**Linux 6 Lesson processes CPU info locate find cron**

Информация о процессоре:

CPU Info

Мы с вами говорили, что можем посмотреть информацию о том, какая система у нас стоит на сервере. Необходимо это может быть по разным причинам. Например, если мы вдруг пытаемся ввести команду, а она не проходит в терминале.И тогда нам необходимо посмотреть, какая же система и дистрибутив стоит на сервере.

Делается это командой:

**cat /etc/os-release**

напомню, что командой cat мы выводим на экран текстовый формат того, что находится в файле.

Теперь, давайте поговорим про то, что мы можем увидеть, какой процессор стоит на нашем сервере и заглянуть чуть глубже в глубины нашей удаленной виртуальной машины.

Для того, чтобы увидеть ревизию нашего процессора, или по английски CPU, нам необходимо открыть файл, который находится в папке **proc**:

**cat /proc/cpuinfo**

Что мы тут видим?

В первую очередь, что процессор у нас тут стоит от Intel.

И что количество процессоров у нас - одна штука. Мы видим его частоту, поколение и другую информацию.

Смотрите, тут у нас есть интересная строка, которая называется **bugs**.

Давайте, чтобы долго не искать ее у себя на экране, при помощи пайпа выведем ее себе на экран:

**cat /proc/cpuinfo | grep bugs**

Старайтесь пользоваться этой командой для быстрого поиска информации на экране, это может здорово сэкономить время.

И вот мы видим, какие баги (или уязвимости) присутствуют в нашем процессоре.

Почему мы это видим? Это больная история со всеми процессорами предыдущих поколений.

Производителей Intel и AMD обязали перечислять все возможные найденные баги в описании.

Я думаю вы слышали о некоторых скандалах, когда процессоры Intel позволяли при помощи багов получать доступ ко всем компьютеру. Вот пример:

Специалист по информационной безопасности Крис Домас (Chris Domas) обнаружил серьезную и очень старую уязвимость в процессорах Intel, выпущенных в 1997-2010 годах (до Sandy Bridge). На конференции Black Hat исследователь [продемонстрировал](https://www.blackhat.com/us-15/briefings.html#the-memory-sinkhole-unleashing-an-x86-design-flaw-allowing-universal-privilege-escalation), что баг 18-летней давности позволяет делать с компьютерами, построенными на процессорах Intel x86, что угодно.

Домас проверял найденную уязвимость только на процессорах Intel х86, но полагает, что чипсеты AMD на этой архитектуре точно так же подвержены данному дефекту.

Уязвимости, обнаруженной Домасом, без малого два десятка лет. Баг напрямую связан с функцией, реализованной в архитектуре x86 в 1997 году, и позволяет атакующему получить доступ к режиму System Management Mode (SMM). Доступ к режиму SMM, это, практически «god mode», так как он по умолчанию выше любого уровня доступа в системе. Атакующий с таким доступом способен делать с компьютером жертвы все, что пожелает, включая перепрошивку BIOS и внедрение вредоносного кода в прошивку ПК. Даже если пользователь переустановит операционную систему и сменит жесткий диск, это все равно не поможет ему избавиться от руткита.

Разумеется, для проведения атаки на старый процессор, хакеру сначала понадобится доступ к ПК с правами администратора, но это не так сложно устроить. Зато данная уязвимость может помочь атакующему «окопаться» на компьютере жертвы всерьез и надолго.

Компанию Intel Домас предупредил о баге заранее, до своего выступления на Black Hat, так что проблема исправлена как на процессорах последнего поколения (где она тоже присутствовала), так и для старых чипсетов скоро будет выпущен соответствующий патч. К сожалению, Домас опасается, что патч «вылечит» далеко не все процессоры.

И таких примеров есть довольно много.

Пример с майнингом на системах Intel.

Те баги, которые мы видим у себя на экранах являются довольно критическими, хоть и устранены производителями. Если вам интересно, вы можете смело погуглить про каждый из них и офигеть в полный рост от того, что можно было делать с такими багами в системе. Некоторые могли помочь перехватить дамп (запись) памяти, некоторые позволяли запускать свои процессы на серверах и даже майнить на чужих компьютерах криптовалюту.

Поэтому следите за своими компьютерами и убивайте ненужные процессы не только на линуксе, но и в Windows.

Как правило, первые заплатки, которые выходят после нахождения уязвимостей сильно тормозят производительность процессоров, но со временем драйвера делаются более правильно и скорость работы приходит в норму. Ну, а пока пишутся заплатки, то акции компании падают, народ перебегает от одного производителя к другому, возникают скандалы, интриги, расследования и прочее.

И давайте посмотрим на вашем компьютере информацию о процессоре, но посмотрим правильно, по сисадмински:

**cat /proc/cpuinfo | grep processor**

сколько ядер у вас получается?

Помним, что у нас счет начинается с 0.

Но опять же, мы можем по-сисадмински все посмотреть:

**cat /proc/cpuinfo | grep processors | wc -l**

Теперь мы увидели количество строчек на сисадминском!

Так, с процессами разобрались, с процессором разобрались. Что еще у нас в системе есть такого, что жизненно необходимо?

**Оперативная память.**

**Операти́вная па́мять** ([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) *Random Access Memory, RAM* — память с [произвольным доступом](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF)) — в большинстве случаев [энергозависимая](https://ru.wikipedia.org/wiki/%D0%AD%D0%BD%D0%B5%D1%80%D0%B3%D0%BE%D0%B7%D0%B0%D0%B2%D0%B8%D1%81%D0%B8%D0%BC%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C) часть системы [компьютерной памяти](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%D0%B0%D0%BC%D1%8F%D1%82%D1%8C), в которой во время работы компьютера хранится выполняемый машинный код ([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)), а также входные, выходные и промежуточные данные, обрабатываемые [процессором](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80). ***Оперативное запоминающее устройство*** (*ОЗУ*) — техническое устройство, реализующее функции оперативной памяти. ОЗУ может изготавливаться как отдельный внешний модуль или располагаться на одном кристалле с процессором, например, в однокристальных [ЭВМ](https://ru.wikipedia.org/wiki/%D0%AD%D0%92%D0%9C) или однокристальных [микроконтроллерах](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%BB%D0%B5%D1%80).

пример - Чип М1 от Apple, где в первую очередь неправильно называть его процессором, ведь он выполняет функцию и оперативной памяти и видеопамяти и функцию процессора.

Итак, чтобы посмотреть количество оперативной памяти в нашей системе нам нужно ввести:

**free -m**

Тут мы видим общее количество оперативки и сколько использовано и сколько свободно.

Помните, мы говорили про команду **top**?

Давайте ее заведем и посмотрим на наши процессы.

Что нам покажет эта команда? Диспетчер задач. Как в системе Windows мы нажимаем сочетание клавиш **ctrl+alt+del**, только в Linux мы выводим на экран при помощи **top**.

Что мы видим? Видим, сколько сервер запущен, сколько пользователей сейчас находится, и сколько процессов запущено.

Так же видим zombie, помните, я обещал про него рассказать?

**Проце́сс-зо́мби, зо́мби** — дочерний процесс в Unix-системе, завершивший своё выполнение, но ещё присутствующий в списке процессов операционной системы, чтобы дать родительскому процессу считать код завершения.

Но двигаемся дальше.

Как нам понять, насколько нагружена система и может перегружена ли вообще?

Мы знаем количество ядер нашего процессора. На сервере - 1, у других серверов или компьютеров эта цифра может отличаться.

И тут мы видим **load average**.

Если цифры совпадают с количеством ядер - значит система работает нормально и без перегруза.

Если число превышает число ядер в два раза - значит система нагружена сильно и могут возникать тормоза.

За этим параметром нужно следить и отстреливать те процессы, которые могут сильно грузить процессор. И если эти процессы отстрелить нет возможности (вспомним команду kill), то нам необходима более мощная система и нужно переходить на другой виртуальный сервер.

Вы наверняка сталкивались, что заходя на сайт, вам вылазит сообщение, что сайт перегружен или на нем ведутся работы и он сейчас недоступен. Очень часто это бывает, что из-за увеличенного количества пользователей на этом сайте запускается большее количество процессов и сервер начинает перегружаться данными. Перегреваться и тормозить. Тогда ему необходимо “переехать” на более крутое железо с большим количеством оперативной памяти или с более мощным процессором. И тогда, сайт может быть временно недоступен, ведь переезд занимает некоторое количество времени.

И как системные администраторы понимают, что необходимо переезжать? В первую очередь по загрузке **load average**.

**Теперь на пару минут давайте вернемся к нашим процессам.**

Помните я говорил, что на нашем сервере есть процесс, который запускается вместе с системой и следит за тем, чтобы мы подключались к серверу при помощи ключей **ssh**?

Давайте выведем все наши процессы, а после, отфильтруем по слову ssh:

**ps -ef**

**ps -ef | grep ssh**

Теперь мы видим процессы, в которых есть слово ssh.

Именно тот процесс, который запущен вместе с системой и есть главный. И такие процессы называются **ДЕМОНАМИ**.

**root 4744 1 0 Feb01 ? 00:00:05 /usr/sbin/sshd -D**

**Демон Linux** - это программа, у которой есть определённая уникальная цель. Обычно, это служебные программы, которые незаметно работают в фоновом режиме для того чтобы отслеживать состояние и обслуживать определённые подсистемы и гарантировать правильную работу всей операционной системы в целом.

Демон может следить не только за подключением пользователей на сервере, но и за подключением наших флешек, дисков и так далее.

Традиционно названия демон-процессов заканчиваются на букву **d**, чтобы показать, что этот процесс является демоном, и для различия нормальной компьютерной программы и демона.

В нашем случае - единичка - означает, что сама система его запустила. И он находится в постоянном ожидании.

Давайте сделаем так, чтобы снова выкинуло кого-то.

для этого возьмем родительский процесс, но сначала посмотрим, кто есть на сервере.

**ps -ef | grep ssh**

И если мы сейчас попробуем выкинуть этот родительский процесс, то нам не даст этого сделать.

Как поступить так, чтобы нас выкинуло?

Очень просто. Воспользоваться командой **sudo** (**сокращенно от super user DO**)

**sudo kill** (номер процесса)

Можно указывать несколько идентификаторов процесса для их завершения. Указывать нужно через пробел.

Так же мне напомнили, что есть такая команда как **killall**.

Чем интересна она? А тем, что она работает от имени процесса. Только вы сейчас смотрите и не повторяйте, а то выкинет всех.

например, можно сделать:

**killall php**

И остановятся все процессы, у которых в имени есть слово php. Но у нас их нету, поэтому и останавливать нечего.

**Поиск файлов на сервере.**

В дистрибутивах есть такая программа как locate.

Сразу скажу. В браузерной версии ее нету, потому что она жрёт много ресурсов.

Почему же? Потому что locate создает маленькую базу данных файлов с именами их локаций.

давайте попробуем:

**locate ssh**

Видите, сразу все вывалило нам. Да как быстро!

Минусом Locate является то, что он не мгновенно индексирует файлы. Требуется какое-то время.

Теперь откатываемся на десятки лет назад и пробуем команду find

Эта команда обладает огромным количеством функций.

откроем наши учебные терминалы:

https://bellard.org/jslinux/vm.html?url=alpine-x86.cfg&mem=192

## **ОСНОВНАЯ ИНФОРМАЦИЯ О FIND**

Команда find - это одна из наиболее важных и часто используемых утилит системы Linux. Это команда для поиска файлов и каталогов на основе специальных условий. Ее можно использовать в различных обстоятельствах, например, для поиска файлов по разрешениям, владельцам, группам, типу, размеру и другим подобным критериям.

Утилита find предустановлена по умолчанию во всех Linux дистрибутивах, поэтому вам не нужно будет устанавливать никаких дополнительных пакетов. Это очень важная находка для тех, кто хочет использовать командную строку наиболее эффективно.

Команда find имеет такой синтаксис:

**find [папка] [параметры] критерий шаблон [действие]**

**Папка** - каталог в котором будем искать

**Параметры** - дополнительные параметры, например, глубина поиска, и т д

**Критерий** - по какому критерию будем искать: имя, дата создания, права, владелец и т д.

**Шаблон** - непосредственно значение по которому будем отбирать файлы.

## **ОСНОВНЫЕ ПАРАМЕТРЫ КОМАНДЫ FIND**

Я не буду перечислять здесь все параметры, рассмотрим только самые полезные.

* **-P** - никогда не открывать символические ссылки.
* **-L** - получает информацию о файлах по символическим ссылкам. Важно для дальнейшей обработки, чтобы обрабатывалась не ссылка, а сам файл.
* **-maxdepth** - максимальная глубина поиска по подкаталогам, для поиска только в текущем каталоге установите 1.
* **-depth** - искать сначала в текущем каталоге, а потом в подкаталогах.
* **-mount** искать файлы только в этой файловой системе.
* **-version** - показать версию утилиты find.
* **-print** - выводить полные имена файлов.
* **-type f** - искать только файлы.
* **-type d** - поиск папки в Linux.

Как это работает?

пишем:

**find /home -type f -name "\*.\*"**

где f - это файл, бывает еще d - директория.

Специфика работы find - он как честный гражданин сам заходит и проверяет наличие файлов. Поэтому в первый поиск это может занять чуть больше времени чем по второй и далее.

То есть, если файлов у нас много, например, терабайты, то времени это займет немало. Будьте готовы к этому.

Именно поэтому чаще используют locate, который имеет проиндексированную базу данных и делает поиск оттуда. Что занимает меньше времени. Ведь легче напрячь базу данных, чем заходить в каждую папку отдельно и напрягать жесткий диск.

Давайте откроем наши учебные терминалы:

<https://bellard.org/jslinux/vm.html?url=alpine-x86.cfg&mem=192>

Помните, как мы быстро нашли все названия файлов при помощи **Locate**?

давайте сделаем тоже самое в этом терминале:

locate / -name ssh

Но многое поменяется, если мы попробуем найти через **find**:

find / -name ssh

Терминал начал долгий поиск и выводит пути по мере их нахождения.

создадим папку и в ней подпапку, а в подпапке файл:

mkdir /test

mkdir /test/1

touch /test/1/test.txt

теперь давайте попробуем найти наш файл командой:

find / -name test.txt

что мы видим? Терминал подвесился и ищет.

Нажимаем ctrl+c

Это говорит нам о том, что система пытается зайти в каждую папку и понюхать каждый файл на предмет совпадения. И для того, чтобы найти его быстрее, нам необходимо воспользоваться программой Locate, но ее нету на нашем учебном терминале. Поэтому мы или ждем, или указываем чуть правильный путь.

find /test -name test.txt

Можно указать глубину поиска через параметры:

**find /home -maxdepth 2 -type f -name "\*.\*"**

Хорошая статья по этому поводу насчет поиска:

<https://losst.ru/komanda-find-v-linux>

**А теперь перейдем к cron.**

## **Что такое Cron и crontab?**

Если в двух словах, то Cron – это планировщик задач. Если подробнее, то это утилита, позволяющая выполнять скрипты на сервере в назначенное время с заранее определенной периодичностью.

К примеру, у вас есть скрипт, который собирает какие-либо статистические данные каждый день в 6 часов вечера. Такие скрипты называют «заданиями», а их логика описывается в специальных файлах под названием сrontab.

crontab – это таблица с расписанием запуска скриптов и программ, оформленная в специальном формате, который умеет считывать компьютер. Для каждого пользователя системы создается отдельный crontab-файл со своим расписанием. Эта встроенная в Linux утилита доступна на низком уровне в каждом дистрибутиве.

В Linux-дистрибутивах с поддержкой systemd Cron считается устаревшим решением, его заменили утилитой systemd.timer. Ее предназначение и функциональность не отличается, но фактически частота использования Cron все еще выше.

### **Для чего обычно используют Cron?**

Обычно Cron заставляют повторять вполне очевидные задачи в духе регулярного создания резервных копий данных. Но это не все.

Некоторые пользователи с помощью планировщика корректируют системное время. На многих компьютерах оно настраивается через Network Time Protocol. А так как этот протокол настраивает только время ОС, время, установленное для «железа», может отличаться. Cron позволяют регулярно корректировать время, установленное для аппаратного обеспечения, в соответствии со временем ОС.

Еще один популярный сценарий – создание оповещений, появляющихся каждое утро и рассказывающих о состоянии компьютера. В эти сообщения может входить любая полезная для пользователя информация.

Cron иногда работает даже без ведома пользователя. Эту утилиту используют такие сервисы, как Logwatch, logrotate и Rootkit Hunter. Повторяющиеся задачи они настраивают, как и пользователи, через Cron.

С помощью Cron пользователи автоматизируют самые разные задачи, сокращая вмешательство системного администратора в работу сервера.

Давайте посмотрим на практике, как это работает.

Говоря простым языком - cron - это планировщик, который запускает по расписанию какую-то программу или какой-то скрипт.

Штука очень надежная, лично сталкивался с несколькими огромными компаниями, которые используют крон для обмена финансовой информацией при помощи этого планировщика. Он запускался через определенный промежуток времени и проверял, появились ли новые файлы в системе.

Сам планировщик называется **cron**. Расписание планировщика называется **crontab**.

(заходим на сервер и будем смотреть)

**crontab -l**

показывает список задач текущего пользователя.

**touch /tmp/script2.sh /tmp/output2.txt**

**echo -e '#!/bin/bash\n date\n echo "it works"' > /tmp/script2.sh**

**chmod +x /tmp/script2.sh**

Что мы делаем? Мы файлом создаем будущий файлик скрипта и создаем будущий файл, куда этот скрипт будет что-то писать:

**touch /tmp/script2.sh /tmp/output2.txt**

Дальше применяем читерство:

**echo -e '#!/bin/bash\n date\n echo "it works"' > /tmp/script2.sh**

Все помнят команду, с помощью которой мы выводим на экран, echo, так вот у него есть ключик -е, который интерпретирует **\n \s** и другие штуки в разные конечные символы. В нашем случае, бэк **\n** будет интерпретирован как перенос строки.

По итогу мы передадим все, что у нас в одинарных кавычках:

**#!/bin/bash\n date\n echo "it works"**

с переносом строки в новый файл:

**/tmp/script2.sh**

Это еще один способ создания скриптов, одной строкой, не заходя в редактор нано или другие.

теперь, если мы на него посмотрим, мы увидим:

**cat /tmp/script2.sh**

#!/bin/bash

date

echo "it works"

[ec2-user@ip-172-31-39-17 ~]$

Этот скрипт является полноценным. Видите, у него присутствует и шебанг и прочее.

Чтобы он стал исполняемым, мы меняем ему права на выполнение:

**chmod +x /tmp/script2.sh**

единственное, что я сейчас сделаю, я его скопирую, чтобы его потом не убили перезаписью:

**cp /tmp/script2.sh /tmp/chucron.sh**

**cp /tmp/script2.sh /tmp/chumak.sh**

Теперь давайте посмотрим, что у нас тут лежит:

**ls -la /tmp**

**Видим кучу скриптов и непонятных файлов. Значит записало.**

**[ec2-user@ip-172-31-39-17 ~]$ ls -la /tmp**

**total 820**

**drwxrwxrwt 8 root root 286 Feb 22 12:46 .**

**dr-xr-xr-x 19 root root 269 Feb 21 14:40 ..**

**drwxrwxrwt 2 root root 6 Jan 11 12:04 .ICE-unix**

**drwxrwxrwt 2 root root 6 Jan 11 12:04 .Test-unix**

**drwxrwxrwt 2 root root 6 Jan 11 12:04 .X11-unix**

**drwxrwxrwt 2 root root 6 Jan 11 12:04 .XIM-unix**

**-rw------- 1 ec2-user ec2-user 12288 Feb 14 22:18 .crontab.on2Swp.swp**

**drwxrwxrwt 2 root root 6 Jan 11 12:04 .font-unix**

**-rwxrwxr-x 1 ec2-user ec2-user 36 Feb 14 21:48 cron.sh**

**-rw-r--r-- 1 ec2-user ec2-user 810732 Feb 14 21:33 output.txt**

**-rw-rw-r-- 1 ec2-user ec2-user 0 Feb 22 12:32 output2.txt**

**-rwxrwxr-x 1 ec2-user ec2-user 36 Feb 14 21:37 script.sh**

**-rwxrwxr-x 1 ec2-user ec2-user 35 Feb 22 12:34 script2.sh**

**drwx------ 3 root root 17 Jan 11 12:04**

Вижу, что кто-то уже работает, молодцы)

Теперь давайте скажем нашему планировщику - а запусти ты нам этот скрипт, с интервалом в минуту.

Для того, чтобы это сделать мы запускаем редактор crontab:

**crontab -e**

давайте удалим, все, что есть в файле. Напомню, что удалять можно построчно командой **dd**

Сохранимся и выйдем:

**ZZ**

теперь снова откроем:

**crontab -e**

и добавим странное сочетание (напомню, чтобы войти в режим редактирования нужно нажать **i**)

(можно не перезаходить, но я хочу, чтобы мы повторили команды)

набираем странное сочетание:

**\* \* \* \* \* \* (через пробел)**

\* \* \* \* \* /tmp/chumak.sh >> /tmp/output2.txt

выходим с сохранением.

теперь давайте, чтобы он у нас был перед глазами:

**[ec2-user@ip-172-31-39-17 ~]$ crontab -l**

**\* \* \* \* \* /tmp/chumak.sh >> /tmp/output2.txt**

**Теперь задание какое у него есть:**

**/tmp/chumak.sh дописать в /tmp/output2.txt**

Теперь, что же значат наши звездочки. В crontab каждая звездочка означает определенное время. А именно:

<https://crontab.guru/>

Все заходим на сайт и смотрим.

Выглядит страшно, не правда-ли? Но сверху есть подсказка и она нам может помочь.

Мы же выбрали все звездочки, как в ГТА. Но не бойтесь, полиции не будет)

Звездочки означают когда делать, а после них - что делать.

Тоесть у нас будет дозаписывать в файл слова It works каждую минуту.

Давайте убедимся:

**cat /tmp/output2.txt**

**Tue Feb 22 13:31:01 UTC 2022**

**it works**

**Tue Feb 22 13:32:01 UTC 2022**

**it works**

**Tue Feb 22 13:33:01 UTC 2022**

**it works**

**Tue Feb 22 13:34:01 UTC 2022**

**it works**

**Tue Feb 22 13:35:01 UTC 2022**

**it works**

**Tue Feb 22 13:36:01 UTC 2022**

**it works**

**Tue Feb 22 13:37:01 UTC 2022**

**it works**

**Tue Feb 22 13:38:01 UTC 2022**

**it works**

**Tue Feb 22 13:39:01 UTC 2022**

**it works**

**Tue Feb 22 13:40:01 UTC 2022**

**it works**

**Tue Feb 22 13:41:01 UTC 2022**

**it works**

вот пример простейшего скрипта, который работает в планировщике.

И давайте вернемся к нашему редактору. Всем понравился редактор nano, но по умолчанию мы видим тут vi. Не страшно.

Что необходимо сделать?

Нужно перед открытием crontab сделать:

export EDITOR=nano

То есть, что мы сделали. У нас есть переменная EDITOR (помните, про джентльменское соглашение писать переменные большими буквами?)

По умолчанию там стоит Vi, но мы перезаписываем эту переменную и говорим открывать это в Nano.

ссылка на Хабр по крону:

<https://habr.com/ru/company/badoo/blog/468061/>

**Не забывайте про копирование средней кнопкой мыши! (у кого она есть)**