

Программирование на Bash: Базовое программирование на Bash. Коды возврата, функции.



Шило
Петр



Шило Петр

DevOps-инженер

Arifonica



[Шило Петр](#)

Модуль «Программирование на Bash»

Цель модуля:

- писать и читать простейшие bash-скрипты;
- правильно использовать вывод различных утилит;
- знает регулярные выражения в объеме, необходимом для разбора логов приложения и агрегации записей в них;
- производить дебаг bash-скриптов и искать в них проблемы.



Структура модуля

1. Базовое программирование на Bash. Коды возврата, функции.
2. Regexp и их использование для синтаксического анализа. Полезные утилиты.
3. Полезные утилиты.
4. Разбор скриптов и написание своих скриптов. Linter. Shell check.

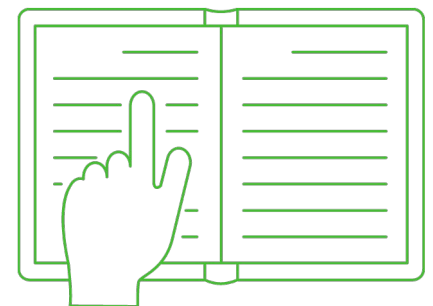


Предисловие

На этом занятии мы поговорим о:

- основах программирования на Bash;
- кодах возврата;
- функциях, условиях и циклах в языке Bash.

По итогу занятия вы получите представление о работе Bash и научитесь программировать на Bash.





План занятия

1. [Предисловие](#)
2. [Основы Bash-скриптинга](#)
3. [Переменные и коды ответов](#)
4. [Условные операторы](#)
5. [Циклы](#)
6. [Функции](#)
7. [Итоги](#)
8. [Домашнее задание](#)



Основы Bash-скриптинга

Обзор

Bash (Bourne-again shell) – самая популярная оболочка командной строки для Linux.

Помимо внешних утилит имеет широкий набор встроенных функций, таких как циклы, условия и прочее.

Простейший скрипт - последовательный набор команд для выполнения.

Пример простейшего скрипта:

```
#!/bin/bash  
echo "Creating file"  
touch file  
echo "Done"
```


Запуск скриптов

Файлы скриптов обычно имеют расширение `.sh`

```
touch filename.sh
```

Для запуска скрипта пользователь должен иметь права на его чтение и выполнение.

```
chmod +rx filename.sh
```

В начале скрипта необходимо указать интерпретатор

```
#!/bin/bash или #!/bin/sh
```

Эти правила можно обойти:

```
touch scriptname
```

Если прав на выполнение нет то можно просто запустить:

```
bash scriptname
```

Если не указать интерпретатор, то Bash будет использоваться по умолчанию.



Переменные и коды ответов

Переменные

Присваивание переменных доступно несколькими способами.

Простое:

```
a=2
```

С помощью let:

```
let a=2
```

Чтение ввода пользователя:

```
read a
```

Как результат выполнения команды:

```
a=`cat /etc/hostname`
```

Использование переменных в скриптах:

```
#!/bin/bash
```

```
botname=Linux
```

```
echo "What is your name"
```

```
read name
```

```
server=`cat /etc/hostname`
```

```
echo "Hello $name, my name is $botname"
```

```
echo "Welcome to the $server"
```

Перенаправление потоков

Стандартные потоки ввода/вывода:

- **0** (STDIN) ввод пользователя;
- **1** (STDOUT) стандартный поток вывода;
- **2** (STDERR) стандартный поток ошибок.

Для перенаправления результата выполнения команды в другую команду используется **|** (pipe).

Пример:

```
cat /etc/passwd | wc -l
```

wc -l – используется для подсчета количества строк.

Еще пример:

```
ps aux | grep bash | wc -l
```

Коды ответов

Код ответа (Return code, код возврата) – код который возвращает команда или функция после выполнения.

Код возврата предыдущей команды хранится в переменной **\$?**

Пример:

```
cat /etc/not_existing_file  
echo $?
```

Код возврата – число от 0 до 255

Традиционно:

- 0 – успех;
- Любое другое число – как правило, код ошибки (свой у каждого приложения).



Условные операторы

Проверка условий

2 варианта написания:

`[]` или `[[]]`

Примеры:

`[-d /tmp]`

`[[$a -eq 8]]`

Инверсия результата достигается с помощью «!».

Например:

`[[! -d /tmp]]` будет успешна если директория не существует

Таблица проверочных выражений

| Работа с файлами | |
|--|--|
| <ul style="list-style-type: none">-e – Проверить существует ли файл или директория-f – Файл существует (!-f - не существует)-d – Каталог существует (!-f - не существует)-s – Файл существует и он не пустой-r – Файл существует и доступен для чтения-w – ... для записи-x – ... для выполнения-h – символическая ссылка | |
| Работа со строками | Операции с числами |
| <ul style="list-style-type: none">-z – Пустая строка-n – Не Пустая строка== – Равно!= – Не равно | <ul style="list-style-type: none">-eq – Равно-ne – Не равно-lt – Меньше-le – Меньше или равно-gt – Больше-ge – Больше или равно |

Условные операторы

В Bash реализованы 2
условных оператора: **if** и
case.

- **if** используется для бинарной проверки (да / нет);
- **case** для выбора из множества вариантов (однако возможны применения различных вариантов).

Синтаксис if:

```
if [ ... ]; then echo "true"; else echo  
"false"; fi;  
if [ ... ] && [ ... ]; then  
    ...  
elif [[ ... && ... ]]; then  
    ...  
else  
    ...  
fi;
```

Условные операторы

В Bash реализованы 2
условных оператора: **if** и
case.

- **if** используется для бинарной проверки (да / нет);
- **case** для выбора из множества вариантов (однако возможны применения различных вариантов).

Синтаксис case:

```
case "$variable" in
    "condition1" )
        command...
    ;;
    "condition2" )
        command...
    ;;
esac
```

Пример if и case

```
port=80
if [[ $port -eq 80 || $port -eq 8080 ]]
then
    echo "This is HTTP"
elif [[ $port -eq 22 ]]; then
    echo "This is SSH"
else
    if [[ $port -eq 53 ]]; then
        echo "And this is DNS"
    else
        echo "I dont know what is that"
    fi
fi
```

```
port=80
case "$port" in
    ('80'|'8080')
        echo "This is HTTP"
    ;;
    '22')
        echo "This is SSH"
    ;;
    '53')
        echo "And this is DNS"
    ;;
    *)
        echo "I dont know what is
that"
    ;;
esac
```

Логические операторы

Для упрощения записи команд есть 3 оператора, позволяющие избежать ненужного усложнения с помощью `if` или `case`:

- `команда1 ; команда2` – команда2 выполняется после команды1 независимо от результата ее работы ;
- `команда1 && команда2` – команда2 выполняется только после успешного выполнения команды1 (т.е. с кодом завершения 0), это аналог операции **AND** (Логическое И);
- `команда1 || команда2` – команда2 выполняется только после неудачного выполнения команды1 (т.е. код завершения команды1 будет отличным от 0), это аналог операции **OR** (Логическое ИЛИ).



Циклы

Циклы: while

while – самый простой цикл

Синтаксис:

```
while [[ условие ]] do
    echo "Ok"
done
```

➡ Бесконечное выполнение команд пока условие выполняется.

Пример:

```
#!/bin/bash

i=0

while [[ $i -le 4 ]]; do
    echo $i
    i=$(( $i + 1 ))
done
```

Бесконечный цикл:

```
while true; do
    echo 'I can do that all day'
done
```

Циклы: for

Если нет условия, а только список элементов – используем **for**

```
for имя in элемент1 элемент2 do
    ...
done
```

➡ Выполняет тело цикла для каждого из **элементов**, загружая их поочередно в переменную **ИМЯ**

Пример:

```
#!/bin/bash
for file in /tmp/*; do
    echo "$file"
done
```

Еще пример:

```
#!/bin/bash
for item in coffee tea; do
    echo "We have a $item"
done
```

Циклы: select

Не совсем цикл, но стоит упомянуть.

Синтаксис похож на **for**:

```
select имя in элемент1 элемент2; do
    ...
done
```

➡ Дает пользователю выбрать вариант из предложенных.

Пример:

```
#!/bin/bash

select pill in red blue; do
    echo "You chose $pill"
done
```




Функции

Функции

Функция – часть кода, вынесенная отдельно, для многократного использования.

Объявление функции в Bash:

```
имя_функции () {  
    команды  
}
```

Вызов функции:

```
имя_функции
```

Пример:

```
#!/bin/bash  
currentdate () {  
    date  
}  
date  
date
```

Функции: параметры

Для многократного использования функцию обычно требуется запускать с разными параметрами.

Параметры – доступные только внутри тела функции переменные.

Например, вызов функции:

```
function 5 2
```

Имеет 2 параметра, к которым можно обратиться внутри самой функции с помощью переменных **\$1** и **\$2**

Пример функции с параметром:

```
#!/bin/bash
division () {
    if [[ $2 -ne 0 ]]; then
        echo "$1/$2 = $((($1/$2))"
    else
        echo "division by zero"
    fi
}
division 15 5
division 4 2
division 3 0
```

Функции: код возврата

Указать код возврата для скрипта можно с помощью команды `exit`.

По умолчанию возвращает 0.

Но можно задать любое другое значение с помощью `exit X`.

Выход из функции осуществляется с помощью `return X`.

➡ Это позволяет передавать ошибку и обрабатывать, если уже известно, что что-то пошло не так.

Пример функции с кодом возврата:

```
#!/bin/bash
division () {
    if [[ $2 -ne 0 ]]; then
        echo "$1/$2 = $((($1/$2))"
        return 0
    else
        echo "division by zero"
        return 1
    fi
}
division 15 5
division 4 2
division 3 0
```

Отладка

Интерпретатору `#!/bin/bash` можно передавать параметры работы такие как:

- `-n` – читать все команды, но не выполнять их;
- `-v` – выводить все строки по мере их обработки интерпретатором;
- `*-x` – выводить все команды и их аргументы по мере их выполнения.



Итоги

Итоги

Сегодня мы рассмотрели базовые навыки программирования Bash и теперь:

- умеем составлять скрипты из команд;
- можем использовать функции, циклы и условия.





Домашнее задание

Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и
пишите отзыв о лекции!**

Петр Шило