий GIT

1 Билет: Git. Основные понятия и определения:

Git — распределённая система управления версиями. Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки.

Версия — это состояние файла (или нескольких файлов) в какой-то конкретный момент времени. Например, пустой файл (1), тот же файл с каким-то текстом (2) и этот же файл, в котором была исправлена опечатка (3) — три разные версии одного файла, которые были получены последовательной модификацией (изменением) файла.

Система управления версиями — программа, позволяющая сохранять состояние файлов (те самые версии), возвращаться к ранее сохраненному состоянию, сохранять последовательность изменений внесенных в файлы, отменять или заново применять эти изменения, отслеживать авторство изменений.

Разделение версий — независимые изменения одного файла.

Слияние версий — объединение двух и более независимых версий. Для примера выше, слиянием будет объединение двух существующих версий нашего файла в одну — файл, в котором будет и новый текст, и исправленная опечатка.

История разработки — совокупность всех версий файлов, над которыми ведется работа. Историей разработки в данном случае будет список изменений:

- создание файла

- добавление изначального текста
- исправление опечатки
- добавление нового текста
- объединение двух версий файла (при выполнении слияния)

Нелинейная история — история, в которой изменения вносятся не одно за другим последовательно, а может быть внесено несколько независимых изменений на основе одной версии файла (исправление опечатки и добавление нового текста). Т.е. мы создаем две параллельные истории изменений файла.

Репозиторий (repository) — совокупность файлов, состояние которых отслеживается, и история их изменений. По факту, репозиторий — это проект, над которым ведется работа, и все изменения в этом проекте. Для отслеживания состояния файла его необходимо добавить в репозиторий.

Коммит (commit) — сохраненное состояние (версия) файлов репозитория.

Ветка (branch) — последовательность коммитов (история изменения состояния репозитория). Каждый коммит в ветке имеет «родителя» (*parent commit*) — коммит, на основе которого был получен текущий. В репозитории может быть несколько веток (в случаях, когда к одной версии репозитория применяется несколько независимых изменений).

HEAD — указатель на текущий коммит (указатель на состояние, в котором репозиторий находится на данный момент).

Мастер (master, main) — основная ветка репозитория, создается автоматически при создании репозитория.

Мердж (слияние, merge) — объединение двух или более веток. В процессе мерджа изменения с указанной ветки переносятся (копируются) в текущую.

Целевая ветка мерджа — ветка, изменения с которой объединяются с текущей веткой.

База слияния (merge base) — последний общий коммит двух веток.

Мердж коммит (merge commit) — коммит, который создается автоматически по завершению процесса слияния веток. Мердж коммит содержит в себе все изменения целевой ветки мерджа, которые отсутствуют в текущей (все коммиты целевой ветки, которые начиная с базы слияния, но не включая её).

Слияние перемоткой (fast-forward merge) — слияние веток, при котором в текущей ветке отсутствуют новые коммиты (последний коммит текущей ветки является базой слияния). При таком мердже текущая ветка просто переходит в состояние целевой ветки (указатель *HEAD* переносится на последний коммит целевой ветки). Мердж коммит при этом не создается.

Слияние без перемотки (non fast-forward merge) — слияние, при котором новые коммиты (относительно базы слияния) присутствуют как в текущей, так и в целевой ветках.

2 и 3 Билеты: В чем заключается экономия времени при использовании системы контроля версий? Системы контроля версий. Преимущества и недостатки

Системы контроля версий (СКВ или VCS) разработаны специально для того, чтобы максимально упростить и упорядочить работу над проектом (вне

зависимости от того, сколько человек в этом участвуют). СКВ дает возможность видеть, кто, когда и какие изменения вносил; позволяет формировать новые ветви проекта, объединять уже имеющиеся; настраивать контроль доступа к проекту; осуществлять откат до предыдущих версий.

Типы системы контроля версий: ***Локальный СКВ*** - Данная система контроля версий хранит и все зарегистрированные в ней файлы, и любые вносимые в них изменения. Специально к текстовым файлам применяется алгоритм дельта-компрессии. Это означает, что система сохраняет последнюю версию и все предшествующие ей изменения; ***Централизованные СКВ*** - Для привлечения к работе над одним проектом нескольких специалистов были разработаны централизованные системы контроля версий, такие как, к примеру, CVS, Subversion, Perforce. Они имеют собственный центральный сервер, где накапливаются все файлы проекта. Система контролирует и их сохранность, и выдачу их копий определенному ряду пользователей. Именно по такой схеме много лет и работали СКВ; ***Распределенные СКВ*** - суть их работы состоит в выгрузке клиентам не только версий файлов с последними изменениями, а всего репозитория.

2. Linux

4 Билет: Операционные системы на ядре Linux. Особенности файловой системы. Принципы построения и функционирования. Архитектура операционной системы

В ОС Linux все файлы организованы в каталоги, которые, в свою очередь, иерархически соединены друг с другом, образуя одну общую файловую структуру. При обращении к файлу необходимо указывать не только его имя, но и место, которое он занимает в этой файловой структуре. Можно создавать

любое количество новых каталогов, добавляя их к файловой структуре. Команды для работы с файлами:

find; cp; mv; ln, - позволяет находить файлы, копировать их и перемещать из одного каталога в другой. В совокупности все эти элементы и образуют файловую структуру ОС Linux.

Имя файла может содержать любые буквы, знаки подчеркивания и цифры. Можно включать в имена точки и запятые. При этом имя файла не должно начинаться ни с цифры, ни с точки (за исключением некоторых особых случаев). Символы: / \ ? * (косая черта, вопросительный знак, звездочка), – зарезервированы в системе в качестве специальных и в именах файлов использоваться не должны. Максимальная длина имени файла 256 символов. Имя файла может включать и расширение. Для отделения расширения от собственно имени файла служит точка.

Файлы в ОС Linux организованы в иерархическую систему каталогов. Из-за сходства такую структуру называют древовидной структурой. Если быть более точным, то эта структура скорее похожа на куст, перевернутый вверх ногами. Ствола здесь нет. Вниз от корня отходят ветви. Каждая ветвь отходит только от одной, а от нее самой может отходить множество ветвей до самого нижнего уровня. В этом смысле файловую структуру скорее, можно назвать «родители-потомки». Аналогичным образом любой каталог является подкаталогом другого каталога, каждый подкаталог может содержать множество подкаталогов, но сам должен быть потомком только одного родительского каталога.

Основными архитектурными компонентами Linux является ядро, shell, X Windows.

Ядро (*kernel*) – это важнейшая часть Linux, как и любой другой ОС, поскольку именно ядро обеспечивает взаимодействие с аппаратной частью компьютера. Когда загружается какое-то приложение с жесткого диска в оперативную память, или происходит переключение между уже работающими

приложениями, или тогда, когда какое-то приложение записывает информацию в файл на диске, ОС или активное приложение должно запросить доступ к той части аппаратуры, которая ему необходима. Ядро обеспечивает исполнение таких запросов других частей ОС и приложений, а также распределяет память между запускаемыми приложениями. Ядро, таким образом, является посредником между аппаратным и программным обеспечением компьютера, обеспечивающим их взаимодействие.

Ядро Linux состоит из модулей. Модули ядра позволяют обеспечивать ядру новые функции без его рекомпиляции.

Существенное отличие Linux от некоторых других операционных систем состоит в способности устанавливать или удалять поддержку оборудования, файловых систем, языков и т.п. даже без перезагрузки системы.За взаимодействие с пользователем в Linux отвечают специальные программы. Существует два вида таких программ – оболочка shell для работы в текстовом режиме (интерфейс командной строки) и графический интерфейс пользователя (GUI), организующий взаимодействие с пользователем в графическом режиме. Причем любая программа в Linux может быть запущена как через оболочку (если запущен X-сервер), так и через графический интерфейс пользователя (GUI). Запуск программ из оболочки эквивалентен двойному щелчку мышкой по иконке программы в GUI. Некоторые программы не приспособлены для запуска через GUI, так что можно сказать, что существуют программы, предназначенные для исполнения только из командной строки (через оболочку).Shell представляет собой некий аналог командному процессору command.com в MS DOS, или интерпретатору команд.

Собственно говоря, сама по себе, если ее рассматривать как отдельную программу, оболочка *shell*– вещь довольно таки бесполезная, потому что она не может ничего делать. Но она обеспечивает нахождение вызываемых программ, запуск их на выполнение и организацию ввода/вывода. Кроме того,

оболочка обеспечивает работу с переменными окружения и выполняет некоторые преобразования (подстановки) аргументов. Но главное свойство оболочки, которое делает ее мощным инструментом пользователя – это то, что она включает в себя простой язык программирования. Как давно доказано в математике, любой алгоритм можно построить из пары-тройки основных операций и одного условного оператора. Вот реализацию условных операторов (а также операторов цикла) и берет на себя оболочка. Оболочка использует все остальные утилиты и программы (и те, которые имеются в составе операционной системы, и те, что устанавливаются отдельно) как базовые операции поддерживаемого ею языка программирования, обеспечивает передачу им аргументов, а также передачу результатов их работы другим программам или пользователю. В результате получается очень мощный язык программирования. И в этом основная сила и основная функция оболочки.

X Window – завершенный графический интерфейс для Unix-систем, в том числе для Linux. X Window – это окружение, которое обеспечивает множество дополнительных функций, как для пользователя, так и для разработчика программного обеспечения.

Основой концепции ядра X Window является технология "клиент-сервер". На практике это означает, что X Window обеспечивает среду, которая не связана с единственным процессором. Приложение может выполняться на каком-либо сервере или компьютере сети, но отображается (с помощью X Window) на терминалах или рабочих станциях в любом другом месте сети.

Ядром X Window системы является X сервер. Он выполняет следующие задачи:

- поддержка различных типов видеоадаптеров и мониторов;

· управление разрешением, частотой регенерации и глубиной цвета изображения;

· базовое средство управления окнами: отображение и закрытие окон, отслеживание перемещений мыши и нажатий клавиш.

5 Билет: Linux и Unix. Сходства и различия

UNIX

UNIX (*не стоит* путать с определением «UNIX-подобная операционная система») — семейство операционных систем (Mac OS X, GNU/Linux). Первая система была разработана в 1969 в Bell Laboratories, бывшей американской корпорации.
Отличительные особенности UNIX:

1. Простое конфигурирование системы путем использования простых, обычно текстовых, файлов.
2. Широкое использование командной строки.
3. Использование конвейеров.

В наше время UNIX используют в основном на серверах, и как систему для оборудования.

Нельзя не отметить огромную историческую важность UNIX систем. В настоящее время они признаны одними из самых исторически важных ОС. В ходе разработки UNIX систем был создан язык Си.

UNIX-подобная ОС

UNIX-подобная ОС (*иногда* используют сокращение *nix) — система, образованная под влиянием UNIX.
Слово UNIX используется как знак соответствия и как торговая марка.

Консорциум The Open Group обладает торговой маркой «UNIX», но наиболее известен как сертифицирующий орган для торговой марки UNIX. Недавно на The Open Group был пролит свет в связи с публикацией спецификации «Single UNIX Specification», стандартов которым должна удовлетворять ОС чтобы гордо называться Unix.

Linux

Linux — общее название UNIX-пободных операционных систем, которые разработаны в рамках проекта GNU (проект по разработке СПО). Linux работает на огромном множестве архитектур процессора, начиная от ARM заканчивая Intel x86.

Наиболее известными и распространенными дистрибутивами являются Arch Linux, CentOS, Debian. Также существует много «отечественных», российских дистрибутивов — ALT Linux, ASPLinux и другие.

Возникает довольно много споров об именовании GNU/Linux.

Сторонники «open source» используют термин «Linux», а сторонники «free software» — «GNU/Linux». Я предпочитаю первый вариант. Иногда для удобства представления термина GNU/Linux используют написания «GNU+Linux», «GNU-Linux», «GNU Linux».

В отличие от коммерческих систем (MS Windows, Mac OS X) Linux не имеет географического центра разработки и определенной организации, которая владела бы системой. Сама система и программы для нее — результат работы огромных сообществ, тысяч проектов. Присоединиться к проекту или создать свой может каждый!

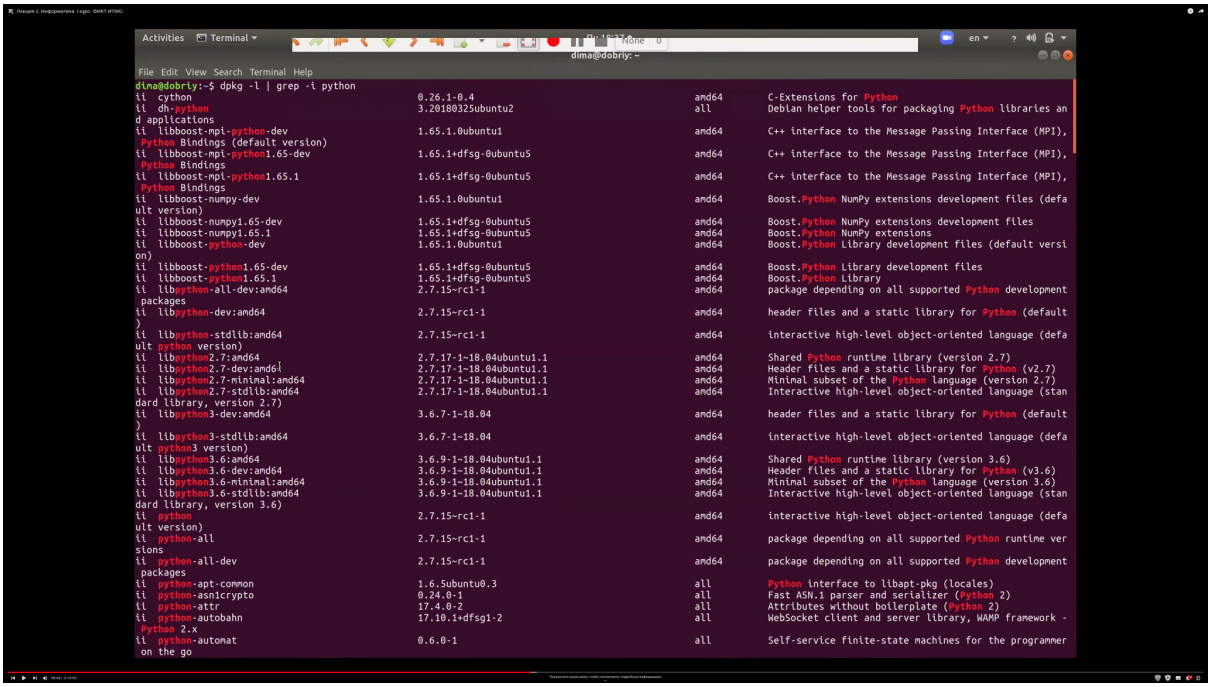
Таким образом у нас получилась цепочка: UNIX -> UNIX-подобная ОС -> Linux.

Подводя итог, я могу сказать, что отличия между Linux и UNIX очевидны. UNIX — намного более широкое понятие, фундамент для построения и сертификации всех UNIX-подобных систем, а Linux — частный случай UNIX.

6 Билет: Регулярные выражения и grep.

grep, **egrep**, **fgrep** – печатают строки, соответствующие шаблону.

Grep ищет по ШАБЛОНУ в каждом ФАЙЛЕ. ФАЙЛ, обозначенный как «-» является стандартным вводом. Если ФАЙЛ не дан, рекурсивные поиски исследуют рабочую директорию (если рекурсия указана соответствующей опцией), и нерекурсивные поиски считывают стандартный ввод. По умолчанию, grep печатает совпадающие строчки3530



Регулярное выражение – это шаблон, который описывает набор строк. Регулярные выражения конструируются аналогично арифметическим выражениям с использованием различных операторов для комбинирования более маленьких выражений.

Программа grep понимает три различных типа синтаксисов регулярных выражений: «**basic**» (BRE), «**extended**» (ERE) и «**perl**» (PCRE). В GNU grep нет разницы в доступной функциональности между базовым «basic» и расширенным синтаксисами «extended». В других реализациях, базовые регулярные выражения менее мощные. Последующее описание применимо к расширенным регулярным выражениям; отличия для базовых регулярных выражений подытожены в конце. Совместимые с Perl регулярные выражения дают дополнительную функциональность, они документированы в **pcresyntax(3)** и **pcrepattern(3)**, но работают только если в системе доступен PCRE.

7 Билет: Опишите синтаксис оператора CASE в скриптах Bash.

Оператор `case` Bash имеет следующую форму:

```
case EXPRESSION in

  PATTERN_1)
    STATEMENTS
    ;;

  PATTERN_2)
    STATEMENTS
    ;;

  PATTERN_N)
    STATEMENTS
    ;;

  *)
    STATEMENTS
    ;;
esac
```

- Каждый оператор case начинается с ключевого слова case , за которым следует выражение case и ключевое слово in . Заявление заканчивается esac словом esac .
- Вы можете использовать несколько шаблонов, разделенных | оператор. Оператор) завершает список шаблонов.
- В шаблоне могут быть специальные символы .
- Шаблон и связанные с ним команды известны как предложение.
- Каждое предложение должно заканчиваться символом ;; .
- Выполняются команды, соответствующие первому шаблону, соответствующему выражению.
- Обычной практикой является использование подстановочного знака звездочки (*) в качестве последнего шаблона для определения регистра по умолчанию. Этот шаблон всегда будет совпадать.
- Если шаблон не найден, статус возврата равен нулю. В противном случае статус возврата — это статус завершения выполненных команд.

8 Билет: Циклы в Bash

- **for** - позволяет перебрать все элементы из массива или использует переменную-счетчик для определения количества повторений;
- **while** - цикл выполняется пока условие истинно;
- **until** - цикл выполняется пока условие ложно

Цикл for

Каждый цикл for независимо от типа начинается с ключевого слова **for**. Дальше все зависит от типа. В этом случае после for указывается имя переменной, в которую будет сохранен каждый элемент списка, затем идет ключевое слово **in** и сам список. Команды, которые нужно выполнять в цикле размещаются между словами **do** и **done**.

```
# !/bin/bash
for number in 1 2 3 4 5
do
echo $number
done
```

```
# !/bin/bash
for ((i=1; i < 10; i++))
do
echo $i
done
```

Цикл while

Суть цикла While в том, что он будет повторяться до тех пор, пока будет выполняться условие, указанное в объявлении цикла. Здесь нет никаких счетчиков и если они вам нужны, то организовывать их вам придется самим. Bash цикл while имеет такой синтаксис:

```
# !/bin/bash
x=1
while [ $x -lt 5 ]
do
echo "Значение счетчика: $x"
x=$(( $x + 1 ))
done
```

Цикл until

Нам осталось рассмотреть последний цикл. Это цикл until. Он очень похож на while и отличается от него только работой условия. Если в первом нужно чтобы условие всегда было истинным, то здесь все наоборот. Цикл будет выполняться пока условие неверно. Синтаксис:

```
# !/bin/bash
count=1
until [ $count -gt 10 ]
do
echo "Значение счетчика: $count"
count=$(( $count + 1 ))
done
```

9 Билет: Функция. Рекурсия

Функция - это набор команд, объединенных одним именем, которые выполняют определенную задачу. Функция вызывается по ее имени, может принимать параметры и возвращать результат работы. Одним словом, функции Bash работают так же, как и в других языках программирования.

Так задается функция:

имя_функции аргумент1 аргумент2 ... аргументN

Примеры функции и рекурсии:

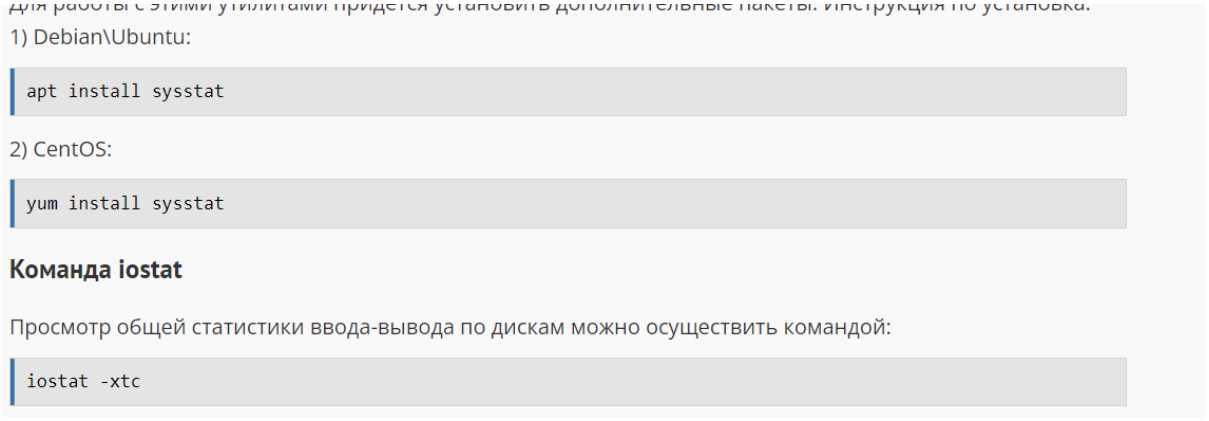
1.
```
$ vi function.sh

#!/bin/bash
printstr(){
echo "hello world"
}
printstr
```

2.
```
# !/bin/bash
printstr(){
echo "hello world"
printstr
}
printstr
```

Как в первой так и во второй функции заданным именем функции является “printstr()”

10 Билет: Каким образом можно просмотреть загруженность диска операциями ввода-вывода?

Статистику по операциям ввода-вывода для дисков можно посмотреть при помощи команд **iostat** и **pidstat**. Это поможет понять какие процессы создают набольшую нагрузку на дисковую подсистему. Еще есть iotop — аналог top для отслеживания нагрузки на диск в реальном времени


Для работы с этими утилитами придется установить дополнительные пакеты. Инструкция по установке:

1) Debian\Ubuntu:

```
apt install sysstat
```

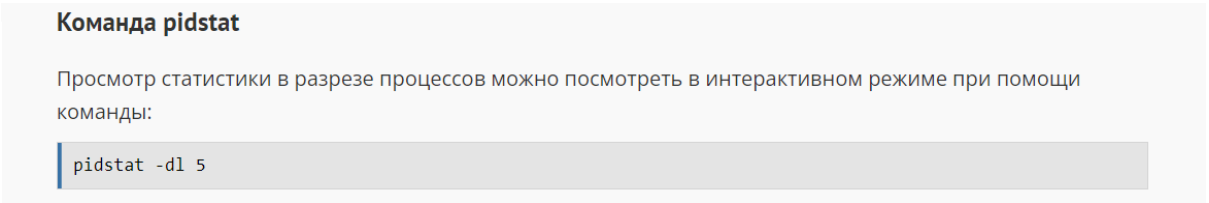
2) CentOS:

```
yum install sysstat
```

Команда iostat

Просмотр общей статистики ввода-вывода по дискам можно осуществить командой:

```
iostat -xtc
```

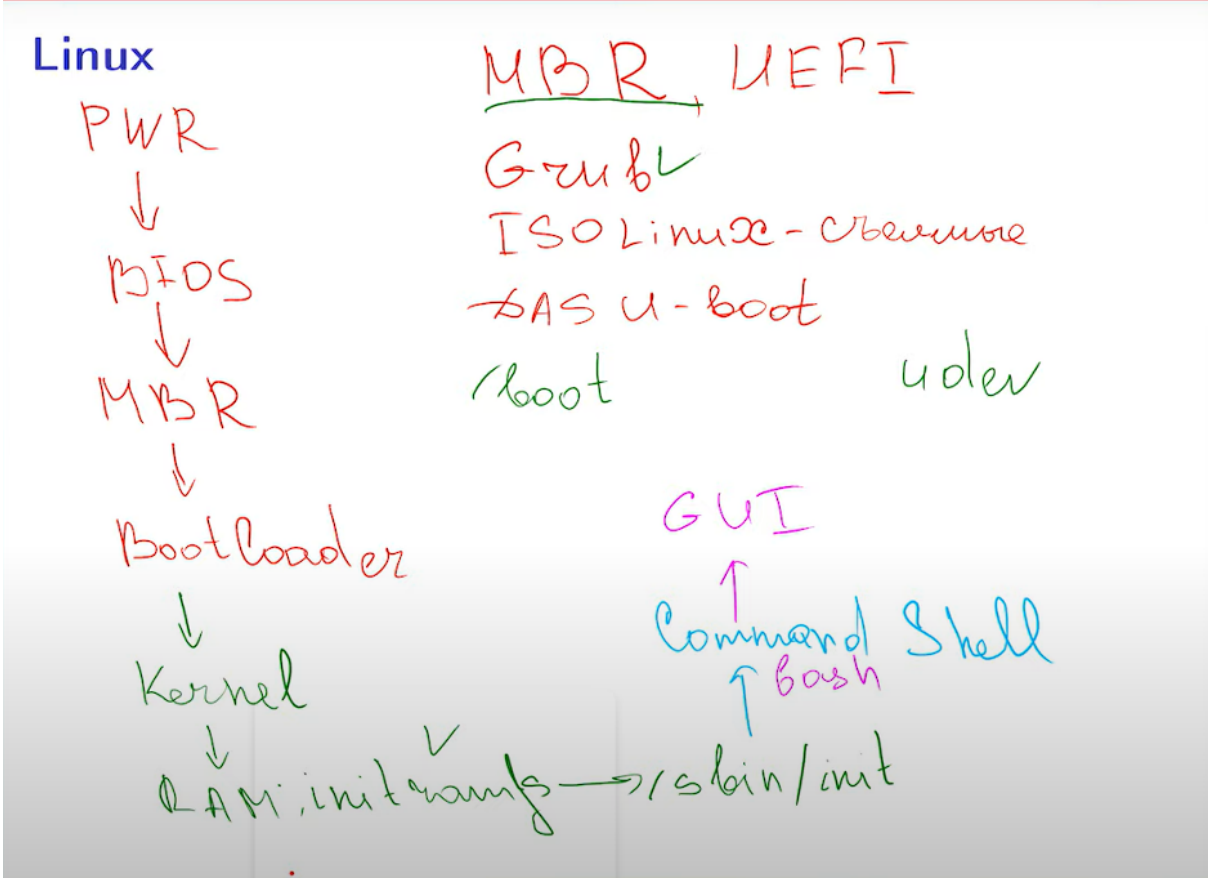

Команда pidstat

Просмотр статистики в разрезе процессов можно посмотреть в интерактивном режиме при помощи команды:

```
pidstat -dl 5
```

11 Билет: Опишите процесс загрузки системы

Процесс загрузки Linux – процедура инициализации системы, включает в себя все с момента включения компьютера до перехода пользовательского интерфейса в рабочее состояние.



PWR – нажатие кнопки питание

Базовая система ввода-вывода инициализирует оборудование (BIOS) – включает экран клавиатуру, проверяет память (POST – самотестирование при включении), находится на микросхеме платы, остальная часть загрузки контролируется операционной системой,

MBR раздел загрузки

UEFI(extensible firmware interface) – считывает данные своего диспетчера загрузки для того, чтобы определить, какое именно приложение UEFI должно быть запущено и откуда (на каком диске и на каком разделе), потом программа запускает приложение GRUB. Более универсален, чем MBR.

DAS U – boot - загрузчики на встроенных устройствах, могут загружать другие ОС

Загрузчик отвечает за первоначальную загрузку ядра и начального диска или файловой системы, которая содержит некоторые важные файлы и драйверы устройств, которые необходимы для запуска системы, он их подгружает в память. Загрузчик находится в первом секторе жесткого диска.

MBR (main boot record) – основная загрузочная запись, она находится в первом секторе жесткого диска, размер – 512 байт, загрузчик проверяет таблицу разделов, находит загрузочный раздел, после того, как он нашел его, он ищет

загрузчик второй ступени (например, grub) и загружает его в оперативную память.

Bootloader (GRUB – унифицированный, isolinux-загрузка с носителя) – загружает ОС, хранится на жестком диске системы в разделе загрузки, до этого момента машина не имеет доступа к личным файлам и дискам, после этого информация о времени, дате, периферийных устройствах загружается из значения Simos (хранения и память с батарейным питанием, позволяет отслеживать дату и время при выключенном питании).

Загрузчик второго уровня находится в директории boot, отображает экран заставки и позволяет выбрать ОС, после этого загружает ядро ОС в оперативную память и передает управление ей. Ядра всегда сжаты, поэтому оно сначала должно себя распаковать, потом проверить себя, анализировать аппаратное обеспечение и инициализировать все драйверы аппаратных устройств, которые встроены в ядро.

Kernel-ядро.

RAM – первоначальная загрузка памяти, далее инициализация файловой системы эмитером ramfs, содержащую файлы, которые выполняют все действия, которые необходимы для монтирования правильной корневой системы (обеспечения функциональности ядра для необходимой файловой системы, драйверов устройств, контроллеров запоминающих устройств с помощью средства udef, отвечающего за инициализацию подключенных устройств, поиск драйверов устройств). После того, как найдена корневая файловая система, она проходит проверку на наличие ошибок и монтируется. Программа монтирования сообщает ОС, что файловая система готова к использованию и связывает ее с определенной точкой в общей иерархии файловой системы (точкой монтирования). Если это произошло успешно, управление переходит к init, находящейся в корневой файловой системе, она обрабатывает монтирование и переходит к корневой файловой системе. Если требуются специальные драйвера, они должны быть в initramfs.

Ближе к концу загрузки программа init запускает ряд запросов на вход для начала в текстовом режиме. Если графического интерфейса нет, можно уже там работать. Если есть интерфейс, мы его даже не увидим.

Command Shell (по умолчанию bash – bourne again shell) - отражает поле ввода, можно вводить команды.

Загрузчик загружает как ядро, так и файловую систему в память. Ядро настривает все оборудование и загружает некоторые необходимые приложения пользовательского пространства, инициализирует и настраивает память компьютера. Потом запускает is been init.

Большинство процессов в системе прослеживают свое происхождение вплоть до init, исключение – процессы ядра, они запускаются ядром и управляют внутренними деталями процессов системы. Init отвечает за поддержание работы системы и ее корректное завершение, одна особенность – действует в качестве менеджера для всех неядерных процессов, очищает после них по завершении и перезапускает необходимые службы входа пользователя по мере необходимости, когда пользователи входят и выходят из системы, тоже самое делает для других фоновых системных служб.

Если есть графический интерфейс, финальный шаг – запуск графического интерфейса (GUI).

12 Билет: Стандартные потоки ввода-вывода. Стандартный поток ошибок.

В стандартный поток ввода поступают данные, которые вводятся с клавиатуры. Стандартным потоком вывода является экран монитора. Другими словами, в командную строку Bash'a поступают данные, вводимые с клавиатуры. Если команда предполагает текстовый вывод, Bash выводит данные на экран.

Однако такое поведение по-умолчанию можно изменить. Данные могут поступать, например, из файла, а не с клавиатуры. Аналогично, они могут выводиться в файл, а не на экран.

Предположим, мы хотим, не выходя из терминала в графический режим, записать в файл заметку. Для этого не обязательно использовать консольный текстовый редактор. Это можно сделать с помощью программы cat, перенаправив ее вывод в файл. Вспомним, что cat без аргументов работает в интерактивном режиме, и все, что мы вводим, тут же выводится на экран. В Bash знак больше > обозначает перенаправление стандартного потока вывода. Два знака больше >> – это тоже перенаправление вывода, но такое, когда данные добавляются в конец объекта (в данном случае файла), если он существует. Используй мы только один знак больше, файл был бы перезаписан.

Файл стандартного потока ошибок (stderr) имеет дескриптор 2. В этот файл записываются сообщения об ошибках, возникающих в ходе выполнения команды. По умолчанию сообщения об ошибках выводятся на экран терминала

(устройство /dev/tty), но их также можно перенаправить в файл. Зачем же для регистрации ошибок выделять специальный файл? Дело в том, что это очень удобный способ выделения из результатов работы команды собственно выходных данных, а также хорошая возможность эффективно организовать ведение различного рода журнальных файлов.

Чтобы перенаправить стандартный поток ошибок в файл, используйте оператор `2>'. Укажите после имени команды оператор 2>, а затем имя файла, который будет служить приемником ошибок выполнения программы. Например, чтобы записать ошибки выполнения программы programma в файл programma.errors, введите:

$ programma 2> programma.errors [Enter]

Чтобы добавить стандартный поток ошибок в уже существующий файл, используйте оператор `»' вместо `>'.

$ programma 2>> programma.errors [Enter]

Чтобы перенаправить в один и тот же файл поток вывода и поток ошибок, используйте оператор `SPMamp;>'.

$ programma &> result_with_errors [Enter]

13 Билет: Переменные среды. Переменные оболочки

Переменные среды — это переменные, которые были определены для текущей оболочки и *наследуются всеми дочерними оболочками* или процессами. Переменные среды используются для передачи информации процессам, запущенным из оболочки.

Переменные оболочки — это локальные переменные, которые содержатся исключительно в оболочке, в которой они были установлены или определены. Они часто используются для отслеживания текущих данных (к примеру, текущего рабочего каталога).

Для *просмотра списка всех переменных среды*, используются команды env или printenv.

Некоторые особенно полезные переменные среды и оболочки используются очень часто. Ниже приведен список основных переменных среды:

- SHELL: описывает оболочку, которая интерпретирует введенные команды. В большинстве случаев по умолчанию установлена bash, но это значение можно изменить в случае необходимости.
- TERM: указывает вид терминала, эмулируемого при запуске оболочки. В зависимости от операционных требований можно эмулировать разные аппаратные терминалы.
- USER: текущий пользователь.
- PWD: текущий рабочий каталог.

Ознакомившись со списком переменных среды, посмотрим на список переменных оболочки:

- BASHOPTS: список опций, использованных при выполнении bash. Это можно применять для того, чтоб проверить, работает ли среда должным образом.
- BASH_VERSION: запущенная версия bash в удобочитаемой форме.
- BASH_VERSINFO: версия bash в машиночитаемом формате.
- COLUMNS: определяет ширину вывода в столбцах.

Это делается очень просто, для этого нужно только указать имя и значение. Как уже было сказано, для написания имен таких переменных используются заглавные буквы.
$ TEST_VAR='Hello World!'

Теперь попробуем превратить переменную оболочки в переменную среды. Это делается путем экспорта переменной:
$ export TEST_VAR

14 Билет: Перенаправление потоков и соединение их в определенном порядке для эффективной обработки текста.

Для того, чтобы понять то, о чём мы будем тут говорить, важно знать, откуда берутся данные, которые можно перенаправлять, и куда они идут. В Linux существует три стандартных потока ввода/вывода данных.

Первый — это стандартный поток ввода (standard input). В системе это — поток №0 (так как в компьютерах счёт обычно начинается с нуля).

Второй поток — это стандартный поток вывода (standard output), ему присвоен номер 1. Это поток данных, которые оболочка выводит после выполнения каких-то действий.

И, наконец, третий поток — это стандартный поток ошибок (standard error), он имеет дескриптор 2.

Предположим, вы хотите создать файл, в который будут записаны текущие дата и время. Дело упрощает то, что имеется команда, удачно названная `date`, которая возвращает то, что нам нужно. Обычно команды выводят данные в стандартный поток вывода. Для того, чтобы эти данные оказались в файле, нужно добавить символ `>` после команды, перед именем целевого файла. До и после `>` надо поставить пробел.

При использовании перенаправления любой файл, указанный после `>` будет перезаписан. Если в файле нет ничего ценного и его содержимое можно потерять, в нашей конструкции допустимо использовать уже существующий файл. Обычно же лучше использовать в подобном случае имя файла, которого пока не существует. Этот файл будет создан после выполнения команды. Назовём его `date.txt`. Расширение файла после точки обычно особой роли не играет, но расширения помогают поддерживать порядок.

И, наконец, поговорим о перенаправлении стандартного потока ошибок. Это может понадобиться, например, для создания лог-файлов с ошибками или объединения в одном файле сообщений об ошибках и возвращённых некоей командой данных.

Например, что если надо провести поиск во всей системе сведений о беспроводных интерфейсах, которые доступны пользователям, у которых нет прав суперпользователя? Для того, чтобы это сделать, можно воспользоваться мощной командой `find`.

Обычно, когда обычный пользователь запускает команду `find` по всей системе, она выводит в терминал и полезные данные и ошибки. При этом, последних обычно больше, чем первых, что усложняет нахождение в выводе команды того, что нужно. Решить эту проблему довольно просто: достаточно перенаправить стандартный поток ошибок в файл, используя команду 2> (напомним, 2 — это дескриптор стандартного потока ошибок). В результате на экран попадёт только то, что команда отправляет в стандартный вывод.

15 Билет: Основы управления файлами и директориями

Специальный каталог:

```
.                       Представляет этот каталог слоя
..                      Представляет верхний класс
 - представляет предыдущий рабочий каталог
 ~ Представитель[Текущая идентичность пользователя]Главный каталог пользователей
 ~ Учетная запись представляет домашний каталог пользователя пользователя
```

инструкции по обработке Общего каталога:

```
cd              Преобразовать работу каталога
pwd             Отображение текущего каталога -P, где отображается реальный каталог, не исп
mkdir   Создайте новый каталог -M Profile Permission, чтобы установить напрямую, не нужно в
rmdir   Удалить пустой каталог -P рекурсивное удалить все воздушные каталоги
```

Получение 2.1, файлов и каталогов: ls

```
[root@mysql ~]# ls [опция и параметр] Имя файла или имя каталога ...
 Опции и параметры:
 -A: Все файлы вместе со скрытыми файлами (файлы, начинающиеся.)
 -A: все файлы, но не включены...содержание
 -d: только перечисляет себя вместо списка данных файла в каталоге.
 -H: Перечислите содержимое файла, чтобы легко прочитать.
 -Я: Листинг INODE номер
 -L: Строковые списки длинных данных, включая свойства и разрешения файла и т. Д.
-n: Список UID и GID
 -S: размер файла сортировать по мощности
--time={atime,ctime}: Выход Время доступа или изменить время (CTime), не содержание изменен
```

2.2.3, мв (перемещение файлов и каталогов, или переименовать)

```
[Корень @ MySQL ~] # мв [-fiu] целевой файл исходного файла
 Опции и параметры:
 -f: смысл сил, цель уже существует, непосредственно перезаписывать
 -i: интерактивный режим, запрос, прежде чем крышка
 -u: Обновление для обновления
```

```
# 1, скопировать файл, создать каталог, переместить файл в
[root@mysql tmp]# cp ~/.bashrc bashrc
[root@mysql tmp]# mkdir mvtest
[root@mysql tmp]# mv bashrc mvtest

# 2, переименуйте каталог
[root@mysql tmp]# mv mvtest mvtest2
```

16 Билет: Управление учетными записями пользователей и групп и связанные системные файлы

UNIX-подобные операционные системы являются многопользовательскими. Пользователи и группы в которых они состоят используются для управления доступом к системным файлам, каталогам и периферии.

Пользователь - это любой кто пользуется компьютером. Каждому пользователю назначается уникальный идентификационный номер(UID). Операционная система отслеживает пользователя именно по UID, а не по его имени (логину).

Для разграничения прав в linux, помимо пользователей, существуют **группы**. Так же как и пользователь, группа обладает правами доступа к тем или иным каталогам, файлам, периферии. Для каждого файла определён не только пользователь, но и группа. Группы группируют пользователей для предоставления одинаковых полномочий на какие-либо действия.

Каждой группе назначается идентификационный номер, который является уникальный идентификатором группы. Операционная система отслеживает группу по GID.Принадлежность пользователя к группе устанавливается администратором.

Все операции в многопользовательских ОС должны выполняться именно с группами пользователей. Для групп должны назначаться определённые права доступа и безопасности. Если мы хотим предоставить пользователю какие-то определённые права, то мы просто добавляем его в соответствующую группу. Даже если у нас только один пользователь с таким видом доступа, то для него всё равно нужно создать группу. Ведь если появится пользователь которому потребуются такие же права, то мы просто добавим его в определённую группу и не нужно лишний раз настраивать права доступа для самого пользователя и его учётной записи.

17 Билет: Настройка и создание простых сценариев.

1. Создайте сценарий, который выводит имя города.

```
$ echo 'echo Antwerp' > first.bash
$ chmod +x first.bash
$ ./first.bash
Antwerp
```

2. Сделайте так, чтобы сценарий исполнялся в командной оболочке bash.

```
$ cat first.bash
#!/bin/bash
echo Antwerp
```

3. Сделайте так, чтобы сценарий исполнялся в командной оболочке Korn shell.

```
$ cat first.bash
```

```
#!/bin/ksh
echo Antwerp
```

Обратите внимание на то, что хотя данный сценарий и будет технически исполняться как сценарий командной оболочки Korn shell, расширение файла .bash может смутить пользователя.

4. Создайте сценарий, в котором осуществляется объявление двух переменных с последующим выводом их значений.

```
$ cat second.bash
#!/bin/bash

var33=300
var42=400

echo $var33 $var42
```

5. Предыдущий сценарий не должен оказывать воздействия на вашу рабочую командную оболочку (объявленные в нем переменные не будут существовать вне сценария). А теперь запустите сценарий таким образом, чтобы он оказал влияние на вашу рабочую командную оболочку.

```
source second.bash
```

6. Существует ли более короткая форма команды для использования рабочей командной оболочки с целью исполнении сценария?

```
. ./second.bash
```

7. Добавьте комментарии в ваши сценарии для того, чтобы знать о выполняемых ими операциях.

```
$ cat second.bash
#!/bin/bash
# сценарий для проверки работы механизма использования рабочей
командной оболочки

var33=300
var42=400

# вывод значений этих переменных
echo $var33 $var42
```

18 Билет: Права доступа в Linux-системах.

Изначально каждый файл имел три параметра доступа. Вот они:

- **Чтение** - разрешает получать содержимое файла, но на запись нет. Для каталога позволяет получить список файлов и каталогов, расположенных в нем;
- **Запись** - разрешает записывать новые данные в файл или изменять существующие, а также позволяет создавать и изменять файлы и каталоги;
- **Выполнение** - вы не можете выполнить программу, если у нее нет флага выполнения. Этот атрибут устанавливается для всех программ и скриптов, именно с помощью него система может понять, что этот файл нужно запускать как программу.

Но все эти права были бы бессмысленными, если бы применялись сразу для всех пользователей. Поэтому каждый файл имеет три категории пользователей, для которых можно устанавливать различные сочетания прав доступа:

- **Владелец** - набор прав для владельца файла, пользователя, который его создал или сейчас установлен его владельцем. Обычно владелец имеет все права, чтение, запись и выполнение.
- **Группа** - любая группа пользователей, существующая в системе и привязанная к файлу. Но это может быть только одна группа и обычно это группа владельца, хотя для файла можно назначить и другую группу.
- **Остальные** - все пользователи, кроме владельца и пользователей, входящих в группу файла.

Именно с помощью этих наборов полномочий устанавливаются права файлов в linux. Каждый пользователь может получить полный доступ только к файлам, владельцем которых он является или к тем, доступ к которым ему разрешен. Только пользователь Root может работать со всеми файлами независимо от их набора их полномочий.

Но со временем такой системы стало не хватать и было добавлено еще несколько флагов, которые позволяют делать файлы не изменяемыми или же выполнять от имени суперпользователя, их мы рассмотрим ниже:

Специальные права доступа к файлам в Linux

Для того, чтобы позволить обычным пользователям выполнять программы от имени суперпользователя без знания его пароля была придумана такая вещь, как SUID и SGID биты. Рассмотрим эти полномочия подробнее.

- **SUID** - если этот бит установлен, то при выполнении программы, id пользователя, от которого она запущена заменяется на id владельца файла. Фактически, это позволяет обычным пользователям запускать программы от имени суперпользователя;
- **SGID** - этот флаг работает аналогичным образом, только разница в том, что пользователь считается членом группы, с которой связан файл, а не групп, к которым он действительно принадлежит. Если SGID флаг установлен на каталог, все файлы, созданные в нем, будут связаны с группой каталога, а не пользователя. Такое поведение используется для организации общих папок;
- **Sticky-bit** - этот бит тоже используется для создания общих папок. Если он установлен, то пользователи могут только создавать, читать и выполнять файлы, но не могут удалять файлы, принадлежащие другим пользователям.

Чтобы узнать права на файл linux выполните такую команду, в папке где находится этот файл:

```
$ ls -l
```

Чтобы изменить права на файл в linux вы можете использовать утилиту chmod. Она позволяет менять все флаги, включая специальные. Рассмотрим ее синтаксис:

$ chmod опции категориядействиефлаг файл

Опции сейчас нас интересовать не будут, разве что только одна. С помощью опции -R вы можете заставить программу применять изменения ко всем файлам и каталогам рекурсивно.

Категория указывает для какой группы пользователей нужно применять права, как вы помните доступно только три категории:

- **u** - владелец файла;
- **g** - группа файла;
- **o** - другие пользователи.

Действие может быть одно из двух, либо добавить - знак **"+"**, либо убрать - знак - **"-"**. Что касается самих прав доступа, то они аналогичны выводу утилиты ls: **r** - чтение, **w** - запись, **x** - выполнение, **s** - suid/sgid, в зависимости от категории, для которой вы его устанавливаете, **t** - устанавливает sticky-bit.

19 Билет: Удаленный доступ. SSH, telnet.

SSH

SSH (Secure Shell) - протокол канала безопасностиВ основном он используется для реализации удаленного входа в систему и удаленной репликации интерфейса символов.
Протокол SSH зашифровал передачу данных связи между связью между сообщениямиОн включает в себя пароль пользователя, введенный при входе в систему пользователя. Поэтому протокол SSH обладает хорошей безопасностью.
сеть
OpenSSH - это программный проект с открытым исходным кодом, который реализует протокол SSH, который подходит для различных операционных систем UNIX и Linux. Система Centos 7 установила программные пакеты, связанные с OpenSSH, и добавила SSHD -сервисы для начала самостоятельного начала. Выполните команду «SystemCtl Start SSHD», чтобы запустить службу SSHD.

Telnet

Telnet - это сетевая утилита, которая позволяет соединиться с удаленным портом любого компьютера и установить интерактивный канал связи, например, для передачи команд или получения информации. Можно сказать, что это универсальный браузер в терминале, который умеет работать со множеством сетевых протоколов.
Она работает по протоколу TELNET, но этот протокол поддерживается многими сервисами, поэтому ее можно использовать для управления ими. Протокол работает на основе TCP, и позволяет передавать обычные строковые команды на другое устройство. Он может использоваться не только для ручного управления но и для взаимодействия между процессами.

Для работы с этим протоколом мы будем использовать утилиту telnet, ею очень просто пользоваться. Давайте рассмотрим синтаксис telnet:

$ telnet опции хост порт

Хост - это домен удаленного компьютера, к которому следует подключиться, а порт - порт на этом компьютере. А теперь давайте рассмотрим основные опции:

- **-4** - принудительно использовать адреса ipv4;
- **-6** - принудительно использовать адреса ipv6;
- **-8** - использовать 8-битную кодировку, например, Unicode;
- **-E** - отключить поддержку Escape последовательностей;
- **-a** - автоматический вход, берет имя пользователя из переменной окружения USER;
- **-b** - использовать локальный сокет;
- **-d** - включить режим отладки;
- **-p** - режим эмуляции rlogin;
- **-e** - задать символ начала Escape последовательности;
- **-l** - пользователь для авторизации на удаленной машине.

Это все, что касается команды telnet для установки соединения. Но соединение с удаленным хостом, это только полдела. После установки подключения telnet может работать в двух режимах:

- **Построчный** - это предпочтительный режим, здесь строка текста редактируется на локальном компьютере и отправляется только тогда, когда она будет полностью готова. На такая возможность есть не всегда и не у всех сервисов;
- **Посимвольный** - все набираемые вами символы отправляются на удаленный сервер. Тут будет сложно что-либо исправить, если вы допустили ошибку, потому что Backspace тоже будет отправляться в виде символа и стрелки движения тоже.

Использование telnet заключается в передаче специальных команд. У каждого сервиса свои команды, но у протокола есть свои команды telnet, которые можно применять в консоли telnet.

- **CLOSE** - закрыть соединение с сервером;
- **ENCRYPT** - шифровать все передаваемые данные;
- **LOGOUT** - выйти и закрыть соединение;
- **MODE** - переключить режим, со строчного на символьный или с символьного на строчный;
- **STATUS** - посмотреть статус соединения;
- **SEND** - отправить один из специальных символов telnet;
- **SET** - установить значение параметра;
- **OPEN** - установить подключение через telnet с удаленным узлом;

- **DISPLAY** - отобразить используемые спецсимволы;
- **SLC** - изменить используемые спецсимволы.

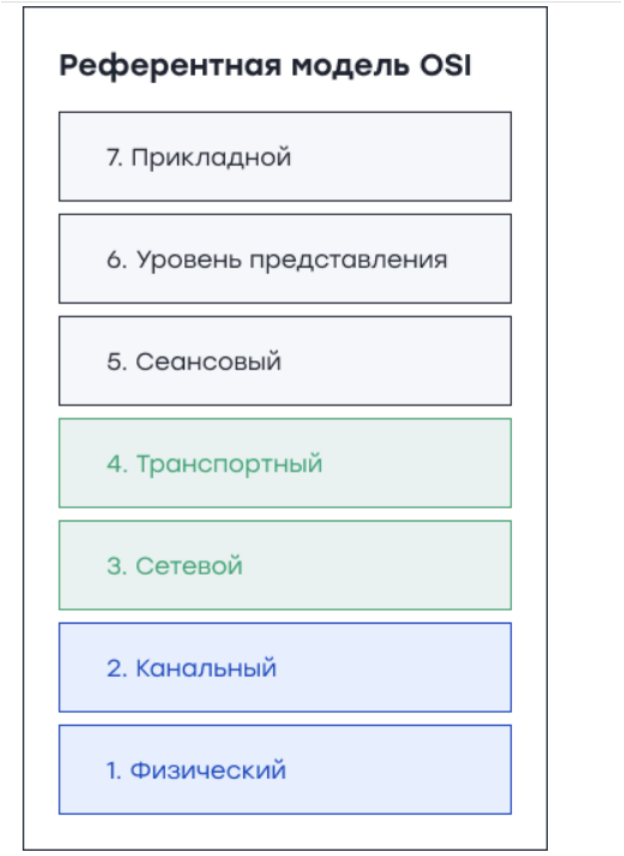
Мы не будем рассматривать все команды, поскольку они вам вряд ли понадобятся, а если и понадобятся, то вы легко сможете их найти в официальной документации.

3. Компьютерные сети

20 Билет: Сетевые модели OSI и TCP/IP.

OSI:

Сетевая модель OSI (The Open Systems Interconnection model) — сетевая модель стека (магазина) сетевых протоколов OSI/ISO. Посредством данной модели различные сетевые устройства могут взаимодействовать друг с другом. Модель определяет различные уровни взаимодействия систем. Каждый уровень выполняет определённые функции при таком взаимодействии.



Модель OSI

Уровень 1: Физический

Начнем (кто бы удивился) с уровня 1. Здесь происходит обмен оптическими, электрическими или радиосигналами между устройствами отправителя и получателя.

На этом уровне железо не распознает данные в классическом для нас виде (картинки, текст, видео), но оно понимает биты (единицы и нули) и работает только с сигналами. Таким оборудованием выступают концентраторы, медиаконвертеры или репитеры. Здесь информация или биты передаются либо по проводам, кабелям, либо без них, например через Bluetooth, Wi-Fi.

Когда возникает проблема с сетью, многие специалисты сразу же обращаются к физическому уровню, чтобы проверить, например, не отключен ли сетевой кабель от устройства.

Уровень 2: Канальный

Мы прошли первый уровень. Что же дальше? Если в локальной сети находится более двух устройств, то необходимо определить, куда конкретно направлять информацию. Этим занимается как раз канальный уровень, принимающий на себя важную роль адресации.

Второй уровень принимает биты и трансформирует их в кадры (фреймы). Здесь существуют MAC-адреса (Media Access Control), которые необходимы для идентификации устройств. На втором уровне происходит еще проверка на ошибки, и исправление информации, а также управление ее передачей. Этим занимается LLC (Logical Link Control).

На канальном уровне работают уже более умные железки – коммутаторы. Их задачей является передача кадров от одного устройства другому, используя MAC-адреса.

Уровень 3: Сетевой

На третьем уровне происходит маршрутизация трафика. Этим занимаются такие устройства, как роутеры или маршрутизаторы.

На сетевом уровне работает протокол ARP (Address Resolution Protocol), который определяет соответствие между логическим адресом сетевого уровня (IP) и физическим адресом устройства (MAC). Здесь пересылаемая информация выступает уже в виде пакетов, состоящих из заголовка и поля данных.

Информация об известных IP и MAC-адресах хранится в виде таблицы (ARP-таблица) с данными, что позволяет устройствам не тратить время на повторную идентификацию.

Уровень 4: Транспортный

Четвертый уровень получает пакеты и передает их по сети. Он отвечает за установку соединения, надежность и управление потоком. Блоки данных делятся на отдельные фрагменты, размеры которых зависят от используемого протокола. Главными героями тут выступают 2 протокола TCP (Transmission Control Protocol) и UDP (User Datagram Protocol). В чем их отличие и когда их применять?

При транспортировке данных, наиболее восприимчивых к потерям, например, web-страницы, задействуется протокол TCP с установлением соединения. Он контролирует целостность информации, в данном случае нашей страницы, ибо потеря какого-то контента заставит задуматься пользователя о его полезности. Чтобы сделать передачу более эффективной и быстрой, транспортный уровень разбивает данные на более мелкие сегменты.

UDP-протокол используется с данными, для которых потери не так критичны, например, мультимедиа-трафик. Для них более заметна будет задержка, поэтому UDP обеспечивает связь без установки соединения. Во время передачи данных с помощью протокола UDP, пакеты делятся уже на автономные датаграммы. Они могут доставляться по разным маршрутам и в разной последовательности.

Уровень 5: Сеансовый

Уровни с пятого по седьмой уже работают с чистыми данными. И здесь за дело берутся не сетевые инженеры, а разработчики.

Сеансовый уровень, исходя из названия, отвечает за поддержание сеанса или сессии. Он координирует коммуникацию между приложениями и отвечает за установление, поддержание и завершение связи, синхронизацию задач и сам обмен информацией. Примером для пятого уровня можно назвать созвон в Zoom или прямой эфир на YouTube. Во время сессии необходимо обеспечивать синхронизированную передачу аудио и видео для всех участников, а также поддерживать саму связь. За это как раз отвечают протоколы сеансового уровня (RPC, H.245, RTCP).

Уровень 6: Уровень представления

Шестой уровень подготавливает информацию для последнего и преобразует (сжимает, кодирует, шифрует) их в понятный язык для пользователя или машины. Например, если вы отправляете картинку, то она сначала приходит в виде битов, а потом трансформируются в JPEG, GIF или другой формат.

Уровень 7: Прикладной

Верхний уровень модели OSI – это прикладной. С помощью своих протоколов он отображает данные в понятном конечному пользователю формате. Сюда входят такие технологии, как HTTP, DNS, FTP, SSH и многое другое. Почти каждый человек ежедневно взаимодействует с протоколами прикладного уровня.

TCP/IP:

TCP/IP — сетевая модель передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю. В модели предполагается прохождение информации через четыре уровня, каждый из которых описывается правилом (протоколом передачи). Наборы правил, решающих задачу по передаче данных, составляют стек протоколов передачи данных, на которых базируется Интернет. Название TCP/IP происходит из двух важнейших протоколов семейства — Transmission Control Protocol (TCP) и Internet Protocol (IP), которые были первыми разработаны и описаны в данном стандарте. Также изредка упоминается как модель DOD (Department of Defense) в связи с историческим происхождением от сети ARPANET из 1970-х годов (под управлением DARPA, Министерства обороны США).

Набор интернет-протоколов — это концептуальная модель и набор коммуникационных протоколов, используемых в Интернете и подобных компьютерных сетях. Он широко известен как TCP/IP, поскольку базовые протоколы в пакете — это протокол управления передачей (TCP) и интернет-протокол (IP). Его иногда называют моделью Министерства обороны (MO), поскольку разработка сетевого метода финансировалась Министерством обороны Соединенных Штатов через DARPA.

Набор интернет-протоколов обеспечивает сквозную передачу данных, определяющую, как данные должны пакетироваться, обрабатываться, передаваться, маршрутизироваться и приниматься. Эта функциональность организована в четыре слоя абстракции, которые классифицируют все связанные протоколы в соответствии с объемом задействованных сетей. От самого низкого до самого высокого уровня — это уровень связи, содержащий методы связи для данных, которые остаются в пределах одного сегмента сети; интернет-уровень, обеспечивающий межсетевое взаимодействие между независимыми сетями; транспортный уровень, обрабатывающий связь между хостами; и прикладной уровень, который обеспечивает обмен данными между процессами для приложений.

Уровневая модель TCP/IP

Выше мы уже упоминали, что модель TCP/IP разделена на уровни, как и OSI, но отличие двух моделей в количестве уровней. Документами, определяющими сертификацию модели, являются RFC 1122 и RFC1123. Эти стандарты описывают четыре уровня абстракции модели TCP/IP: прикладной, транспортный, межсетевой и канальный. Существуют и другие версии описания модели, в том числе включающие иное количество уровней и их наименований. Однако в этой статье мы придерживаемся оригинальной версии и далее рассмотрим четыре уровня модели.

Канальный уровень (link layer)

Предназначение канального уровня — дать описание тому, как происходит обмен информацией на уровне сетевых устройств, определить, как информация будет передаваться от одного устройства к другому. Информация здесь кодируется, делится на пакеты и отправляется по нужному каналу, т.е. среде передачи.

Этот уровень также вычисляет максимальное расстояние, на которое пакеты возможно передать, частоту сигнала, задержку ответа и т.д. Все это — физические свойства среды передачи информации. На канальном уровне самым распространенным протоколом является Ethernet, который мы рассмотрим в конце статьи.

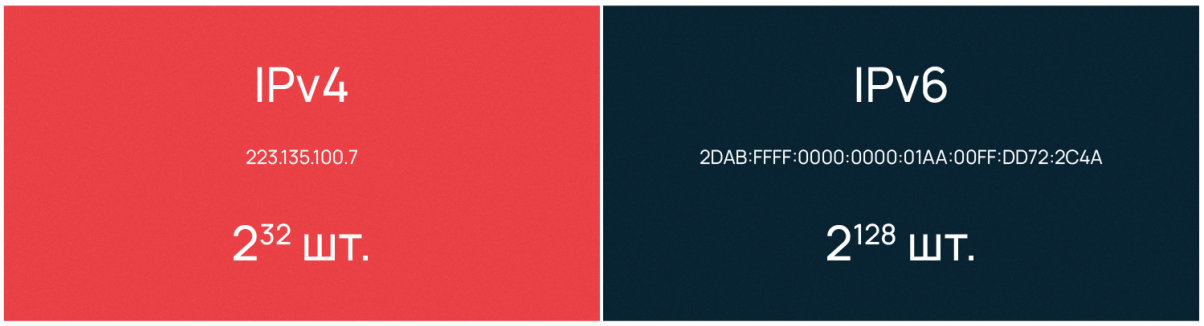
Межсетевой уровень (internet layer)

Глобальная сеть интернет состоит из множества локальных сетей, взаимодействующих между собой. Межсетевой уровень используется, чтобы описать обеспечение такого взаимодействия.

Межсетевое взаимодействие — это основной принцип построения интернета. Локальные сети по всему миру объединены в глобальную, а передачу данных между этими сетями осуществляют магистральные и пограничные маршрутизаторы.

Именно на межсетевом уровне функционирует протокол IP, позволивший объединить разные сети в глобальную. Как и протокол TCP, он дал название модели, рассматриваемой в статье.

Маска подсети и IP-адреса



Маска подсети помогает маршрутизатору понять, как и куда передавать пакет. Подсетью может являться любая сеть со своими протоколами. Маршрутизатор передает пакет напрямую, если получатель находится в той же подсети, что и отправитель. Если же подсети получателя и отправителя различаются, пакет передается на второй маршрутизатор, со второго на третий и далее по цепочке, пока не достигнет получателя.

Протокол IP (Internet Protocol) используется маршрутизатором, чтобы определить, к какой подсети принадлежит получатель. Свой уникальный IP-адрес есть у каждого сетевого устройства, при этом в глобальной сети не может существовать два устройства с одинаковым IP. Протокол имеет две действующие версии, первая из которых — IPv4 (IP version 4, версии 4) — была описана в 1981 году.

IPv4 предусматривает назначение каждому устройству 32-битного IP-адреса, что ограничивало максимально возможное число уникальных адресов 4 миллиардами (2^32). В более привычном для человека десятичном виде IPv4 выглядит как четыре блока (октета) чисел от 0 до 255, разделенных тремя точками. Первый октет IP-адреса означает класс сети, классов всего 5: A, B, C, D, E. Gри этом адреса сети D являются мультикастовыми, а сети E вообще не используются.

Рассмотрим, например, IPv4 адрес класса C 223.135.100.7. Первые три октета определяют класс и номер сети, а последний означает номер конечного устройства. Например, если необходимо отправить информацию с компьютера

номер 7 с IPv4 адресом 223.135.100.7 на компьютер номер 10 в той же подсети, то адрес компьютера получателя будет следующий: 223.135.100.10.

В связи с быстрым ростом сети интернет остро вставала необходимость увеличения числа возможных IP-адресов. В 1995 году впервые был описан протокол IPv6 (IP version 6, версии 6), который использует 128-битные адреса и позволяет назначить уникальные адреса для 2^128 устройств.

IPv6 имеет вид восьми блоков по четыре шестнадцатеричных значения, а каждый блок разделяется двоеточием. IPv6 выглядит следующим образом:

2dab:ffff:0000:0000:01aa:00ff:dd72:2c4a.

Так как IPv6 адреса длинные, их разрешается сокращать по определенным правилам, которые также описываются RFC:

- Для написания адреса используются строчные буквы латинского алфавита: a, b, c, d, e, f.
- Ведущие нули допускается не указывать — например, в адресе выше :00ff: можно записать как :ff:.
- Группы нулей, идущие подряд, тоже допустимо сокращать и заменять на двойное двоеточие. На примере выше это выглядит так: 2dab:aaaa::01aa:00ff:dd72:2c4a. Допускается делать не больше одного подобного сокращения в адресе IPv6 на наибольшей последовательности нулей. Если одинаково длинных последовательностей несколько — на самой левой из них.

IP предназначен для определения адресата и доставки ему информации. Он предоставляет услугу для вышестоящих уровней, но не гарантирует целостность доставляемой информации.

IP способен инкапсулировать другие протоколы, предоставлять место, куда они могут быть встроены. К таким протоколам, например, относятся ICMP (межсетевой протокол управляющих сообщений) и IGMP (межсетевой протокол группового управления). Информация о том, какой протокол инкапсулируется, отражается в заголовке IP-пакета. Так, ICMP будет обозначен числом 1, а IGMP будет обозначен числом 2.

ICMP

ICMP в основном используется устройствами в сети для доставки сообщений об ошибках и операционной информации, сообщающей об успехе или ошибке при связи с другим устройством. Например, именно с использованием ICMP осуществляется передача отчетов о недоступности устройств в сети. Кроме того, ICMP используется при диагностике сети — к примеру, в эксплуатации утилит ping или traceroute.

ICMP не передает какие-либо данные, что отличает его от протоколов, работающих на транспортном уровне — таких как TCP и UDP. ICMP, аналогично IP, работает на межсетевом уровне и фактически является неотъемлемой частью при реализации модели TCP/IP. Стоит отметить, что для разных версий IP используются и разные версии протокола ICMP.

Транспортный уровень (transport layer)

Постоянные резиденты транспортного уровня — протоколы TCP и UDP, они занимаются доставкой информации.

TCP (протокол управления передачей) — надежный, он обеспечивает передачу информации, проверяя дошла ли она, насколько полным является объем полученной информации и т.д. TCP дает возможность двум конечным устройствам производить обмен пакетами через предварительно установленное соединение. Он предоставляет услугу для приложений, повторно запрашивает потерянную информацию, устраняет дублирующие пакеты, регулируя загруженность сети. TCP гарантирует получение и сборку информации у адресата в правильном порядке.

UDP (протокол пользовательских датаграмм) — ненадежный, он занимается передачей автономных датаграмм. UDP не гарантирует, что всех датаграммы дойдут до получателя. Датаграммы уже содержат всю необходимую

информацию, чтобы дойти до получателя, но они все равно могут быть потеряны или доставлены в порядке отличном от порядка при отправлении.

UDP обычно не используется, если требуется надежная передача информации. Использовать UDP имеет смысл там, где потеря части информации не будет критичной для приложения, например, в видеоиграх или потоковой передаче видео. UDP необходим, когда делать повторный запрос сложно или неоправданно по каким-то причинам.

Протоколы транспортного уровня не интерпретируют информацию, полученную с верхнего или нижних уровней, они служат только как канал передачи, но есть исключения. RSVP (Resource Reservation Protocol, протокол резервирования сетевых ресурсов) может использоваться, например, роутерами или сетевыми экранами в целях анализа трафика и принятия решений о его передаче или отклонении в зависимости от содержимого.

Прикладной уровень (application layer)

В модели TCP/IP отсутствуют дополнительные промежуточные уровни (представления и сеансовый) в отличие от OSI. Функции форматирования и представления данных делегированы библиотекам и программным интерфейсам приложений (API) — своего рода базам знаний, содержащим сведения о том, как приложения взаимодействуют между собой. Когда службы или приложения обращаются к библиотеке или API, те в ответ предоставляют набор действий, необходимых для выполнения задачи и полную инструкцию, каким образом эти действия нужно выполнять.

Протоколы прикладного уровня действуют для большинства приложений, они предоставляют услуги пользователю или обмениваются данными с «коллегами» с нижних уровней по уже установленным соединениям. Здесь для большинства приложений созданы свои протоколы. Например, браузеры используют HTTP для передачи гипертекста по сети, почтовые клиенты — SMTP для передачи почты, FTP-клиенты — протокол FTP для передачи файлов, службы DHCP — протокол назначения IP-адресов DHCP и так далее.

21 Билет: Протоколы TCP и UDP. Отличия. Какие типы приложений используют и TCP и UDP?

33 Билет: Опишите составные части архитектуры Docker.

Основные составные части архитектуры Docker — это:

- сервер, содержит сервис Docker, образы и контейнеры. Сервис связывается с Registry, образы — метаданные приложений, запускаемых в контейнерах Docker.
- клиент, применяется для запуска различных действий на сервере Docker.
- registry, используется для хранения образов. Есть публичные, доступные каждому, например, Docker Hub и Docker Cloud.

34 Билет: Расскажите кратко о жизненном цикле контейнера Docker.

Жизненный цикл контейнера:

- Создание контейнера
- Работа контейнера
- Приостановка контейнера
- Возобновление работы контейнера
- Запуск контейнера
- Остановка контейнера
- Перезапуск контейнера
- Принудительная остановка контейнера
- Удаление контейнера

35 Билет: Назовите наиболее важные команды Docker.

Наиболее важные команды Docker:

- build, сборка образа для Docker
- create, создание нового контейнера
- kill. принудительная остановка контейнера
- dockerd, запуск сервиса Docker
- commit, создание нового образа из изменений в контейнере

36 Билет: Опишите функции и случаи применения Docker.

С помощью Docker можно:

- Сделать процесс настройки проще и упростить настройку на уровне инфраструктуры;
- Помочь разработчикам сосредоточиться исключительно на коде, сокращая время разработки и увеличивая продуктивность;
- Усилить возможности отладки с использованием встроенных функций;
- Изолировать приложения;
- Улучшить плотность использования серверов в форме контейнеризации;
- Делать быстрое развертывание на уровне операционной системы.

36 Билет: Приведите необходимые шаги для развертывания докеризированного приложения, сохраненного в репозитории Git.

Шаги, необходимые для развертывания приложения зависят от окружения, основной процесс развертывания будет таким:

- Сборка приложения с использованием Docker build в каталоге с кодом приложения
- Тестирование образа
- Выгрузка образа в Registry
- Уведомление удаленного сервера приложений, что он может скачать образ из Registry и запустить его
- Перестановка порта в прокси HTTP(S)
- Остановка старого контейнера