
Разработка скриптов на BASH

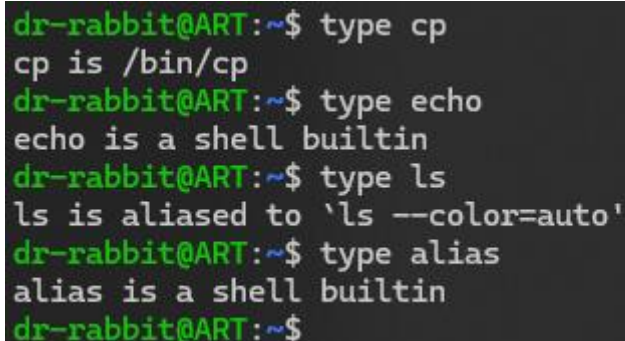
Цель работы: получить практические навыки программирования на bash, освоить практические приемы работы с командной оболочкой bash.

Необходимо:

- ОС Linux на реальной или виртуальной машине

Краткие теоретические сведения:

Традиционным средством автоматизации для операционных систем является использование скриптовых языков. Скрипт представляет из себя текстовый файл с кодом программы, который подается на исполнение в бинарный файл командного интерпретатора. Т.е. не происходит компиляция кода в бинарный исполняемый файл. Часто командные интерпретаторы выступают в роли основного shell – программной среды, предоставляющей интерфейс управления ОС для пользователя. Для UNIX-подобных систем одним из популярных (но не единственных) командных интерпретаторов является bash - Bourne-Again SHell («перерожденный шел» старый UNIX шелл - sh). В Linux системах он располагается по пути /bin/bash и является шеллом по умолчанию для пользователей. Bash позволяет запускать команды (внешние бинарные файлы) и использовать встроенные в сам bash команды.



```
dr-rabbit@ART:~$ type cp
cp is /bin/cp
dr-rabbit@ART:~$ type echo
echo is a shell builtin
dr-rabbit@ART:~$ type ls
ls is aliased to 'ls --color=auto'
dr-rabbit@ART:~$ type alias
alias is a shell builtin
dr-rabbit@ART:~$
```

На рисунке выше с помощью команды type определяется, чем на самом деле являются вызываемые команды. Так cp – это внешняя программа, echo – встроенная команда bash, а ls – это алиас на команду ls –color=auto, созданный с помощью команды alias 😊

Опишем основные концепции языка bash.

Во-первых, все переменный одного типа – строка. То, что строка содержит число интерпретатор определяет по контексту программы.

Переменные объявляются так:

VARIABLE_NAME="value" или так VARIABLE2_NAME=5

Обращение к переменным делается так:

\$VARIABLE_NAME или \${VARIABLE_NAME}.

Второй способ используется, когда необходимо «отделить» переменную от окружающих символов, или задействовать «хитрые» возможности bash по работе с переменными. Ниже на рисунке приведен пример.

```

dr-rabbit@ART:~$ NAME="Пётр"
dr-rabbit@ART:~$ echo $NAME
Пётр
dr-rabbit@ART:~$ echo ${NAME}
Пётр
dr-rabbit@ART:~$ echo "Вокруг столько строк $NAME1, держитесь! Еще нужно город основать!"
Вокруг столько строк , держитесь! Еще нужно город основать!
dr-rabbit@ART:~$ echo "Вокруг столько строк ${NAME}1, держитесь! Еще нужно город основать!"
Вокруг столько строк Пётр1, держитесь! Еще нужно город основать!
dr-rabbit@ART:~$
dr-rabbit@ART:~$ echo "Если не знаем фамилию, то она точно ${LAST_NAME_DID_NOT_SET:-Романов}"
Если не знаем фамилию, то она точно Романов
dr-rabbit@ART:~$ |

```

Задав переменную, мы можем ее просто вывести, обратившись двумя способами. Но если мы хотим прямо при выводе прибавить к строке в переменной строку «1», то начинаются проблемы. Конструкция `${}` позволяет их решить.

В последнем примере с помощью конструкции `:-` задано значение переменной по умолчанию.

В переменную можно записать результат выполнения команды. На рисунке ниже приведен пример когда вывод команды `uname` помещается в переменную `horrey`. Обратите внимание, что в команде `echo` использован ключ `-e` и эскейп-последовательность `"\n"`, делающая перенос строки.

```

dr-rabbit@ART:~$ uname -o
GNU/Linux
dr-rabbit@ART:~$ horrey=$(uname -o)
dr-rabbit@ART:~$ echo -e "Ничего на свете лучше нету, чем ${horrey}! \nЛинус Торвальдс"
Ничего на свете лучше нету, чем GNU/Linux!
Линус Торвальдс
dr-rabbit@ART:~$ |

```

Для арифметических операций используется конструкция `$(())`.

Проще показать на примере, чем объяснять. Вот классический пример:

```

#!/bin/bash
# Задание значений переменных
num1=10
num2=5
# Выполнение арифметических операций
sum=$((num1 + num2))
difference=$((num1 - num2))
product=$((num1 * num2))
quotient=$((num1 / num2))
remainder=$((num1 % num2))
# Вывод результатов
echo "Сумма: $sum"
echo "Разность: $difference"
echo "Произведение: $product"
echo "Частное: $quotient"
echo "Остаток от деления: $remainder"

```

Учитывается порядок операций и скобочки:

```

result=$(( (5 + 3) * 2 ))
echo $result # Вывод: 16

```

Это здорово, но все операции в bash **целочисленные**. Если нужно считать что-то сложное, то используйте утилиту bc (https://www.gnu.org/software/bc/manual/html_mono/bc.html). Например, вот деление с точностью до 4-го знака после запятой.

```
echo 'scale = {{4}}; {{5 / 3}}' | bc
```

В bash есть условный оператор с набором сравнений целых чисел: -eq – равно, -ne - не равно, -lt – меньше, -le - меньше или равно, -gt – больше и -ge - больше или равно:

```
if [ "$variable" -eq 10 ]; then
    echo "Variable is 10"
else
    echo "Variable is not 10"
fi
```

Также существует оператор множественного выбора case .

```
case переменная in
    значение1 )
        команда 1
        ;;
    значение2 )
        команда 2
        ;;
esac
```

Вот пример:

```
echo "Выберите вариант:"
echo "1. Приветствие"
echo "2. Дата и время"
echo "3. Выход"
read -p "Введите ваш выбор (1-3): " choice
case $choice in
    1)
        echo "Привет! Как дела?"
        ;;
    2)
        echo "Текущая дата и время: $(date)"
        ;;
    3)
        echo "Выход из программы. Пока!"
        ;;
    *)
        echo "Неверный выбор! Пожалуйста, выберите
вариант от 1 до 3."
        ;;
esac
```

Кстати, в этом примере показано как читать ввод со строки в переменную с именем choice с помощью команды read.

Есть и циклы. For используется для повторения команд для каждого элемента в списке.

```
for i in 1 2 3; do
    echo "Number: $i"
done
```

или так, с использованием утилиты seq, когда нужно указать большие диапазоны:

```
for i in $(seq 1 100); do
    echo "Number: $i"
done
```

While выполняет команды до тех пор, пока условие истинно.

```
while [ "$count" -lt 5 ]; do
    echo "Count: $count"
    ((count++))
done
```

Для управления ходом выполнения цикла служат команды break и continue. Они точно соответствуют своим аналогам в других языках программирования. Команда break прерывает исполнение цикла, в то время как continue передает управление в начало цикла, минуя все последующие команды в теле цикла.

Можно разработать свою функцию и переиспользовать код или сделать код более читаемым. Например:

```
my_function() {
    echo "This is a function"
}
my_function
```

А вот пример определения и вызова функции с параметрами

```
# Определение функции
greet() {
    local name=$1    # Первый параметр
    local age=$2     # Второй параметр
    echo "Привет, $name! Вам $age лет."
}

# Вызов функции с параметрами
greet "Алексей" 25
```

Инструментальные средства:

Утилита для расчётов с плавающей точкой: bc

Прочие утилиты: tail, head, cat

Файлы: .bashrc

Утилиты работы с текстом: echo, grep

Редакторы: nano

Порядок выполнения работы:

Часть 1. Работ в среде bash

1. Настройте приглашение ввода так, чтобы сначала выводилось время, потом имя пользователя, потом текущий каталог, потом символ \$.
2. Создайте алиас с произвольным коротким именем, который выводил на терминал строку приветствия «Have a nice day, loginName!», где loginName имя текущего пользователя.
3. Сделайте так, чтобы эти изменения сохранились при перезагрузке.

Часть 2. Простые задачи на bash

Напишите скрипты по задачам, приведенным ниже. Каждый скрипт назовите по script22N, где N номер задачи этой части.

1. В параметрах скрипта передаются две строки. Вывести сообщение о равенстве или неравенстве переданных строк.
2. В параметрах при запуске скрипта передаются три целых числа. Вывести максимальное из них.
3. Считывать строки с клавиатуры, пока не будет введена строка "q". После этого вывести последовательность считанных строк в виде одной строки.
4. Считывать с клавиатуры целые числа, пока не будет введено четное число. После этого вывести количество считанных чисел.
5. Создать текстовое меню с четырьмя пунктами. При вводе пользователем номера пункта меню происходит запуск редактора nano, редактора vi, браузера links или выход из меню. При запуске скрипта нужно проверить, установлены ли программы, а если нет, то выводить сообщение об их отсутствии и завершать скрипт.
6. Создать скрипт для чтения содержимого файла, имя которого передается в параметре скрипта. Вторым параметром надо передать ключ «h» или «t», которые заставят читать строки с начала файла или с конца. Третий параметр число этих строк. Следует подавить вывод ошибок, если файл не существует или недоступен.

Часть 3. Разработка скрипта расчета производительности

1. В этой части вам понадобится разработать скрипт с именем script23, который определяет производительность вашего CPU в FLOPS (Floating Point Operations Per Second) или «операции с плавающей запятой в секунду». Этот показатель используется для измерения вычислительной мощности компьютеров, когда требуется выполнять сложные математические операции с высокой точностью. Операции с плавающей запятой требуют больше ресурсов по сравнению с операциями с целыми числами. Измеряя производительность во FLOPS определяют сколько таких операций процессор может выполнить за одну секунду.
2. Создайте скрипт, который производит достаточно большое количество (не менее миллиона) операций умножения двух одинаковых нецелых чисел с точностью 10 знаков после запятой и, определив затраченное время в

секундах вычисляет значение FLOPS. Для расчётов следует использовать bc, но так, чтобы при расчетах ничего на консоль не выводилось. Вычисление следует оформить в виде функции. Время можно сохранять в переменные перед и после запуска функции.

Содержание отчета

Требуется подготовить отчеты в формате DOC\DOCX или PDF. Отчет содержит титульный лист, артефакты выполнения и ответы на вопросы и задания.

Артефакты:

1. Строку alias для Части 1 п.2
2. Все скрипты из Части 2 и 3.

Вопросы и задания:

1. Как определить сколько параметров пользователь передал в скрипт при запуске?
2. В bash есть инкремент и декремент:

```
(( count++ )) # Инкремент  
(( count-- )) # Декремент
```

Попробуйте запустить простой скрипт:

```
count=10  
echo $(( count-- ))  
echo $(( count-- ))
```

что оператор echo выведет в первый раз, что во второй? Почему?

3. Какие недостатки есть у нашего подхода к определению производительности в FLOPS?

Отчет выслать в течение 4-х недель на адрес edu-net@yandex.ru.

Поддержка работы

Дополнительные материалы по теме курса публикуются на Telegram-канале ITSMDao (t.me/itsmdao). Обсуждать работу и задавать вопросы можно в чате ITSMDaoChat (t.me/itsmdaochat).