



**Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

Студент Равашдех Фадей Хешамович

Группа ИУ7-15Б

Тип практики Проектно-технологическая практика

Название предприятия НУК ИУ МГТУ им. Н. Э. Баумана

Студент _____ Равашдех Ф Х

Руководитель практики _____ Ломовской И. В.

Руководитель практики _____ Кострицкий А. С.

Оценка _____

2022 г.

Оглавление

ВВЕДЕНИЕ

Задача: разработать скрипты командной оболочки для сравнения содержимого двух текстовых файлов по определённым правилам.

1. Последовательности целых чисел;
2. Текст после первого вхождения подстроки «string:»;
3. Сравниваются последовательности чисел с плавающей точкой, записанных не в экспоненциальной форме.

Декомпозиция: Декомпозиция на подзадачи является общей для всех скриптов. Так, скрипт можно разбить на следующие подзадачи:

1. Считывание аргументов
2. Проверка аргументов
3. Создание временных файлов и дескрипторов
4. Обработка текста из файлов
5. Сравнение

Сценарий командной оболочки

Скрипт №1

Данный скрипт нужен для нахождения и сравнения числовых последовательностей, находящихся в двух файлах.

Скрипт может принимать в себя от двух до трёх аргументов:

1. Путь к первому файлу
2. Путь ко второму файлу
3. Необязательный аргумент -v, при котором программа будет выводить подробности выполнения.

Особенности реализации:

1. Создание функций для вывода, считывание введённых аргументов

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

```
echo_verbose () {
    if [ $v -eq 1 ]; then
        echo $1
    fi
}

usage () {
    [ $v -eq 1 ] && echo "Использование: $PROGNAME file1 file2 [-v]"
    exit 2
}

# parameters
FILE1=""
FILE2=""
v=0
while [[ -n $1 ]]; do
    if [ $1 == "-v" ]; then
        v=1
        break
    elif [ -z $FILE1 ]; then
        FILE1=$1
        shift
    elif [ -z $FILE2 ]; then
        FILE2=$1
        shift
    else
        usage
    fi
done
```

2. Обработка переданных аргументов: проверка существования файлов и прав на их чтение.

```
# check filenames
if [ -z $FILE1 ]; then
    usage
elif [ -z $FILE2 ]; then
    usage
fi

# check file exists
[ ! -f $FILE1 ] && exit 3
[ ! -f $FILE2 ] && exit 3

# check readable
[ ! -r $FILE1 ] && exit 3
[ ! -r $FILE2 ] && exit 3
```

3. Создание двух временных файлов для хранения в них последовательностей, извлечённых из файлов; назначение дескрипторов.

```
# temp files
temp=$(basename $0)
TEMPFILE1=$(mktemp /tmp/"$temp"1.XXXXXX)
trap "rm -f $TEMPFILE1" EXIT SIGKILL
TEMPFILE2=$(mktemp /tmp/"$temp"2.XXXXXX)
trap "rm -f $TEMPFILE2" EXIT SIGKILL

# descriptors
exec 4<>$FILE1
exec 5<>$FILE2
exec 6<>$TEMPFILE1
exec 7<>$TEMPFILE2
```

4. Обработка текста из файлов. поиск чисел с помощью регулярных выражений. Запись этих чисел во временные файлы.

```
# fill tempfile1
while read -u 4 -r || [[ -n $REPLY ]]; do
    for word in $REPLY; do
        if echo "$word" | grep -Eq "^[+-]?[0-9][0-9]*$"; then
            echo_verbose $word
            echo "$word " >&6
        fi
    done
done

# fill tempfile2
while read -u 5 -r || [[ -n $REPLY ]]; do
    for word in $REPLY; do
        if echo "$word" | grep -Eq "^[+-]?[0-9][0-9]*$"; then
            echo_verbose $word
            echo "$word " >&7
        fi
    done
done
```

done

5. Сравнение временных файлов, содержащих в себе числовые последовательности из файлов.

```
# compare tempfiles
opt='-s'
if [ $v -eq 1 ]; then
    opt=''
fi
if cmp $opt $TEMPFILE1 $TEMPFILE2; then
    echo_verbose "Последовательности совпали"
    exit 0
else
    echo_verbose "Последовательности не совпали"
    exit 1
fi
```

Тестирование:

Пройдено 7 тестов для проверки корректной работы скрипта.

Результат тестирования:

1. ok
2. ok
3. ok
4. ok
5. ok
6. ok
7. ok

Скрипт №2

Данный скрипт нужен для нахождения и сравнения текста в двух файлах, .

Скрипт может принимать в себя от двух до трёх аргументов:

4. Путь к первому файлу
5. Путь ко второму файлу
6. Необязательный аргумент -v, при котором программа будет выводить подробности выполнения.

Особенности реализации:

1. Создание функций для вывода, считывание введённых аргументов

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

```
echo_verbose () {  
    if [ $v -eq 1 ]; then  
        echo $1  
    fi  
}  
  
usage () {  
    [ $v -eq 1 ] && echo "Использование: $PROGNAME file1 file2 [-v]"  
    exit 3  
}  
  
# parameters  
FILE1=""  
FILE2=""  
v=0  
while [[ -n $1 ]]; do  
    if [ $1 == "-v" ]; then  
        v=1  
        break  
    elif [ -z $FILE1 ]; then  
        FILE1=$1  
        shift  
    elif [ -z $FILE2 ]; then  
        FILE2=$1  
        shift  
    else  
        usage  
    fi  
done
```

2. Обработка переданных аргументов: проверка существования файлов и прав на их чтение.

```
# check filenames
if [ -z $FILE1 ]; then
    usage
elif [ -z $FILE2 ]; then
    usage
fi

# check file exists
[ ! -f $FILE1 ] && exit 3
[ ! -f $FILE2 ] && exit 3

# check readable
[ ! -r $FILE1 ] && exit 3
[ ! -r $FILE2 ] && exit 3
```

3. Создание двух временных файлов для хранения в них последовательностей, извлечённых из файлов; назначение дескрипторов.

```
# temp files
temp=$(basename $0)
TEMPFILE1=$(mktemp /tmp/"$temp"1.XXXXXX)
trap "rm -f $TEMPFILE1" EXIT SIGKILL
TEMPFILE2=$(mktemp /tmp/"$temp"2.XXXXXX)
trap "rm -f $TEMPFILE2" EXIT SIGKILL

# descriptors
exec 4<>$FILE1
exec 5<>$FILE2
exec 6<>$TEMPFILE1
exec 7<>$TEMPFILE2
```

4. Обработка текста из файлов, поиск подстроки «string:» с помощью grep. Нахождение второй части строки (после подстроки) и последующих строк текста. Запись этих чисел во временные файлы.

```
# fill tempfile1
flag=0
while read -u 4 -r || [[ -n $REPLY ]]; do
    if [ $flag -eq 0 ]; then
        if echo $REPLY | grep -q "string"; then
            LINE=$REPLY
            LINESTART=""
            LINEEND=""
            while read -n 1; do
                LINESTART=$LINESTART$REPLY
                if echo $LINESTART | grep -q "string:"; then
                    LINEEND=$LINEEND$REPLY
                fi
            done <<< $LINE
            echo_verbose $LINEEND
            echo $LINEEND >&6
            flag=1
        fi
    elif [ $flag -eq 1 ]; then
        echo_verbose $REPLY
        echo $REPLY >&6
    fi
done

# fill tempfile2
flag=0
while read -u 5 -r || [[ -n $REPLY ]]; do
    if [ $flag -eq 0 ]; then
        if echo $REPLY | grep -q "string"; then
            LINE=$REPLY
            LINESTART=""
            LINEEND=""
            while read -n 1; do
                LINESTART=$LINESTART$REPLY
                if echo $LINESTART | grep -q "string:"; then
                    LINEEND=$LINEEND$REPLY
                fi
            done <<< $LINE
            echo_verbose $LINEEND
            echo $LINEEND >&7
            flag=1
        fi
    elif [ $flag -eq 1 ]; then
        echo_verbose $REPLY
        echo $REPLY >&7
    fi
done
```

5. Сравнение временных файлов, содержащих в себе числовые последовательности из файлов.

```
# compare tempfiles
opt='-s'
if [ $v -eq 1 ]; then
    opt=''
fi
if cmp $opt $TEMPFILE1 $TEMPFILE2; then
    echo_verbose "Последовательности совпали"
    exit 0
else
    echo_verbose "Последовательности не совпали"
    exit 1
fi
```

Тестирование:

Пройдено 5 тестов для проверки корректной работы скрипта.

Результат тестирования:

1. ok
2. ok
3. ok
4. ok
5. ok

Скрипт №3

Данный скрипт нужен для нахождения и сравнения числовых последовательностей, находящихся в двух файлах.

Скрипт может принимать в себя от двух до трёх аргументов:

7. Путь к первому файлу
8. Путь ко второму файлу
9. Необязательный аргумент -v, при котором программа будет выводить подробности выполнения.

Особенности реализации:

1. Создание функций для вывода, считывание введённых аргументов

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

```
echo_verbose () {  
    if [ $v -eq 1 ]; then  
        echo $1  
    fi  
}  
  
usage () {  
    [ $v -eq 1 ] && echo "Использование: $PROGNAME file1 file2 [-v]"  
    exit 2  
}  
  
# parameters  
FILE1=""  
FILE2=""  
v=0  
while [[ -n $1 ]]; do  
    if [ $1 == "-v" ]; then  
        v=1  
        break  
    elif [ -z $FILE1 ]; then  
        FILE1=$1  
        shift  
    elif [ -z $FILE2 ]; then  
        FILE2=$1  
        shift  
    else  
        usage  
    fi  
done
```

2. Обработка переданных аргументов: проверка существования файлов и прав на их чтение.

```
# check filenames
if [ -z $FILE1 ]; then
    usage
elif [ -z $FILE2 ]; then
    usage
fi

# check file exists
[ ! -f $FILE1 ] && exit 3
[ ! -f $FILE2 ] && exit 3

# check readable
[ ! -r $FILE1 ] && exit 3
[ ! -r $FILE2 ] && exit 3
```

3. Создание двух временных файлов для хранения в них последовательностей, извлечённых из файлов; назначение дескрипторов.

```
# temp files
temp=$(basename $0)
TEMPFILE1=$(mktemp /tmp/"$temp"1.XXXXXX)
trap "rm -f $TEMPFILE1" EXIT SIGKILL
TEMPFILE2=$(mktemp /tmp/"$temp"2.XXXXXX)
trap "rm -f $TEMPFILE2" EXIT SIGKILL

# descriptors
exec 4<>$FILE1
exec 5<>$FILE2
exec 6<>$TEMPFILE1
exec 7<>$TEMPFILE2
```

4. Обработка текста из файлов. поиск чисел с помощью регулярных выражений. Запись этих чисел во временные файлы.

```
# fill tempfile1
while read -u 4 -r || [[ -n $REPLY ]]; do
    for word in $REPLY; do
        if echo "$word" | grep -Eq "^[+-]?[0-9][.]?[0-9]*$"; then
            echo_verbose $word
            echo "$word " >&6
        fi
    done
done

# fill tempfile2
while read -u 5 -r || [[ -n $REPLY ]]; do
    for word in $REPLY; do
        if echo "$word" | grep -Eq "^[+-]?[0-9][.]?[0-9]*$"; then
            echo_verbose $word
            echo "$word " >&7
        fi
    done
done
```

done

5. Сравнение временных файлов, содержащих в себе числовые последовательности из файлов.

```
# compare tempfiles
opt='-s'
if [ $v -eq 1 ]; then
    opt=''
fi
if cmp $opt $TEMPFILE1 $TEMPFILE2; then
    echo_verbose "Последовательности совпали"
    exit 0
else
    echo_verbose "Последовательности не совпали"
    exit 1
fi
```

Тестирование:

Пройдено 7 тестов для проверки корректной работы скрипта.

Результат тестирования:

1. ok
2. ok
3. ok
4. ok
5. ok
6. ok
7. ok

ЗАКЛЮЧЕНИЕ

В результате задача решена, необходимые скрипты написаны и проверены.
Задание выполнено.