**Скрипт (англ. script - сценарий) - это небольшая программа, которая содержит последовательность действий, созданных для автоматического выполнения задачи.**

Скрипты пишутся не только на разных языках программирования, но и в терминале bash в Linux.

Делается это для упрощения работы и удобства.

Одна из самых полезных возможностей **bash**-**скриптов** — это возможность извлекать информацию из вывода команд и назначать её переменным, что позволяет использовать эту информацию где угодно в файле сценария.

Давайте попробуем разобраться на практике со скриптами.

Открываем наши учебные терминалы. Почему учебные? Потому что нам нужно потренироваться там, где мы не можем принести вред.

<https://bellard.org/jslinux/index.html>

Пишем:

nano script.sh

нам сейчас неважно, где мы находимся, нам нужно понять сам принцип.

Почему мы не создаем файл при помощи команды touch? Да потому что он создастся пустым, а нам нужно сделать так, чтобы в нем была информация.

Так же напомню, что файл, который мы создадим при помощи редактора nano создастся только тогда, когда мы будем выходить из редактора с сохранением. А выходить с сохранением нужно ctrl+x

Зашли и открыли. И что нам теперь нужно? А нужно написать скрипт.

#!/bin/bash

Давайте разбираться по порядку.

#!

Эти два символа называют по-разному. Решетка\восклицательный знак, диез\восклицательный знак и прочее.

Но в нашем случае это ШЕБАНГ (тот случай, когда можно написать по-русски).

Что это? Говоря простым языком - эта штука показывает, чем же открывать наш файл.

Например во всем известном Windows ассоциации файлов с программами с которыми они работают осуществляется при помощи расширения.

В Linux чуть по-другому. Файл является исполняемым, если у него есть права на исполнение. То есть если ему присвоили права на исполнение.

А чем открывать этот файл мы передаем первой строкой в этом самом файле. И инструкция о том, чтобы система понимала, чем именно нужно открывать файл начинается с ШЕБАНГА.

Что мы видим в первой строчке? То, что система должна понять, что скрипт (или сценарий) будет запускаться при помощи bash, который находится в папке /bin.

**Bash нужен для приема команд пользователя и их отправки операционной системе для последующей обработки.**

Говоря простым языком - мы пишем команду, а bash нам возвращает выполненную команду или результат действия на экран. Или не возвращает и просто отрабатывает. И терминал ошибок не прощает, так что будьте внимательны.

А тут, в нашем конкретном случае мы пишем в начале файла, что нам необходимо запустить все, что дальше в файле bash-ем.

Это будет набор инструкций, где каждая инструкция будет идти с новой строки.

Вместо /bash после /bin может идти /pyton чтобы все открывалось в интерпретаторе пайтона. Либо java.

**Интерпрета́ция** — построчный анализ, обработка и выполнение исходного кода программы или запроса, в отличие от компиляции, где весь текст программы, перед запуском анализируется и транслируется в машинный или байт-код без её выполнения.

В принципе системе все-равно как и чем запускать код. Мы будем рассматривать только bash оболочку.

Почему стоит ее рассматривать? Bash есть практически во всех современных дистрибутивах. Кроме очень экстремальных, где каждый байт имеет значение. Но тогда там будет облегченная версия (кажется, куда уже меньше и проще, правда?).

Как это понять? Чем меньше букв в названии, тем более облегченная оболочка.

Но как узнать в какой оболочке мы работаем? Все просто. Если мы нажимаем на **TAB** и нам подсказывается слово, то будьте уверены - перед вами bash - полная версия.

Если вы нажимаете **TAB** и у вас нет подсказок, то перед вами облегченная версия оболочки. И оболочка может называться **ash.**

**cat /etc/os-release - проверить Версию Linux**

**Бывают настолько обрезанные дистрибутивы, что в них даже не показывает твой настоящий путь. Тоесть ты не видишь где ты находишься и тебе приходится постоянно перепроверяться командой pwd (которая укажет твой путь). Не говоря уже о подсказках через TAB.**

Но такие системы встречаются довольно редко и работать с ними некоторым не придется. Однако, мы с вами с ними будем сталкиваться в процессе обучения, когда будем говорить о “контейнерах”. Когда каждый скрипт может быть заперт в своем контейнере и контейнер видит себя как отдельную операционную систему. И там уже, для экономии места и ресурсов, иногда урезается **bash** и бывает самый простой пример оболочки **sh**.

Может возникнуть вопрос, а что если мы написали в файле **bash**, а его нету? Тогда система должна сама перенаправить на выполнение в оболочку **ash**.

Но это уже особенности определенных операционных систем.

Пока у нас есть **bash** и мы будем с ним работать.

И поэтому сценарии, с которыми мы будем работать так и называются **bash** скрипты.

Многие это даже называют программированием на bash, хотя это и не является настоящим программированием.

Ведь это набор инструкций с использованием переменных, которые вдруг нам могут понадобиться.

А теперь вернемся к учебному терминалу и попробуем написать парочку команд:

#!/bin/bash

echo Hello

date

Напомню, что команда **echo** выводит на экран текст, который мы напишем, а команда **date** - дату

Теперь выйдем и сохранимся:

**ctrl+x**

И проверим командой **ls**, что файл сохранился.

Вот, мы видим свой файл:

localhost:~# ls

bench.py hello.c hello.js readme.txt script.sh

И чтобы убедиться, что у нас файл содержит команды давайте введем команду:

localhost:~# cat script.sh

#!/bin/bash

echo Hello

date

Теперь введем команду:

localhost:~# ls -la

total 44

drwxr-xr-x 5 root root 263 Jan 9 2021 .

drwxrwxrwx 21 root root 461 Feb 15 14:45 ..

-rw------- 1 root root 40 Feb 15 14:45 .ash\_history

drwx------ 3 root root 61 Jul 6 2020 .cache

drwx------ 5 root root 124 Jul 6 2020 .mozilla

drwxr-xr-x 4 root root 202 Jul 6 2020 .wine

-rw-r--r-- 1 root root 114 Jul 5 2020 bench.py

-rw-r--r-- 1 root root 76 Jul 3 2020 hello.c

-rw-r--r-- 1 root root 22 Jun 26 2020 hello.js

-rw-r--r-- 1 root root 151 Jul 5 2020 readme.txt

-rw-r--r-- 1 root root 29 Feb 15 14:43 script.sh

И что мы видим? Наш файл, который мы создали не более чем текстовик. И его можно только прочитать и записать. И то, только суперадмину. Что нам делать? Ааааа! Паника!

Это еще не скрипт, а простой файл.

Что нам нужно сделать, чтобы он стал полноценным скриптом?

Что думаете, что нужно сделать?

Помните, мы затрагивали тему прав на файлы? Что есть некая команда chmod, которая позволяет раздавать права направо и налево?

**Итак, мы хотим добавить владельцу возможность исполнения.**

localhost:~# chmod u+x script.sh

Что это значит?

**chmod - change modification**

**u - user**

**+x - дать права на выполнение.**

**вместо U может быть:**

владельцу (u), группе (g), остальным (o) или всем (a). Можно указать как одну, так сразу несколько категорий.

Вторая группа означает запрещение, разрешение или назначение права и всегда состоит из одного символа: -, + или =.

В третьей группе перечисляется изменяемые права: чтение (r), запись (w), исполнение (x). Можно указывать сразу несколько прав.

Теперь проверим:

ls -la

total 44

drwxr-xr-x 5 root root 263 Jan 9 2021 .

drwxrwxrwx 21 root root 461 Feb 15 14:53 ..

-rw------- 1 root root 68 Feb 15 14:55 .ash\_history

drwx------ 3 root root 61 Jul 6 2020 .cache

drwx------ 5 root root 124 Jul 6 2020 .mozilla

drwxr-xr-x 4 root root 202 Jul 6 2020 .wine

-rw-r--r-- 1 root root 114 Jul 5 2020 bench.py

-rw-r--r-- 1 root root 76 Jul 3 2020 hello.c

-rw-r--r-- 1 root root 22 Jun 26 2020 hello.js

-rw-r--r-- 1 root root 151 Jul 5 2020 readme.txt

-rwxr--r-- 1 root root 29 Feb 15 14:43 script.sh

И у нас у владельца есть право на исполнение.

Это мы можем видеть по зеленому цвету файла. Если цвет белый - то это значит, что перед нами текстовик.

**Вообще, что касается цвета файлов:**

* **зеленый**: Исполняемые файлы.
* **белый**: Обычный файл.
* **синий**: Каталоги или папки.
* **Небесный**: Символическая ссылка.
* **красный**: Сжатые файлы (.tar, .gz, .zip, .rpm).
* **фуксин или фиолетовый**: Файлы изображений (.jpg, gif, bmp, png, tif)

## **Установка прав с помощью чисел**

Способ использования команды chmod, когда предоставляемые права выражаются одной восьмеричной цифрой для каждой категории пользователей, несколько проще с точки зрения записи. Однако надо помнить, что обозначают цифры:

7 – разрешены чтение, запись, исполнение

6 – разрешены чтение и запись

5 – разрешены чтение и исполнение

4 – разрешено только чтение

0 – ничего не разрешено

В первом аргументе chmod указываются три цифры: первая обозначает права владельца, вторая – группы, третья – остальных.

Пример с цифрами:

chmod 744 script.sh

localhost:~# ls -la

total 44

drwxr-xr-x 5 root root 263 Jan 9 2021 .

drwxrwxrwx 21 root root 461 Feb 15 15:04 ..

-rw------- 1 root root 96 Feb 15 15:04 .ash\_history

drwx------ 3 root root 61 Jul 6 2020 .cache

drwx------ 5 root root 124 Jul 6 2020 .mozilla

drwxr-xr-x 4 root root 202 Jul 6 2020 .wine

-rw-r--r-- 1 root root 114 Jul 5 2020 bench.py

-rw-r--r-- 1 root root 76 Jul 3 2020 hello.c

-rw-r--r-- 1 root root 22 Jun 26 2020 hello.js

-rw-r--r-- 1 root root 151 Jul 5 2020 readme.txt

-rwxr--r-- 1 root root 29 Feb 15 14:43 script.sh

Разница заключается в том, что при помощи цифр мы передает то состояние, которое мы хотим. То есть у владельца - все возможное, у группы чтение исполнение. Мы сразу передаем то состояние, которое мы хотим видеть у файла.

Если мы будем делать первым способом, то нам нужно несколько раз указывать, что и кому мы хотим дать. **Второй способ с цифрами чуть быстрее.**

Есть еще один трюк, который не совсем безопасный, но очень быстрый.

localhost:~# chmod +x script.sh

Этим мы дадим всем права на запуск файла.

Теперь вернемся к нашему скрипту, который написан в файлике.

Есть 3 способа запустить наш скрипт.

1. Если мы находимся в той же папке, что и наш скрипт, то:

localhost:~# ./script.sh

тут мы указываем, где взять в нашей папке то, что запустить.

Если мы введем название скрипта (файла), просто в терминал, то нас скажет, что файл не найден. Ведь его не ввели в переменные.

echo $PATH

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

Помните мы говорили про переменные?

Так вот,запуская через точку слеш мы говорим системе, что нам необходимо в этой же папке взять наш файл и запустить.

1. Запустить через полный путь:

localhost:~# pwd

/root

localhost:~# /root/script.sh

Hello

Tue Feb 15 15:17:46 UTC 2022

1. запустить через bash:

localhost:~# bash script.sh

Hello

Tue Feb 15 15:18:57 UTC 2022

Но третий не совсем правильный. Скорее запасной. Все дело в том, что мы делаем бесполезную ссылку на то, чем мы этот скрипт запускаем.

localhost:~# cat script.sh

#!/bin/bash

echo Hello

date

Помните, вначале мы когда писали скрипт, мы говорили системе, чем нужно запускать скрипт?

И если мы в терминале пишем **bash**, то тем самым исключаем нашу первую строчку из текстового файла. А значит выполнили лишнее действие и потратили больше времени. Не совсем правильно.

И в этом виде интерпретатор будет относиться к текстовому файлу не как к скрипту, а как к простому текстовому файлу с двумя командами.

Однако есть и бонус в этой записи. Если, например, в нашем файле нету строки #!/bin/bash, то скрипт все равно отработает. И еще, если мы принудительно запускаем bash-ом, то файл может быть и не исполняемым. То есть есть свои плюсы.

Но пользоваться этим нужно осторожно и понимая когда и зачем.

Этим можно сэкономить время, но это не является полноценным скриптом.

Окей, запустили, теперь давайте мы модифицируем наш скрипт.

Зайдем еще раз в наш файл:

nano script.sh

#!/bin/bash

CITY=Berlin

echo Hello from $CITY

date

Что я сейчас сделал?

Сохранимся и выйдем.

И запустим наш скрипт:

localhost:~# ./script.sh

Hello from Berlin

Tue Feb 15 15:31:45 UTC 2022

Помните, когда мы говорили про переменные, мы делали так:

export MYVAR=hello

export NAME=Ivan

Но в скрипте мы пишем проще:

CITY=Berlin

Но вызываем мы переменную через значек доллара. Напомню, что переменные пишутся большими буквами, но это джентельменское соглашение - не более. Вы можете назначать их и маленькими буквами, но если хотите, чтобы ваш код был более понятен для остальных - пишите большими.

И не забывацте, где нужно ставить пробелы, а где нет. Например, тут после CITY=Berlin пробел ставить нельзя.

Можно попробовать, кстати.

Но если вам уж очень нужен пробел, тогда его нужно поставить в кавычки:

CITY=" Berlin"

echo Hello from $CITY

date

И все заработает.

Двигаемся дальше.