**урок 7 практика**

## *копирование файлов при помощи скрипта:*

создадим файл вручную в папке tmp и скопируем его в папку opt при помощи скрипта:

#!/bin/bash

cp /tmp/\*.txt /opt

усложним задачу. Давайте создадим файл, допишем в него дату и перенесем его в папку opt/3/ c переименовыванием. Так же напишем, что при удачном переносе нам на экран выведется Done!:

#!/bin/bash

touch /tmp/2.txt

date >> /tmp/2.txt

mkdir -p /opt/3

mv /tmp/2.txt /opt/3/newfile.txt

echo "Done!"

## 

## *Как запускать скрипты без дополнительных символов:*

все довольно просто. Нужно закинуть файл скрипта с правильным названием в папку /bin и дать ей права на исполнение.

\*иногда можно сделать копирование файла скрипта непосредственно в самом скрипте в папку /bin и дать права на исполнение скрипта уже из папки /bin

Выглядеть это будет следующим образом:

cp /tmp/ФАЙЛ\_СКРИПТА /opt

chmod +x /opt/ФАЙЛ\_СКРИПТА

И теперь можно запускать скрипты непосредственно из любого места.

Давайте посмотрим на примере с командой, которая показывает нам наш айпишник. Создадим скрипт, который будет показывать наш айпи адрес:

curl ifconfig.co

файл скрипта будет выглядеть так:

#!/bin/bash

curl ifconfig.co

только давайте добавим к нему предложение - это твой IP:

для этого нужно в скрипт добавить:

echo “Твой айпи”

localhost:/tmp# cp myip.sh /bin/

localhost:/tmp# myip.sh

sh: myip.sh: Permission denied

localhost:/bin# chmod +x /bin/myip.sh

localhost:/bin# myip.sh

Твой айпи:

107.170.233.148

не забываем дать права на исполнение именно в той папке, где находится файл скрипта, а именно /bin/

chmod +x ~/bin/myip.sh

## 

## 

## 

## 

## *Получение данных от пользователя*

*Ключи и параметры командной строки — это отличный способ получить данные от того, кто пользуется скриптом, однако в некоторых случаях нужно больше интерактивности.*

*Иногда сценарии нуждаются в данных, которые пользователь должен ввести во время выполнения программы. Именно для этой цели в оболочке bash имеется команда read.*

*Эта команда позволяет принимать введённые данные либо со стандартного ввода (с клавиатуры), либо используя другие дескрипторы файлов. После получения данных, эта команда помещает их в переменную:*

*#!/bin/bash*

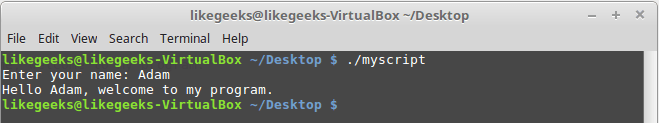
*echo -n "Enter your name: "*

*read name*

*echo "Hello $name, welcome to my program."*

*Обратите внимание на то, что команда echo, которая выводит приглашение, вызывается с ключом -n. Это приводит к тому, что в конце приглашения не выводится знак перевода строки, что позволяет пользователю скрипта вводить данные там же, где расположено приглашение, а не на следующей строке.*

*давайте не будем голословными, а попробуем убрать ключ -n и снова запустим файл.*

**

*Обработка пользовательского ввода*

При вызове read можно указывать и несколько переменных:

#!/bin/bash

read -p "Enter your name: " first last

echo "Your data for $last, $first…"

Итак, прежде всего, нам необходимо разобраться в структуре скрипта. Она не представляет собой ничего сложного. По большому счету, простейший скрипт – просто перечисление системных команд. Например:

echo This is just example

whoami

вывести на экран строку “Простой скрипт”. Написать “Вы:” (кто вы в системе имя зарегистрированного пользователя). Написать “ваша система запущена” (указать время когда). Показать дату. Написать “Завершение”

#!/bin/bash

echo Простой скрипт # Команда echo выводит сообщение на экран

echo Вы:

whoami # whoami показывает имя зарегистрированного пользователя

echo Ваша система стартовала

uptime # uptime показывает время включения машины

echo Сегодня

date # date показывает текущую дату и время

echo Пока все

exit

Сохраним этот файл под именем script.sh. Выполним команду чтобы сделать скрипт исполняемым.

chmod +rx script.sh

Результатом выполнения скрипта будет:

./script.sh

Простой скрипт

Вы:

voland

Ваша система стартовала

14:38:46 up 1:48, 2 users, load average: 0.47, 0.43, 0.41

Сегодня

Вск Фев 26 14:38:46 MSK 2006

Пока все

Давайте создадим скрипт, который будет сравнивать строки или текст, который мы вводим в терминале. Я уже подбросил в песочницу русский язык, так что будет интересно!

Откроем песочницу и создадим файл в папке /tmp с названием script.sh

Теперь откроем файл в редакторе нано и введем скрипт:

#!/bin/bash

read -p "введите первую строку: " STR1

read -p "введите вторую строку: " STR2

if [[ "$STR1" == "$STR2" ]]; then

echo "строки эквивалентны!"

else

echo "строки не эквивалентны!"

fi

If-предложения в Bash очень похожи на аналогичные в других языках. Они следуют базовой форме:

if [[ "$some\_variable" == "good input" ]]; then

echo "You got the right input."

elif [[ "$some\_variable" == "ok input" ]]; then

echo "Close enough"

else

echo "No way Jose."

fi

Дополнительный синтаксис в виде слова then, а также слегка странное закрывающее ключевое слово fi придает предложению немного экзотический вид, но в основе все то же самое, что и в других языках:

*Если (if) что-то является истиной, тогда (then) выполни вот это. В противном случае проверяй другие условия по порядку и делай то же самое. Если ни одно из условий не сработало, выполни последнее указание.*

Вы можете отбросить одну или все ветки elif, а также ветку else, если в них нет нужды!

### **Как насчет двойных квадратных скобок?**

Довольно странно, однако [[ двойные квадратные скобки ]] и ((двойные круглые скобки)) это не совсем команды. Они представляют собой ключевые слова языка Bash, в результате чего они ведут себя немного более предсказуемо. Тем не менее, в зависимости от своего содержимого, они по-прежнему возвращают код выхода.

[[ Двойные квадратные скобки ]] работают в целом так же, как и [одинарные квадратные скобки], но имеют дополнительные возможности вроде лучшей поддержки регулярных выражений.

**homework**

Сделать скрипт script.sh в папке /tmp , который сделает:

выведет строчку со словами:

Привет студент!

попросит ввести Ваше имя (используя ключик -n для НЕпереноса на новую строку)

выведет строчку:

"Терминал рад тебе $name, добро пожаловать в систему."

создаст папку **test** в каталоге **tmp**

создаст файл под названием **mydate.txt** в папке /**tmp/test**

запишет в новосозданный файл **mydate.txt** текущее время (**не просто дату**)

выведет на экран надпись “**Сохраняем данные**” и начнет отсчет до **10** с интервалом в полсекунды.

выведет на экран “Данные сохранены, продолжаю работу”

дозапишет в файл **mydate.txt** все сведения о состоянии нашего диска в человеческом виде.

скопирует файл **mydate.txt** в новую подпапку **mydate** в папке /**opt**/ с переименованием файла в **newmydate.txt**

снова сделает интервал ожидания, но уже с отсчетом в пять раз по одной секунде и выведет на экран фразу “Все сделано, Босс”

**\* сделать файл исполняемым из любого места в системе.**

что пригодится:

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

for run in {a..b}

sleep x

echo $run

read name