РАЗРАБОТКА СКРИПТОВ В ОБОЛОЧКЕ BASH

Васин Ю.О.
Научный руководитель: Муравьёв К.А.
МГТУ им. Н.Э.Баумана, кафедра ИУ4, Москва, Россия

DEVELOPMENT OF SCRIPTS IN BASH SHELL

Vasin Y.O.
Supervisor: Murav'ev K.A.
MSTU, Moscow, Russia

Аннотация
В статье описан основные понятия и процесс разработке срипта в оболоке bash. Рассмотрены основные понятия, такие как переменные, циклы, варианты запуска скриптов и работа с очень полезной утилитой IP. Представлены примеры по каждому из пунктов, в которых есть коментарии. Основная проблема решаемая в статье, это вопрос получения начальных знаний, не всех, но основных, для написания скриптов, которые уже могут быть полезны.

Annotation
У статті описаний основні поняття і процес розробки сріпта в оболока bash. Розглянуто основні поняття, такі як змінні, цикли і варіанти запуску скриптів. Представлені приклади по кожному з пунктів, в яких є коментарии. Основна проблема вирішується в статті, це питання отримання початкових знань, не всіх, але основних, для написання скриптів, які вже можуть бути корисні.

Введение

Командная строка часто используется разработчиками и администраторами сетей, но, к сожалению, команды в нее нужно вводить каждый раз, когда это необходимо. На этот случай есть выход виде файла, с уже прописанной последовательностью команд, который можно вызывать в нужный момент. Такой файл называется сценарием командной строки. При это каждая команда может работать для получения самостоятельного результата, либо помогать другим командам, например, в качестве входных данных. Сценарии – это очень хороший инструмент для автоматизации любых последовательностей действий. Существует несколько оболочек, для которых можно писать скрипты – сценарии команд. Например: ksh, zsh, tcsh, bash. В данной статье будет рассмотрено написание скриптов именно для оболочки bash.

1 Как устроены bash-скрипты

Для того, чтобы создать свой первый скрипт, необходимо создать файл командой touch. Например:

touch test

Первая стока файла говорит о том, какую оболочку будем использовать:

#!/bin/bash

Символ решетка используется, как обозначение комментариев, которые оболочка не обрабатывает. Первая строка это исключение из правил, она указывает на то, какую оболочку будем использовать. После этой строки можно записать любую последовательность команд, которая необходима для решения поставленной задачи. Запишем в файл test следующие строки, как на рис.1. и сохраним его.

```
#!/bin/bash

clear       #очистить терминал
pwd         #показать текущую директорию
sleep 5     #ждем 5 сек
clear       #очистить терминал
whoami      #показать текущего пользователя
```

Рис.1. Скриншот содержимого файла tets

Теперь нужно показать системе, что этот файл скрипт –исполняемый файл. Введем команду в терминал:

chmod +x ./test

Теперь попытаемся его выполнить, введя в терминале:

./test

Теперь, в чистом терминале мы увидим текущую директорию на 5 секунд, мотом имя текущего пользователя. Так получился первый скрипт в оболочке bash.

2 Переменные

Существуют разны типы переменных. Например, переменные среды, такие как $HOME, $1, $#, которые содержат текущую директорию или количество принятых параметров. Также в bsch есть переменные, которые можно создать и хранить значения на протяжении действия скрипта. Используя знак доллара к переменным можно обращаться. Если нам нужен знак доллара, например $1 то его необходимо экранировать так: \$1. Так, например. Можно работать с переменными в скриптах, как показано на рис.2, если создать файл test1 и сделать его исполняемым.

```
#!/bin/bash

echo "You enter $# parametrs"
# $#  системная переменная, показывает, сколько параметров передано скрипту

echo "$1 the first parametr"  # $1  перевый переданный параметр
var=27                        # определяем переменную var
echo "$var is old var"        # вывод переменной var
let "var = $var + $1"         # складываем первый параметр и var
echo "$var is new var"        # вывод измененной переменной
echo "Home directory is  $HOME"
# $HOME  системная переменная, содержащая путь к домашнему каталогу
```

Рис.2. Скриншот содержимого файла test1

```
yury@YV ~ $ ./test1 3 28 6776
You enter 3 parametrs
3 the first parametr
27 is old var
30 is new var
Home directory is  /home/yury
yury@YV ~ $
```

Рис.3. Скриншот исполнения скрипта test1 (передаем скрипту 3 параметра)

Из не описанного выше, в данном примере используется утилита let – применяется для проведения над переменными и числами математических операций. Все действия должны быть в ковычках.

3 Управляющие конструкции и циклы

Иногда в программе необходимо сделать ветвление, в зависимости от некоторого условия. Так можно использовать одно из управляющей конструкции: if-then-else. Она позволяет производить некоторые действия, если условие выполнено, и другие действия, если условия не выполнен. Вот как она устроена (возможно использовать без else):

if *команда* ***then*** *команды* ***else*** *команды* ***fi***

Используя оператор *if* можно сравнивать строки и числа, проверять верность логических выражений и т.д.

Также в оболочке есть возможность использовать цикл for, вот его конструкция:

for *var* ***in*** *list* ***do*** *команды* ***done***

Также есть вариант записи этого цикла, похожая на более известный язык C, возможно более логичная запись и привычная многим:

for *((i=1; i <= 10; i++))* ***do*** *echo "number is $i"* ***done***

Так на рис.4 приведен пример работы с циклом и управляющей конструкции. Данный код содержится в скрипте test3.

```
#!/bin/bash

if [ $1 -ge 20 ] #сравниваем первый параметр с 20
   then
      echo "$1 >= 20"
   else
      echo "$1 < 20"
fi

if [ $2 -le 10 ] #сравниваем второй параметр с 10
   then
      echo "$2 <= 10"
   else
      echo "$2 > 10"
fi

for ((a=1; a<10; a++)) #цикл, где выводим поочереди цифры от 1 до 9 и после 9 стрку
do
    echo "$a"

    if [ $a -eq 9 ]
       then
          echo "The end!!!"
    fi

done
```

Рис.4. Пример работы с конструкцией if и for в bash

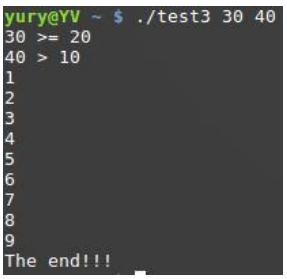


Рис.5. Скриншот с выполнением скрипта test3

На рис.5 пример выполнения скримпта. Первая строка – это вызов скрипта с передачей ему двух параметров 30 и 40. Далее в скрипте идет обработка и сравнение этих параметров при помощи оператора if. После чего цикл for выводит числа от 1 до 9.

4 Выполнение сценариев командной строки в фоновом режиме

Когда скрипт может выполняться большое количество времени, его рекомендуется выполнять в фоновом режиме. Это позволит выполнять другие действия во время выполнения первого скрипта. Это можно сделать просто, стоит всего лишь знак & после вызова скрипта:

./test &

Кроме того имеется возможность отследить выполняемые процесс при помощи команды ps. Если ее запустить до и после запуска скрипта, то можно увидеть, что появиться процесс, новый, который соответствует по номеру тому, что выводит терминам после запуска скрипта в фоновом режиме.

Кроме того, если скрипт выполняется в фоновом режиме, то он продолжает выводить сообщения в терминал.

Так на рис.7 приведен пример кода скрипта, который выполняется в фоновом режиме и в это время в терминале можно выполнять другие команды, например whoami. Также показаны процессы до выполнения скрипта и во время, видно, что добавился один процесс с номером 2898, о присвоении которого сообщила система при запуске скрипта в фоновом режиме.

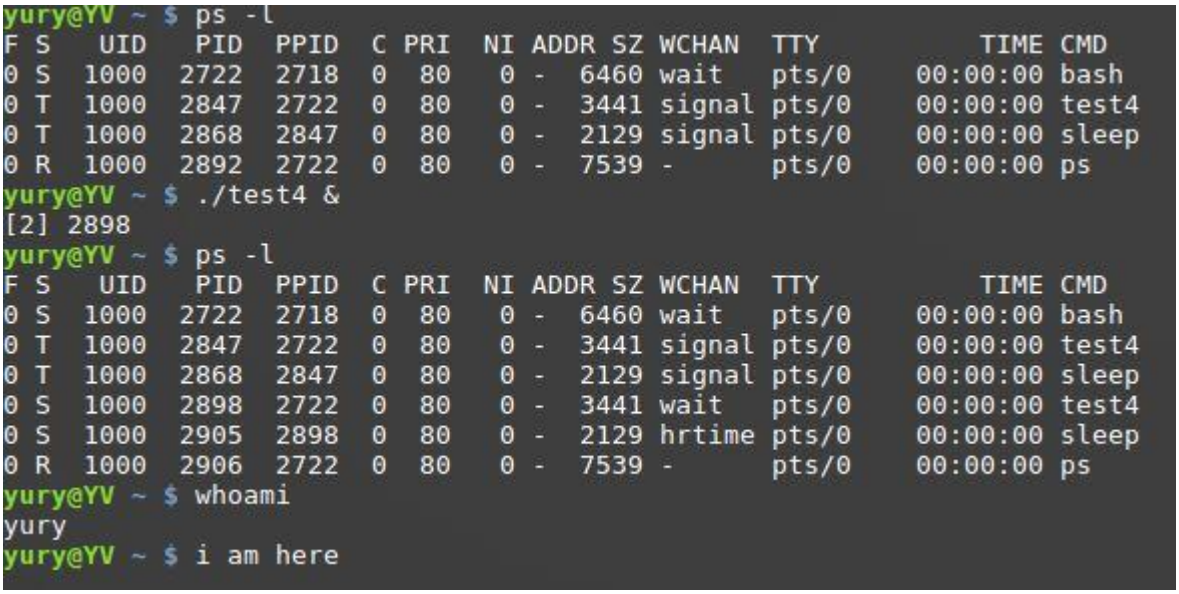
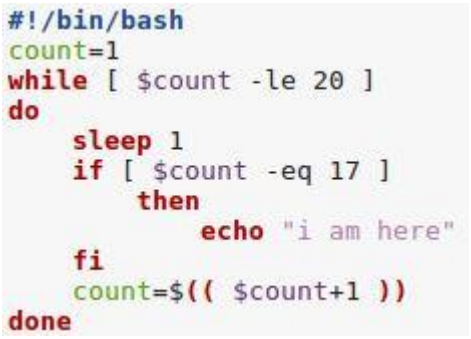


Рис.6. Запуск скрипта в фоновом режиме и проверка ввода другой коменды.



```
#!/bin/bash
count=1
while [ $count -le 20 ]
do
    sleep 1
    if [ $count -eq 17 ]
        then
            echo "i am here"
    fi
    count=$(( $count+1 ))
done
```

Рис.7. Пример кода скрипта test4

На рис.6 приведен пример вызова скрипта для выполнения в фоновом режиме. Скрипт простой: считает от 1 до 20 с интервалом в одну секунду, если счетчик равен 17 то выводит сообщение, которое можно увидеть в терминале даже в фоновом режиме. Хотя пока счетчик считал до 17 можно успеть выполнить команду whoami.

5 Пример работы с файлами, перенаправлением потока и бесконечным циклом

STDOUT — стандартный поток вывода оболочки. Это экран по умолчанию. Почти все команды выводят ответы в STDOUT. Поэтому все ответы появляются в консоли. Эти выводы можно перенаправить в файл, используя символы >>.Большинство bash-команд выводят данные в STDOUT, что приводит к их появлению в консоли. Данные можно перенаправить в файл, присоединяя их к его содержимому, для этого служит команда >>. Если необходимо вывести ответ после команд pwd в файл file то нужно записать в терминале:

pwd >> file

Cat — это часто используемая пользователями Linux утилита. В основном она предоставляет возможности работы с фалами: просмотреть, копировать из одного в другой, склеить несколько файлов, записать в файл, вывести файл и т.д.

В linux есть несколько стандартных потоков: поток ввода, поток вывода. Поток ошибок. Каждый из них можно перенаправить и достаточно гибко работать с ними.

Так, например, STDIN – поток вывода, STDERR- поток ошибок, который по умолчанию совпадает с STDOUT (потоком вывода-терминалом).

Символ < применяют в скриптах для перенаправления стандартного потока ввода, например в файл. Так в примере на рис.8 есть ввода данных из файла. Это все можно употреблять с разными утилитами, например, с cat.

Так, например, на рис.8 приведен текст скрипта, который в бесконечном цикле выводит содержимое двух файлов в терминал. Если есть драйвер для устройств, подключенных по I2C шине. Драйвер, который передает данные на микроконтроллер для управления 8-битным ШИМ-ом для двух вентиляторов, в зависимости от того, какое число записано в файле fan1_input и fan2_input. То есть, по условию задачи числа в файле должны быть числа из диапазона 0-255 включительно. Также необходимо предусмотреть ввод скорости (скважности) ШИМ-а и шага изменения, как параметров скрипта, так и поочередно с клавиатуры.

```
#!/bin/bash
if [ $#  -ne  2 ];  #условие, позволяющее ввод с параметрами и без
    then
        echo "enter speed: "
        read speed  #считать переменную
        echo "enter step: "
        read step   #считать переменную
    else
        let "speed=$1"  #переменная speed равна первому параметру
        let "step=$2"   #переменная step равна второму параметру
fi
for((;;))   #бесконечный цикл
do
    clear;
    echo "speed = " $speed
    echo "$speed" > /home/yury/fan1_input   #перенаправление потока вывода в файл
    echo "$speed" > /home/yury/fan2_input   #перенаправление потока вывода в файл
    let "speed = speed - step"
#-------------------------------------------------------------------
    echo "fan1_input: "
    cat /home/yury/fan1_input   #вывод содержимого файла
#-------------------------------------------------------------------
    echo "fan2_input: "
    cat /home/yury/fan2_input   #вывод содержимого файла
#-------------------------------------------------------------------
    if [[ $speed -le 0 || $speed -ge 255 ]];
#условие позволяющее не выхоить за границы 8-битного ШИМ
        then
            let "step=-1*step"
            if [ $speed -lt 0 ]
                then
                    let "speed= 0"
            fi
            if [ $speed -gt 255 ]
                then
                    let "speed= 255"
            fi
    fi
    sleep 1;
done
```

Рис.8. Пример кода скрипта fan

Первый оператор if позволяет осуществить ввод данных, с клавиатры, если из терминала поступит не 2 параметра. Далее бесконечный цикл:

For((;;)) do *содержимое цикла* ***done***

Каждый раз вначале цикла очищается терминал, что создает ощущение мониторинга содержимого файлов и переменной коросты. Далее поток вывода перенаправляется файлы fan1_input и fan1_input, перезаписывая их. Потом командой:

let "speed = speed –step"

производится декрементация переменной скорости на заданный шаг. Далее выводится сожержимое файлов в терминал. И последнее, происходит проверка, не вышло ли число из диапазона 0-255 и его корекция, в случае выхода. Пауза в 1 секунду и цикл повторяется снова.

Скрипт можно вызывать 2-мя способами, как показано на рис.9 и рис.10.

```
yury@YV ~ $ ./fan 250 70
```

Рис.9. Вызов скрипта fan с параметрами

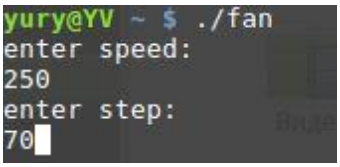


Рис.10. Вызов скрипта fan без параметров, с дальнейшем вводом с клавиатуры.

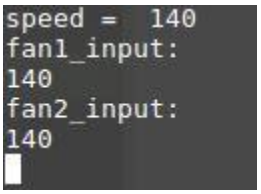


Рис.11. Работа скрипта fan

На рис.11 продемонстрирована работа скрипта fan, который мониторит и перезаписывает данные в файлы, которые далее можно отправлять переферийным устройствам.

6 Утилита IP

Раньше, да и сейчас администраторы сетей часто применяли утилиту ifconfig. Но на данный момент ей на смену пришла более гибкая и удобная утилита IP. Синтаксис будет рассмотрен в этой статье. Эта утилита включает в себя все возможности предшественников. Синтаксис:

ip [опции] объект команда [параметры]

- Опции — самая крупная настройка, которая изменяет работу всей утилиты;
- Объект — то с чем будем работать;
- Команды — какое действие совершить с объектом;
- Параметры — тут происходит передача параметров.

Опции:

- v — только вывод информации об утилите и ее версии;
- s — включает вывод статистической информации;
- o — выводить каждую запись с новой строки.

Часто используемые объекты:

- address — сетевой адрес на устройстве;
- link— физическое сетевое устройство;
- monitor — мониторинг состояния устройств;
- neigh — ARP;
- route — управление маршрутизацией;
- rule — правила маршрутизации;
- tunnel — настройка туннелирования.

Комманды:

- Add;
- Change;
- Del;
- Delete;;
- Flush;
- Get;
- Restore;
- Save;
- Set;
- Update.

Значения команд более ясны, чем все остальное, поэтому не даны пояснения. Если команда не задана, по умолчанию используется show (показать). Также утилита поддержит сокращения, нужно лишь указать несколько первых символов. Но алфавитный порядок соблюдается не всегда. Например, ip a s, означает ip address show, а не ip address set, к сожалению.

Все выше перечислено взаимосвязано. Объект, команда и т.д. Рассмотрим самые основные:

- dev - имя_устройства — сетевое устройство;
- up – включить;
- down – выключить;
- llarrr - MAC — адрес;
- initcwnd - размер окна перегрузки TCP при инициализации;
- window - размер окна TCP;
- cwnd - размер окна перегрузки TCP;
- type – тип;
- via - подключиться к роутеру;
- default - маршрут по умолчанию;
- unreachable - маршрут «недостижимый».

- ip link show — отобразить состояние всех сетевых интерфейсов;
- ip link show eth0 — отобразить состояние eth0;
- ip link set eth1 up — включить eth1;
- ip link set eth1 down выключить eth1.

- ip neigh show — показать все записи ARP;
- ip neigh show dev eth0 — посмотреть все ARP записи для eth0;
- ip neigh flush — удалить все ARP записи;
- ip ne fl dev eth0 — удалить все ARP записи для eth0.

- ip address show — показать все ip адреса и их интерфейсы;
- ip a l permanent — отобразить только статические ip адреса;
- ip a l dynamic — отобразить только динамические ip адреса;
- ip addr add 1.1.1.13/24 dev eth0 — установить ip адрес для интерфейса eth0;
- ip add flush dev eth0 — удалить все ip адреса интерфейса eth0.

- ip r sh показать все маршруты в таблице маршрутизации;
- ip route get 10.10.20.0/24 — отобразить маршрут к этой сети;
- ip route add 10.10.20.0/24 via 192.168.50.100 — создать маршрут.

Заключение

На самом деле, тема bash-программирования огромна. Содержимое этой работы, лишь капля в море информации, которую необходимо изучить и знать для разработки скриптов на bash. Но здесь собрана основная, фундаментальная информация, с которой можно начинать изучение bash. Также приведены реальные, рабочие примеры, даны комментарии в коде и скриншоты работы скриптов, упомянуты некоторые важные и интересные моменты работы со скриптами, такие как работа скрипта в фоновом режиме, работа с системными переменными и т.д. В данной работе были рассмотрены темы переменных, циклов конечных и бесконечных, потоков ввода и вывода, работы с

параметрами и некоторые вопросы утилит cat и let, а также очень полезной утилитой IP, которая часто используется для настройки сетевых интерфейсов.

Литература

1.Bash script step by step tutorial [Электронный ресурс]. URL: https://likegeeks.com/bash-script-easy-guide/ (дата обращения 15.04.2018).

2. Shell Scripting Part4 – Input, Output, and Redirection [Электронный ресурс]. URL: https://likegeeks.com/shell-scripting-awesome-guide-part4/ (дата обращения 17.04.2018).

3. Linux для чайников / Лебланк Ди-Анн, Хоуг Мелани; Москва: Диалектика, 2003.– 216 с.

4. Bash scripting Tutorial [Электронный ресурс]. URL:https://linuxconfig.org/bash-scripting-tutorial (дата обращения 17.04.2018).