Bash

Bash для всех

Автор: Савенко Вадим Анатолиевич @2024

Содержание:

1. Коротко про Bash

Bash был разработан как улучшенная версия оригинального Bourne Shell (sh), включающая в себя новые возможности и улучшения. Он совместим с большинством команд и скриптов, написанных для sh, но также предоставляет множество дополнительных функций, таких как улучшенные возможности обработки строк, работы с массивами и расширенные механизмы управления потоками ввода-вывода. В основном он используется как список команд для Linux дистрибутивах. Обычно файл для Bash проектов имеет формат *.sh*, но однако файлы могут и не иметь такового. Достаточно ввести в терминале команду: “sh /путь/к/файлу”. Но если не хочется замолачиваться писать такую команду вечно, то можно написать команду: “chmod +x /путь/к/файлу”. Теперь, при вводе в консоли “./тот_самый_файл” или “/путь/к/файлу”, то система поймёт, что это Bash проект и надо его запускать через команду: “sh /путь/к/файлу”.

2. Команды

Для написание кода Bash, можно использовать любой редактор кода. Код, который отображает текст в консоли:

```
echo “Hello World”
```

В итоге в консоли будет:

```
Hello World
```

Где было написано “Hello World”, там можно указывать данные разных значений. Есть также команда, которая принимает данные у пользователя. Выглядит таким образом:

```
read var
```

Где было написано *var*, там указывалось, что создаётся переменная *var.* Разговор про переменные будет позже.

Давайте создадим код, который будет нам показывать наше введённой имя и фамилию, с помощью тех команд, которые мы узнали:

```
echo “Your first name:”
read firstName
echo “Your last name:”
read lastName
echo “Your are: $firstName $lastName”
```

Где было написано переменные через знак доллара ($), он обязан быть, если мы используем (Если создаём переменные, то знак доллара ставить не надо).

Теперь при запуске данного кода, нам предлагается ввести наше имя и фамилию, и после ввода наших данных, мы получаем результат в терминале, кто есть мы.

Также можно создавать комментарии в коде. Это предназначено, чтобы пропускались действия команды. Комментарии создаться через решётку (#) в начале строки команды. Вот пример:

```
# echo Коментарий
```

Есть ещё другие команды, но в этой книге будут показаны основные кода для Linux. Словарь с базовыми командами есть на стр. 15.

3. Переменные

На стр. 4 было упоминание про переменные. Бывают случали, когда переменные должны иметь свои данные, а не те, которые ввёл пользователь. Вот пример создание переменных:

```
var1="Hi" # Тип данных: текстовое
var2=50 # Тип данных: числовое
var3=(50 "Hi" 30) # Тип данных: массив
echo "Vars: 1st=$var1; 2nd=$var2; 3rd=${var3[2]}"
```

Как было замечено, все переменные имеют значение, которое имеет свой тип данных. В Bash есть 3 типа данных:

- **Текстовые**: там где храниться любые набор символов, которые хранятся в двойных скобках (“ ”).
- **Числовые**: там где храниться только цифры.
- **Массивы**: там где храниться набор различных значений, с разными типами данных.

Если мы знаем, как отобразить переменные числовых и текстовых типов данных, то к массивам ещё есть индекс, который является целым числом и храниться в квадратных скобках ([]). Индекс — номер последовательности элемента, первый элемент который начинается не с 1 (Так как это уже 2ой элемент), а с 0.

Если индекс меньше 0 (Например: -1; -2; -3; и так далее), то это означает, что берётся элементы обратном порядке. Например: [-1] - последний элемент; [-2] - предпоследний элемент; и так далее.

Также можно добавлять числовые и текстовые переменные в массив. Вот таким образом:

```
var=50
array=($var "Hi" 30)
```

Если в Python после каждого элемента (Кроме последнего) надо ставить запятую и пробел, тем самым подтверждая о создание элемента в списке (Для Bash — это массив), то в Bash достаточно поставить только пробел.

4. Математика

Для выполнения математических операций в Bash используется команда *let*, арифметическое расширение *$(( ))*, или команда *expr*. Тип данных переменных должен быть числовым. Вот код по работе с математическими операций:

```
let "a = 5 + 3"
b=$((5 * 3))
c=$(expr 10 - 3)
```

Если мы текстовые данные умножим на числовое, то будет повторение набор символов текстовых данных. Если к текстовые добавить ещё текстовые данные, то это объединение текстовых данных.

Также кроме умножение, суммы и вычитании, есть ещё другие операции. Вот пример кода с другими арифметическими действиями:

```
quotient=$((first_number / second_number)) # Деление
power=$(echo "$first_number^$second_number" | bc) # Степень
sqrt_first=$(echo "scale=2; sqrt($first_number)" | bc) # Корень
```

КСТАТИ!!! Было забыто рассказать важную вещь. Имя переменной должны быть:

1) Написано английским алфавитом;
2) Не совпадасться с ключевыми именами в Bash (Такие как: *as*, *from*, *with*, *if* и другие. Мы будем изучать все ключевые слова в Bash);
3) Начинаться с буквы;
4) Цифры не могут быть первыми символами название переменной;
5) Не иметь другие символы, кроме: буквы английского алфавита; цифры; тире; (_) символа;

5. Конвертация

Когда мы хотим иметь такие-же данные переменной, но в другом формате, надо сделать конвертацию. Пример конвертации есть на стр. 8

Пример конвертации в коде:

```
# Число в строку
number=789
text="$number" # Хранит number в текстовом формате

# Строка в число
text="1011"
number=$((text))
number=$((text)) # Хранит text в числовом формате
```

6. Условия

Если сегодня понедельник, вторник, среда, четверг или пятница — мы работаем. Если суббота — работаем меньше. Иначе — мы отдыхаем. Данный абзац является — примером условий.

В Bash можна задавать условия. Вот пример кода по условиям:

```
if [ $VAR -gt 10 ]; then # Если это истина
    echo "VAR больше 10"
elif [ $VAR -eq 10 ]; then # Иначе, если это истина
    echo "VAR равно 10"
else # Иначе (Если всё было - ложь)
    echo "VAR меньше 10"
fi
```

Здесь *-gt* означает "больше чем", *-eq* означает "равно", а *-lt* означает "меньше чем". Одним словом, это всё — логические операторы. В Bash есть ещё больше логических операторы. В основном они делятся для строковых и для строковых.

Вот все логические операторы:

Для числовых данных:

- *-gt*: больше чем (greater than)
- *-lt*: меньше чем (less than)
- *-ge*: больше или равно (greater than or equal to)
- *-le*: меньше или равно (less than or equal to)
- *-eq*: равно (equal to)
- *-ne*: не равно (not equal to)

Для строковых данных:

- >: больше чем (greater than)
- <: меньше чем (less than)
- =: равно (equal to)
- !=: не равно (not equal to)

7. Циклы

Мы приводили пример на стр. 8, а теперь представим, что это цикл, который повторяется постоянно до тех пор, пока не уйдём на пенсию.

В Bash есть цикли, которые повторяют действия до тех пор, пока его значение не будет ложью, и циклы, которые выполняют набор команд несколько раз. Вот пример цикла для выполнение несколько раз набор кодов:

```
for i in {1..5}; do
    echo "Число: $i"
done
# Показует результат от 1 до 5.
# Тоисть выполняет команду 5 раз.
# {1..5} - там задаём выполнение количество раз.
```

Вот пример постоянного цикла:

```
COUNTER=0

while [ $COUNTER -lt 5 ]; do
    echo "COUNTER: $COUNTER"
    COUNTER=$((COUNTER + 1))
done
```

Где квадратные скобки (*[ ]*), там мы указываем условия, для понимание истины и лжи.

8. Функции

Функции в программировании, включая скрипты Bash, используются по нескольким важным причинам. Давайте рассмотрим пример по созданию кода с функцией:

```
my_function() {
    local a=$1
    local b=$2
    echo "Это функция"
    echo "Аргумент a: $a"
    echo "Аргумент b: $b"
}

# Вызов функции с аргументами
my_function 3 5
```

Теперь разберём код. Разборка кода есть на стр. 11.

1. Шебанг (Shebang):

Это первая строка скрипта, называемая "шебанг". Она указывает системе, что этот скрипт должен выполняться с помощью интерпретатора Bash.

2. Определение функции:

Здесь определяется функция с именем *my_function*. В Bash функции объявляются с помощью имени функции, за которым следует пара круглых скобок и фигурные скобки {} для определения тела функции.

Внутри функции:

- *local a=$1*: Создается локальная переменная a и присваивается значение первого аргумента функции (*$1*).
- *local b=$2*: Создается локальная переменная b и присваивается значение второго аргумента функции (*$2*).
- *echo "Это функция"*: Выводится строка "*Это функция*".
- *echo "Аргумент a: $a"*: Выводится значение переменной *a*.
- *echo "Аргумент b: $b"*: Выводится значение переменной *b*.

3. Вызов функции с аргументами:

Здесь вызывается функция *my_function* с двумя аргументами: 3 и 5. Эти аргументы передаются функции, где они становятся доступными как *$1* и *$2*.

Подробное описание работы:

1. **Когда вызывается функция** *my_function* 3 5:

- 3 становится первым аргументом и присваивается переменной a внутри функции.
- 5 становится вторым аргументом и присваивается переменной b внутри функции.
  - **Внутри функции**:
- *local a=$1*: Переменной a присваивается значение 3.
- *local b=$2*: Переменной b присваивается значение 5.
- *echo "Это функция"*: Выводится строка "*Это функция*".
- *echo "Аргумент a: $3"*: Выводится строка "*Аргумент a: 3*".
- *echo "Аргумент b: $5"*: Выводится строка "*Аргумент b: 5*".

Полный процесс выполнения:

1. Скрипт начинается с строки #!/bin/bash, которая указывает системе использовать интерпретатор Bash.
2. Определяется функция *my_function*.
3. Функция *my_function* вызывается с аргументами 3 и 5.
4. Внутри функции *my_function*:

- Аргументы 3 и 5 присваиваются локальным переменным a и b.
- Функция выводит строки с текстом и значениями аргументов.

9. Открытие файлов

Известно, что можна хранить данные с помощью переменной, но можно хранить, изменять, добавлять и читать данные из файла. Вот пример кода Bash для чтение, добавление и изменение файлов:

```
# Пример чтения из файла
while IFS= read -r line; do
    echo "$line" # $line - переменная, хранащяя данные из файла
done < "/путь/к/файлу"

# Пример добавления данных в файл
echo "Новая строка" >> "имяФайла"

# Пример записи в файл (перезапись)
echo "Перезаписанная строка" > "имяФайла"
```

10. Словарь базовых переменных Linux

Назначение переменной	Имя переменной	Описание переменной
Путь к исполняемым файлам	$PATH	Список директорий, в которых Bash ищет исполняемые файлы.
Домашняя директория пользователя	$HOME	Путь к домашней директории текущего пользователя.
Имя текущего пользователя	$USER	Имя текущего пользователя.
Команда для оболочки по умолчанию	$SHELL	Путь к исполняемому файлу оболочки по умолчанию.
Директория временных файлов	$TMPDIR (или $TMP)	Путь к директории, используемой для временных файлов.
Версия операционной системы	$OSTYPE	Тип операционной системы.
Стандартный текстовый редактор	$EDITOR	Имя стандартного текстового редактора.
Персональный идентификатор пользователя	$UID	Числовой идентификатор текущего пользователя.
Персональный идентификатор группы	$GID	Числовой идентификатор группы текущего пользователя.
Стандартное количество строк на экране	$LINES	Количество строк в терминале.
Стандартное количество столбцов на экране	$COLUMNS	Количество столбцов в терминале.

11. Словарь базовых команд Linux

Команда	Предназначение
`sudo su`	Перейти в режим суперпользователя
`echo` *текст*	Показует *текст* в терминале
`nano` *путь/к/файлу/файл* Пакет: *nano*	Перейти в редактор текста для редактирование *путь/к/файлу/файл.*
`ip addr` Пакет: *ip*	Узнать IP-адрес хоста (ПК)
`cd` *путь/к/директории*	Перейти в директорию по адресу: *путь/к/директории*
`ls`	Посмотреть, что находиться в директории
`wget` *https://ссылка.на.файл* Пакет: *wget*	Скачать файл с интернета
`rm` *путь/к/файлу/файл*	Удаление файла
`rm –r` *путь/к/директории*	Удаление директории
`cp` *путь/к/файлу/файл путь/к/вставлению*	Копирование файла
`cp –r` *путь/к/директории путь/к/вставлению*	Копирование директории
`mkdir` *имяПапки*	Создание папки “*имяПапки*”
`kill` *процесс*	Останавливает процесс

Команда	Предназначение
chmod +x *путь/к/файлу/файл*	Даёт системе понимать, что это не текстовый файл, а код для выполнение команд Bash
git clone *https://путь.к.репозиторию* Пакет: *git*	Скачивает все файлы с репозитория (Например: с GitHub)
neofetch	Информация о системе
sudo ssh –X *пользовательТогоПК@IPкомпьютера* Пакет: *openssh*	Подключение удалённого доступа и работа через консоль
sudo ssh –X *юзерТогоПК@IPкомпьютера программа* Пакет: *openssh*	Подключение удалённого доступа и работа “*программа*” (Это может быть графическая или консольная)
mv *файл1 файл2 … /перемещение/в/директорию*	Перемещение файл
mv –r */путь/директории /перемещение/в/другую/директорию*	Переместить директорию
scp */путь/к/локальному/файлу username@remote_host:/путь/на/удаленной/машине* Пакет: *openssh*	Копирование файлов на другой ПК

Команда	Предназначение
`gzip` *`файл.txt`* Пакет: *gzip*	Сжатие архива *файл.txt.tar.gz*
`gzip -l` *`архив`* Пакет: *gzip*	Информация об архиве
`bzip2 -d` *`архив`* Пакет: *bzip2*	Распаковка архива

Bash для всех

Автор: Савенко Вадим Анатолиевич @2024