Потренироваться, не домашка:

Сделать файл скрипта, который выполнит:

Создаст файл и запишет в него информацию из команды df  -h о дисковом пространстве, но оставит в информации для записи лишь последний столбец в последней строчке, где заменит слово user на ваше имя.(можно сделать последнюю строчку двумя способами через grep -w и через tail)

Выведет на экран содержимое созданного файла.

Что понадобится:

awk

sed

Решение:

#!/bin/bash

df -h | tail -1 | awk '{print $6}' | sed 's/user/ivan/g' > test.txt

cat test.txt

\*решение покрашено белым текстом, постарайтесь сделать самостоятельно, а потом выделить и перепровериться (=

**Шпаргалка по поиску файлов:**

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

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

Статьи огромные, но очень толковые, советую к прочтению.

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.

## 

## 

## **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**

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**

**(пример с VI - этот редактор открывается по умолчанию)**

**Мы все делали в nano**

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/script2.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/script.sh >> /tmp/output2.txt**

**Теперь задание какое у него есть:**

**/tmp/script2.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**

ссылка на Хабр по крону:

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

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

**архиватор TAR**

### **tar** - наиболее распространенный архиватор, используемый в Linux-системах. Сам по себе tar не является архиватором в привычном понимании этого слова, т.к. он самостоятельно не использует сжатие. В то же время, многие архиваторы (например, [Gzip](http://en.wikipedia.org/wiki/Gzip) или [bzip2](http://en.wikipedia.org/wiki/bzip2)) не умеют сжимать несколько файлов, а работают только с одним файлом или входным потоком. Поэтому чаще всего эти программы используются вместе. tar создает несжатый архив, в который помещаются выбранные файлы и каталоги, при этом сохраняя некоторые их атрибуты (такие как права доступа). После этого полученный файл \*.tar сжимается архиватором, например, gzip. Вот почему архивы обычно имеют расширение .tar.gz или .tar.bz2 (для архиваторов gzip и bzip2 соответственно)

Мы можем архивироваться со сжатием и без. Что это значит? Без сжатия мы просто сделаем архив, в который наш архиватор запихнет некоторые файлы, которые нам нужны.

Но если применим специальный ключ, то запустится не только программа архивации, но и запустится процесс сжатия файлов.

Давайте заархивируем некоторые файлы, но прежде мы их создадим:

date > /tmp/1.txt

ps -ef > /tmp/2.txt

теперь зайдем в эту папку:

cd /tmp

И попробуем заархивировать эти файлы, но укажем, куда мы сразу запишем наш новый архив (в папку opt, например):

tar -czf /opt/arhive.tar.gz \*.txt

Что мы тут сделали? Для начала мы вызвали архиватор tar. Потом ключами указали, что мы будем делать. Напомню, чтобы вызвать подсказку по tar нам нужно в терминале написать:

tar –help (два минуса)

И как мы видим, тут мы указываем:

* с - означает, что мы создаем архив
* z - означает, что мы ужимаем файлы при помощи gzip
* f - указываем имя файла архива

Дальше мы указываем путь, в который попадет наш архив. В данном случае - это папка opt. ВАЖНО! Предварительно нужно находиться в папке с файлами, которые мы будем архивировать и ужимать.

Дальше, через пробел говорим архиватору, чтобы тот нашел в папке все файлы с расширением txt и добавил их в архив.

Теперь давайте попробуем разархивировать наши файлы в другое место и проверить.

Для разархивирования нам понадобится тот же архиватор tar:

tar -xzf /opt/archive.tar.gz -C /tmp/1

Ругается. Не понимает нашей папки 1 в папке /tmp

Создадим папку 1:

mkdir /tmp/1

Перезапускаем команду разархивации.

Что мы сделали?

1. вызвали наш tar
2. указали ключи:

* x - извлекаем
* z - используем gzip (так как мы видим, что файл не только в архиве, но и ужат)
* f - работаем с файлами

1. Указываем папку, из которой извлекаем наш архив и какой файл разархивируем
2. используем ключ -С (большое, чтобы указать другое месторасположение)
3. указываем саму папку, куда хотим разархивировать файлы.

теперь проверим наш новосозданный архив:

ls -la /opt/

и проверим наши файлы в новой папке после разархивации:

ls /tmp/1

cat /tmp/1/2.txt

**Homework CRON**

Создайте папку со своим именем в /opt/37\_38\_MA на сервере linux.telran-edu.de. В папке с Вашим именем создайте скрипт task\_NAME.sh, где NAME - Ваше имя.

Скрипт должен создавать по 10 файлов с порядковым номером и датой создания.  1\_16\_02\_18 , 2\_16\_02\_18 , 3\_16\_02\_18 .. 10\_16\_02\_18

Создайте в планировщике новое задание, которое должно будет запускать Ваш скрипт Каждые 15 минут.

Что пригодится:

date +"%H\_%M\_%S"

touch

export EDITOR=nano

crontab -l

crontab -e

http://crontab.guru/

Если удобно, то вот однострочник, создающий простейший скрипт в /tmp , который надо будет редактировать дальше:

echo -e `#!/bin/bash\n date\n echo “it works!“` > /tmp/script.sh