МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Институт Компьютерных наук
Кафедра Прикладной математики

ЛАБОРАТОРНАЯ РАБОТА

По дисциплине: «Операционные системы Linux».
На тему: «Средства разработки программного обеспечения в ОС Linux».

Студент ПМ-22 (группа) _____ (подпись, дата) Борисов А.В. (фамилия, инициалы)

Руководитель

ктн (ученая степень, ученое звание) _____ (подпись, дата) Кургасов В.В. (фамилия, инициалы)

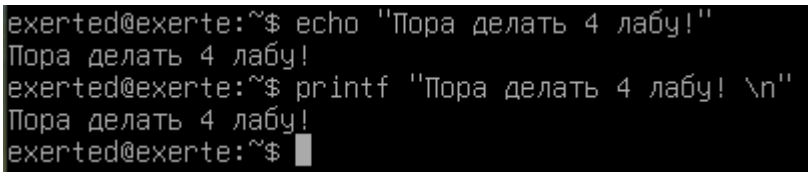
Липецк 2024

Цель

Изучить основные возможности языка программирования высокого уровня Shell. Получить навыки написания и использования скриптов.

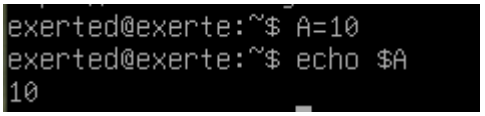
Ход работы

1. Используя команды echo, printf, вывести информационные сообщения на экран.

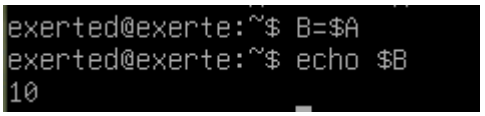


   Printf идеально подходит для ситуаций, когда требуется точный контроль над форматом вывода.

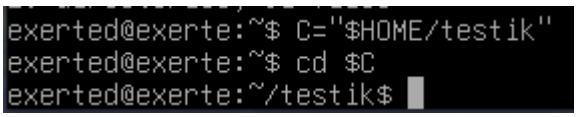
2. Присвоить переменной A целочисленное значение. Просмотреть значение переменной A.



3. Присвоить переменной B значение переменной A. Просмотреть значение переменной B.



4. Присвоить переменной C значение “путь до своего каталога”. Перейти в этот каталог с использованием переменной.



5. Присвоить переменной D значение “имя команды”, а именно, команды PASTE. Выполнить эту команду, используя значение переменной.

```
exerted@exerte:~$ D="paste"
exerted@exerte:~$ $D first_text.txt second_text.txt
Text    Text
from    from
first_text      second_text

exerted@exerte:~$
```

6. Присвоить переменной E значение “имя команды”, а именно, команды просмотра содержимого файла, просмотреть содержимое переменной. Выполнить эту команду, используя значение переменной.

```
exerted@exerte:~$ echo $E
cat
exerted@exerte:~$ $E first_text.txt
Text
from
first_text

exerted@exerte:~$
```

7. Присвоить переменной F значение “имя команды”, а именно, сортировки содержимого текстового файла. Выполнить эту команду, используя значение переменной.

```
exerted@exerte:~$ $F first_text.txt

first_text
from
Text
exerted@exerte:~$
```

Написать скрипты, при запуске которых выполняются следующие действия:

1) Программа запрашивает значение переменной, а затем выводит значение этой переменной:

```
#!/bin/bash

read -p "Введите значение переменной: " value

echo $value
```

```
exerted@exerte:~$ ./prog
Введите значение переменной: 12
12
```

2) Программа запрашивает имя пользователя, затем здоровается с ним, используя значение переменной:

```
#!/bin/bash

read -p "Как вас зовут? " value

echo "Здравствуйте, $value"
```

```
exerted@exerte:~$ ./prog
Как вас зовут? Александр
Здравствуйте, Александр
```

3) Программа запрашивает значения двух переменных, вычисляет сумму (разность, произведение, деление) этих переменных. Результат выводится на экран (использовать команды а) EXPR б) BC)

```
#!/bin/bash

read -p "Введите 2 числа: " num1 num2

echo "expr:"
expr $num1 + $num2
expr $num1 - $num2
expr $num1 / $num2
expr $num1 \* $num2
echo "bc:"
echo "$num1+$num2" | bc
echo "$num1-$num2" | bc
echo "$num1/$num2" | bc
echo "$num1*$num2" | bc
```

```
exerted@exerte:~$ ./prog
Введите 2 числа: 10 5
expr:
15
5
2
50
bc:
15
5
2
50
```

4) Вычислить объем цилиндра. Исходные данные запрашиваются программой. Результат выводится на экран:

```
#!/bin/bash

read -p "Введите высоту и радиус цилиндра: " h r

echo "3.14*$r^2*$h" | bc_
```

```
exerted@exerte:~$ ./prog
Введите высоту и радиус цилиндра: 10 7
1538.60
```

5) Используя позиционные параметры, отобразить имя программы, количество аргументов командной строки, значение каждого аргумента командной строки:

```
#!/bin/bash

all_args=("$@")
len=${#all_args[@]}
for ((i=0;i<$len;i++)); do
        echo $((i+1)) : ${all_args[${i}]}
done
```

```
exerted@exerte:~$ ./prog 1 2 3 4
1 : 1
2 : 2
3 : 3
4 : 4
```

6) Используя позиционный параметр, отобразить содержимое текстового файла, указанного в качестве аргумента командной строки. После паузы экран очищается:

```
#!/bin/bash

cat "$1"

sleep 5
clear_
```

```
exerted@exerte:~$ ./prog first_text.txt
Text
from
first_text
```

```
exerted@exerte:~$
```

7) Используя оператор FOR, отобразить содержимое текстовых файлов текущего каталога поэкранно:

```
#!/bin/bash

for file in *.txt; do
        echo "in file $file: \n"
        more "$file"
done
```

```
in file first_text.txt
Text
from
first_text

(END)
```

8) Программой запрашивается ввод числа, значение которого затем сравнивается с допустимым значением. В результате этого сравнения на экран выдаются соответствующие сообщения:

```
#!/bin/bash

read -p "Введите чилсо от 1 до 100: " num
if !  [[ "$num" =~ ^[0-9]+$ ]]; then
        echo "Введите числовое значение!"
        exit 1
fi

min=1
max=100


if (( $num<$min )) then
        echo "Число меньше допустимого значения!"
elif (( $num>$max )) then
        echo "Число больше допустимого значения!"
fi
```

```
exerted@exerte:~$ ./prog
Введите чилсо от 1 до 100: 3
exerted@exerte:~$ ./prog
Введите чилсо от 1 до 100: 0
Число меньше допустимого значения!
exerted@exerte:~$ ./prog
Введите чилсо от 1 до 100: qwe
Введите числовое значение!
```

Было реализована проверка введенного значения на число и его промежуток.

9) Программой запрашивается год, определяется, високосный ли он. Результат выдается на экран:

```
fi


if (( $year%4==0 && $year%100!=0 )) || (($year%400==0)); then
        echo "Является високосным"
else
        echo "Не является високосным"
fi
~
~
~
~
~
~
```

```
exerted@exerte:~$ ./prog
Введите год: 2000
Является високосным
exerted@exerte:~$ ./prog
Введите год: 2001
Не является високосным
```

10) Вводятся целочисленные значения двух переменных. Вводится диапазон данных. Пока значения переменных находятся в указанном диапазоне, их значения инкрементируются.

```
#!/bin/bash

read -p "Введите 2 числа: " num1 num2
read -p "Введите диапозон значений: " min max

while (( num1 >= min && num1 <= max && num2 >= min && num2 <= max )); do
        echo "num1 = $num1, num2 = $num2"
        ((num1++))
        ((num2++))
done
echo "Цикл завершен: Переменные вышли за пределы диапозона."
```

```
Введите 2 числа: 0 0
Введите диапозон значений: 0 25
num1 = 0, num2 = 0
num1 = 1, num2 = 1
num1 = 2, num2 = 2
num1 = 3, num2 = 3
num1 = 4, num2 = 4
num1 = 5, num2 = 5
num1 = 6, num2 = 6
num1 = 7, num2 = 7
num1 = 8, num2 = 8
num1 = 9, num2 = 9
num1 = 10, num2 = 10
num1 = 11, num2 = 11
num1 = 12, num2 = 12
num1 = 13, num2 = 13
num1 = 14, num2 = 14
num1 = 15, num2 = 15
num1 = 16, num2 = 16
num1 = 17, num2 = 17
num1 = 18, num2 = 18
num1 = 19, num2 = 19
num1 = 20, num2 = 20
num1 = 21, num2 = 21
num1 = 22, num2 = 22
num1 = 23, num2 = 23
num1 = 24, num2 = 24
num1 = 25, num2 = 25
Цикл завершен: Переменные вышли за пределы диапозона.
```

11) В качестве аргумента командной строки указывается пароль. Если пароль введен верно, постранично отображается в длинном формате с указанием скрытых файлов содержимое каталога /etc.

```
#!/bin/bash

correct_password="Topparol"

if (( "$1" == "$correct_password" )); then
        echo "Пароль верен."
        ls /etc | less
else
        echo "Праль неверен"
        exit 1
fi
```

12) Проверить, существует ли файл. Если да, выводится на экран его содержимое, если нет - выдается соответствующее сообщение.

```
#!/bin/bash

if [[ -f "$1" ]]; then
        echo "Файл '$1' существует"
        cat "$1"
else
        echo "Файл '$1' не существует"
        exit 1
fi
```

```
exerted@exerte:~$ ./prog first_text.txt
Файл 'first_text.txt' существует
Text
from
first_text
```

13) Если файл есть каталог и этот каталог можно читать, просматривается содержимое этого каталога. Если каталог отсутствует, он создается. Если файл не есть каталог, просматривается содержимое файла.

```
#!/bin/bash

if [ -d "$1" ]; then
        if [ -r "$1" ]; then
                echo "Каталог '$1' существует и доступен для чтения. Содержимое каталога: "
                ls "$1"
        else
                echo "Каталог '$1' существует, но не доступен для чтения."
                exit 1
        fi
else
        if [ -e "$1" ]; then
                echo "Файл '$1' существует. Его содержимое: "
                cat "$1"
        else
                echo "Каталог '$1' не существует."
                mkdir "$1"
                echo "Каталог '$1' создан."
        fi
fi
```

```
exerted@exerte:~$ ./prog "Документы"
Каталог 'Документы' существует и доступен для чтения. Содержимое каталога:
exerted@exerte:~$
```

14) Анализируются атрибуты файла. Если первый файл существует и используется для чтения, а второй файл существует и используется для записи, то содержимое первого файла перенаправляется во второй файл. В случае несовпадений указанных атрибутов или отсутствия файлов на экран выдаются

соответствующие сообщения (использовать имена файлов и/или позиционные параметры).

```
#!/bin/bash

if [[ $# -ne 2 ]]; then
        echo "Передайте 2 файла в качестве параметров!"
        exit 1
fi


if [[ -e "$1" && -r "$1" ]]; then
        echo "Первый файл '$1' существует и доступен для чтения."
else
        echo "Ошибка: Первый файл '$1' не доступен для чтения либо не существует"
        exit 1
fi

if [[ -e "$2" && -w "$2" ]]; then
        echo "Второй файл '$2' существует и доступен для записи"
else
        echo "Второй файл '$2' не доступен для записи или не существует"
        exit 1
fi

cat "$1" > "$2"
```

```
exerted@exerte:~$ ./prog input.txt output_file
Первый файл 'input.txt' существует и доступен для чтения.
Второй файл 'output_file' существует и доступен для записи
exerted@exerte:~$ cat output_file
Это содержимое первого файла
```

15) Если файл запуска программы найден, программа запускается (по выбору).

```
#!/bin/bash

if [[ -x "$1"]]; then
        echo "Программа '$1' найдена и является исполняемой"
        read -p "Хотите запустить программу '$1'? (y/n): " choice
        if [[ "$choice" == "y" ]]; then
                echo "Запуск программы '$1'..."
        else
                echo "Программа '$1' не будет запущена"
        fi
else
        echo "Ошибка: файл '$1' не найден или не является исполняемым"
        exit 1
fi
```

```
exerted@exerte:~$ ./prog 1.txt
Ошибка: файл '1.txt' не найден или не является исполняемым
```

16) В качестве позиционного параметра задается файл, анализируется его размер. Если размер файла больше нуля, содержимое файла сортируется по первому столбцу по

возрастанию, отсортированная информация помещается в другой файл, содержимое которого затем отображается на экране.

```
#!/bin/bash

if ! [ -e "$1" ]; then
        echo "Ошибка, файл '$1' не нсуществует"
        exit 1
fi

size=$(stat -c %s "$1")

if [ $size -gt 0 ]; then
        echo "Размер файла '$1' составляет $size байт"

        sorted_file="sorted_$(basename "$1")"
        sort -k1,1 "$1" > "$sorted_file"

        echo "Отсортированое содержимое файла 'sorted_file'"
        cat "$sorted_file"
else
        echo "Ошибка, файл '$1' пустой"
        exit 1
fi
```

```
exerted@exerte:~$ cat first_text.txt
banana 3
cucummber 2
apple 1
exerted@exerte:~$ ./prog first_text.txt
Размер файла 'first_text.txt' составляет 29 байт
Отсортированое содержимое файла 'sorted_file'
apple 1
banana 3
cucummber 2
exerted@exerte:~$
```

Вывод

В ходе данной лабораторной работы были изучены основные возможности языка программирования высокого уровня Shell, были написаны простейшие скрипты.

Контрольные вопросы

1. **В чём отличие пользовательских переменных от переменных среды?**

   Пользовательские переменные доступны только в текущем сеансе или скрипте, переменные среды — глобальны, передаются дочерним процессам.

2. **Математические операции в SHELL.**

   Математические операции: выполняются с помощью `$(( ... ))`, `let`, `expr`, и `bc` для сложных операций.

3. **Условные операторы в SHELL.**

   Условные операторы: `if`, `elif`, `else`, `case` для проверки условий.

4. **Принципы построения простых и составных условий.**

   AND (&&): оба условия должны быть истинными.

   OR (`||`): хотя бы одно условие должно быть истинным

   NOT (`!`): инвертирует условие.

5. **Циклы в SHELL.**

   Циклы в SHELL: `for`, `while`, `until` для повторения команд.

6. **Массивы и модули в SHELL.**

   Массивы и модули: массивы хранят несколько значений, модули подгружаются с помощью `. <имя_модуля>`.

7. **Чтение параметров командной строки.**

   Чтение параметров командной строки: параметры доступны как $1, $2, ... и $@.

8. **Как различать ключи и параметры?**

   Ключи и параметры: ключи часто начинаются с `-` или `--`, параметры — просто значения.

9. **Чтение данных из файлов.**

   Чтение данных из файлов: через команды `cat`, `while read line`, `awk`, `sed`.

10. **Стандартные дескрипторы файлов**

    Стандартные дескрипторы файлов: `0` (ввод), 1 (вывод), 2 (ошибки).

11. Перенаправление вывода.

Перенаправление вывода: >, >> для вывода, < для ввода.

12. Подавление вывода.

перенаправление в `/dev/null`.

13. Отправка сигналов скриптам.

Отправка сигналов: `kill`, `trap` для управления процессами.

14. Использование функций.

Использование функций: функции создаются с `function имя { ... }` и вызываются по имени.

15. Обработка текстов (чтение, выбор, вставка, замена данных).

Обработка текстов: `grep`, `sed`, `awk` для поиска и замены данных.

16. Отправка сообщений в терминал пользователя.

Отправка сообщений в терминал: `echo`, `wall` для отправки пользователям.

17. BASH и SHELL – синонимы?

Bash и Shell: `bash` — это тип `shell` с дополнительными функциями.

18. PowerShell в операционных системах семейства Windows: назначение и особенности.

PowerShell в Windows: расширенная оболочка с поддержкой объектов и скриптов, отличается от Bash.