нетология

Командная оболочка Bash: практические навыки

Алексей
Метляков

Алексей Метляков

DevOps Engineer, OpenWay

Алексей Метляков

План занятия

1. Bash - что это такое?
2. Для чего может быть использован bash в DevOps?
3. Основы синтаксиса
4. Как написать первый скрипт?
5. Итоги
6. Домашнее задание

Bash - что это такое?

Что такое Bash?

Bash - Bourne Again Shell, очередная командная оболочка Борна, представляет собой расширенную версию Bourne Shell (sh).

- **Является** основной командной оболочкой для систем Linux, работающий как в интерактивном, так и в скриптовом режиме.
- **Умеет работать** с автодополнением команд, имён файлов и каталогов (клавиша Tab), создавать переменные, работать с циклами, ветвлениями, подстановку вывода результата команд

Для чего нужен bash в DevOps?

В основном, bash используется для деплоя:

- **Автоматизация деплоя контейнеров** позволяет не просто скачать контейнер с приложением, но и подготовить окружение для деплоя
- **Динамическое изменение конфигурации приложений.** В случае, если ваше приложение проходит через несколько этапов тестирования и имеет огромное количество интеграций с другими сервисами, гораздо удобнее подставлять конфигурационные файлы автоматически
- **Остановка, перезапуск сервисов, минимальное тестирование etc,** в общем случае, любые действия, которые необходимо выполнять на системе перед первым запуском приложения, проще автоматизировать один раз, чем выполнять руками

Для чего нужен bash в DevOps?

Bash можно использовать и на этапе разработки:

- **Автоматизация скачивания репозиториев с зависимостями.** Подойдёт в случае, если вы не используете системы сборки приложений и у вас есть проекты/библиотеки/репозитории, которые вы переиспользуете в своём проекте
- **Сборка по коммиту** позволит автоматизировать часть непрерывной интеграции, отлавливая события коммита в репозиторий с последующим запуском сборки

Основы синтаксиса

Переменные, массивы

В **bash** все переменные - целые числа или строки, в зависимости от того, как они определены, их можно задавать в явном и неявном виде:

```
a=33 #Неявное определение целочисленного
b=SA34 #Неявное определение строки
```

Переменные, массивы

Такая неопределённость может порождать разные вариации при работе с переменными:

```
a=33 #Строка с целым числом
let “a+=1” && echo $a #Получаем 34
b=SA34 #Строка с буквами и целым числом
let “b+=1” $$ echo $b #Получаем 1, т.к. переменная не была целочисленной
b=SA34 #Возвращаем значение со строкой
a=${b/SA/33} #Присваиваем переменной a значение b, заменив SA на 33
echo $a # Получим число 3334
let “a+=1” && echo $a #Получаем 3335, то есть значение целочисленное
```

Переменные, массивы

Для большего контроля над переменными при их объявлении можно использовать команду **declare**:

```
declare -i a #Объявление целочисленной переменной
declare -i b;declare -i c
a=32;b=2;c=$a+$b
echo $c #Выведет 34
c=”Hello” #Попытка присваивания строкового значения
echo $c #Выведет 0
d=32;e=2;f=$d+$e
echo $f #Выведет 32+2
```

Переменные, массивы

Массивы представляют из себя одномерный набор элементов, который может включать в себя одновременно и строковые и целочисленные значения:

```
array_int=(0 1 2 3 4 5) #Массив из целых чисел
array_str=(“one” “two” “three” “four” “five”) #Массив строк
array_mix=(1 “two” 3 “four” 5) #Массив с обоими типами данных
echo $array_int #Выводит первый элемент массива
echo ${array_int[1]} #Выводит второй элемент массива
echo ${array_int[@]} #Выводит все элементы массива
i=3
echo ${array_int[$i]} #Выводит четвёртый элемент массива
```

Переменные, массивы

Синтаксис для работы с массивами:

```
echo ${!array_int[@]} #Получить индексы массива
echo ${#array_int[@]} #Получить размер массива
array_int[0]=0 #Перезаписать значение первого элемента
array_int+=(6) #Добавить в конец массива элемент со значением
array_out=() #Создать пустой массив
array_out=$(ls) #Записать вывод ls как строку
array_out=($(ls)) #Записать вывод ls как набор строковых
элементов
```

Арифметические операции

Bash поддерживает все стандартные операции с целыми числами:

```
a=2;b=4;c=0
c=$(($a+$b))
c=$(($b-$a))
c=$(($b*$a))
c=$(($b/$a))
```

Логические операции

Bash позволяет проводить сравнение как чисел, так и строк. Для чисел доступны следующие операции:

```
a=2;b=4
[“$a” -eq “$b”] #числа равны
[“$a” -ne “$b”] #числа не равны
[“$a” -gt “$b”] #число a больше b
[“$a” -ge “$b”] #число a больше или равно b
[“$a” -lt “$b”] #число a меньше b
[“$a” -le “$b”] #число a меньше или равно b
```

Логические операции

Для строк набор операций чуть меньше:

```
a=Hello;b=hello
[“$a” = “$b”] #строки равны
[“$a” == “$b”] #строки равны
[“$a” != “$b”] #строкине равны
[“$a” \> “$b”] #строка a больше b
[“$a” \< “$b”] #строка a меньше b
[ -n “$a”] #строка a не пустая
[ -z “$a”] #строка a пустая
```

Оператор условия

Конструкция if-then-elif-else в bash имеет следующий синтаксис:

```
a=Hello;b=hello
if [“$a” = “$b”]
then
  echo “строка $a равна строке $b”
elif [“$a” \> “$b”]
then
  echo “строка $a больше строки $b”
else
  echo “строка $a меньше строки $b”
fi
```

Циклы

Существует два вида конструкций циклов:

```
array_int=(0 1 2 3 4)
for i in ${array_int[@]}
do
    echo $i
done
```

Выводим построчно значения элементов массива

```
a=5
while (($a > 0))
do
    echo $a
    let “a -= 1”
done
```

Выводим значение переменной

Циклы

Цикл **for** удобен для обработки:

- **Значений массивов**, как указано на предыдущем слайде;
- **Вывода команд,** набором данных может выступать вывод команды **ls**, где через пробел указаны значения, которые будут интерпретироваться как элементы массива;
- **Значений файлов**, в этом случае набором данных может выступать вывод команды **cat**. По умолчанию, разделителями считаются: пробел, знак табуляции, знак перевода строки. Разделитель можно переопределить через переменную **IFS**.

Циклы

Цикл **while** удобен для использования в тех случаях, когда условие выхода из него сложно ограничить конечным количеством итераций. Например, мы должны прекратить отслеживание содержимого файла только в том случае, если файл перестал быть доступен нам для чтения.

Расширение скобок

Эта возможность позволяет формировать строки из наборов символов:

```
echo s{t,tr}ing #Выведет sting string
echo {a..g} #Выведет a b c d e f g
echo {1..9} #Выведет 1 2 3 4 5 6 7 8 9
echo {7..A} #Выведет 7 8 9 : ; < = > ? @ A
```

Данную особенность удобно использовать совместно с командами:

```
ls *.{png,jpg} #Выведет все png и jpg файлы из текущего каталога
```

Как написать первый скрипт

Существует ряд основных правил для формирования правильного bash-скрипта:

- Скрипт должен начинаться с #!/usr/bin/env bash
- Необходимо следить за скобками и правильно ими экранировать операции с переменными

Итоги

Сегодня мы успели узнать:

- Для чего нужен bash в DevOps
- Познакомились с основами его синтаксиса
- Как написать первый скрипт
- Запомнили, что необходимо следить за скобками и правильно ими экранировать операции с переменными

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**.

нетология

Задавайте вопросы и пишите отзыв о лекции!

Алексей Метляков

Алексей Метляков