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 **Беляев Никита Александрович**

Группа **ИУ7-11Б**

Тип практики **Проектно-технологическая практика**

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

Студент _____ **Беляев Н А**

Руководитель практики _____ **Ломовской И. В.**

Руководитель практики _____ **Кострицкий А. С.**

Оценка _____

2022 г.

Оглавление

1. Введение

1.1 Постановка задачи

Целью задания является написание четырех различных скриптов командной оболочки BASH, осуществляющих сравнение содержимого двух текстовых файлов по следующим параметрам:

1. Сравнение по последовательностям знаковых целых чисел в файлах.
   - Числа входят в диапазон 4-х байт (INT C)
   - В последовательность для сравнения входят только те целые числа, что отделены от иных символов в файле пробельными символами.

2. Сравнение по последовательностям символов в файлах после первого вхождения подстроки “string:”.
   - Подстрока “string:” может являться как отдельным словом (Отделена от других символов пробелами), так и находиться внутри других символов (Пр.

123wwstring:p).

- Сравнение дальнейших символов строгое, с учетом пробельных и служебных (Пр. “\n”) символов.

3. Сравнение по последовательностям знаковых ЧПТ (Чисел с плавающей точкой / Floating point numbers), записанных в стандартной, не экспоненциальной, форме.
   - Числа входят в диапазон двойной точности (DOUBLE C)

4. Сравнение по последовательностям знаковых ЧПТ (Чисел с плавающей точкой / Floating point numbers), записанных в том числе в экспоненциальной форме.
   - Числа входят в диапазон двойной точности (DOUBLE C)

(!) Пункты 1, 3, 4 подразумевают сравнение чисел как строк (100 != 0100; 7.1 != 7.10; 10e-1 != 1.0 != 1).

1.2 Специфика реализации

Для каждого варианта сравнения необходимо реализовать:

1. Скрипт comparator[1-4].sh
   - Строка запуска в терминале: ./comparator[1-4].sh file_name1 file_name2

2. Скрипт test[1-4].sh
   - Базовый функционал данного скрипта аналогичен функционалу

соответствующего скрипта comparator[1-4].sh.

- Данный скрипт работает в подробном режиме, т.е. демонстрирует специфику работы алгоритма и обработку случаев, способных нарушить корректную работу скрипта.

3. Каталог с тестовыми текстовыми файлами tests для работы скрипта test[1-4].sh

- Цифра из диапазона [1-4] соответствует номеру соответствующего варианта сравнения из П. “Постановка задачи”

1.3 Подзадачи, необходимые для реализации работы:

1. Проанализировать различные методы взаимодействия с текстовыми файлами с помощью командной оболочки BASH.

2. На основе изученных методов разработать алгоритм, позволяющий производить сравнение текстовых файлов по необходимым характеристикам.

3. Реализовать соответствующее ПО.

4. Провести тестирование ПО.

1.4 Справка по структуре отчета

В основной части отчета приведены:

1. Описание принципа работы алгоритма сравнения текстовых файлов, в данном случае являющегося инвариантом.
2. Листинги каждого из скриптов comparator[1-4].sh с соответствующими пояснениями и демонстрацией избранный результатов работы тестов.

2. Основная часть

2.1 Описание общего алгоритма

Для вариантов сравнения 1, 3, 4:

1. В начале каждой программы с помощью условного оператора проверяется корректность строки запуска скрипта - наличие валидных файлов file_name1 и file_name2. В случае отсутствия одного из файлов в консоль выводится сообщение “Некорректная строка запуска (При запуске укажите оба файла)”, и программа завершается.

2. Если файлы указаны верно, то для каждого из 2-х файлов последовательно выполняется следующий алгоритм (Далее для файла):
   2.1. Создается временный текстовый файл (Буфер)
   2.2. Циклом while итерируемся по строкам исходного файла
   2.3. Внутри while циклом for итерируемся по каждому слову (Последовательности символов, отделенной от остального текста пробельными символами)
   2.4. Сравниваем каждое “выцепленное” слово с шаблоном (Регулярным выражением)

2.5. Если слово удовлетворяет шаблону, то мы записываем это слово в буферный файл

3. В результате п.2 имеется два буферных текстовых файла, содержащих только необходимые для сравнения последовательности символов.

4. Сравниваем буферные файлы и удаляем их.

5. Совпадение содержимого кодируется кодом возврата “0”, несовпадение “1”.

Для варианта сравнения 2:

Принцип “сканирования” файла по словам тот же, однако при первом вхождении подстроки “string:” мы:

1. Перезаписываем текущую строку, начиная от первого вхождения подстроки “string:” в буфер.

2. Добавляем все последующие строки в буфер, учитывая количество пробелов между символами.

3. Сравниваем буферные файлы и удаляем их.

4. Совпадение содержимого кодируется кодом возврата “0”, несовпадение “1”.

(!) Уточнение.

- Скрипты comparator[1-4].sh поддерживают работу только в молчаливом режиме. Их функционал и прохождение текстов продемонстрированы в соответствующих скриптах test[1-4].sh
- Буферные текстовые файлы создаются в каталоге **/tmp** во избежание конфликта с ОС при запуске скрипта не от имени root пользователя.

2.2. Первый вариант сравнения (По INT).

Листинг скрипта comparator1.sh

```
#!/bin/bash

regex="^-?[0-9]+$"
file_1=$1
file_2=$2
touch /tmp/buffer_1
touch /tmp/buffer_2

if [ -z "$file_1" -o -z "$file2" ]; then
      echo "Некорректная строка запуска (При запуске укажите оба файла)."
else
      while IFS= read -r line
      do
            for word in $line; do
                  if [[ $word =~ $regex ]]; then
                        echo $word >> /tmp/buffer_1
                  fi
```

```
        done
    done < "$file_1"

    while IFS= read -r line
    do
        for word in $line; do
            if [[ $word =~ $regex ]]; then
                echo $word >> /tmp/buffer_2
            fi
        done
    done < "$file_2"

    if cmp -s /tmp/buffer_1 /tmp/buffer_2; then
        echo 0
    else
        echo 1
    fi
    rm /tmp/buffer_1 /tmp/buffer_2
fi
```

Описание скрипта comparator1.sh

- В переменные file_1 и file_2 передаются позиционные параметры, указанные в строке запуска скрипта в терминале.
- Параметр IFS= обрезает лишние пробельные символы в начале и конце каждой строки.
- Оператор “=~” используется для сопоставления текущего слова и регулярного выражения.

- Расшифровка регулярного выражения:

  Потенциальный минус в начале; одна или больше цифр далее и до конца.

Демонстрация тестирования:

В левой колонке приведена обрабатываемая строка, а в правой - результат ее обработки скриптом (Что будет добавлено в буфер для последующего сравнения)

Строка	Результат обработки
1101 Hello! 0	1101 0
Th77is -1 tex-9t 8contains5	-1
1 -2 3 -4 5 -6	1 -2 3 -4 5 -6

Результат работы test1.sh: 1 (Целочисленные последовательности различны)

Комментарий к работе теста:

Тестировочный скрипт выводит содержимое исходных файлов, далее текущую строку, и, если в текущей строке обнаружено совпадение с шаблоном, то после строки “Caught:” выводится совпадение, которое будет перенаправлено в буфер.

Скрипт успешно:

- Обрабатывает Знаковые целые числа, отделенные пробельными символами вне зависимости от местонахождения в контексте строки (Начало/конец/середина).

- Игнорирует целые числа, которые НЕ отделены от иного текста пробельными символами.

2.2. Второй вариант сравнения (После “string:”).

Листинг скрипта comparator2.sh

```
#!/bin/bash

string="string:"
file_1=$1
file_2=$2
touch /tmp/buffer_1
touch /tmp/buffer_2

if [ -z "$file_1" -o -z "$file2" ]; then
      echo "Некорректная строка запуска (При запуске укажите оба файла)."
else
      flag=false
      while read -r line
      do
            if [ "$flag" = "true" ]; then
                  echo "$line" >> /tmp/buffer_1
            else
                  for word in $line; do
                        if [[ $word =~ $string ]]; then
                              edited=$(sed 's/SeqStart/NotSeq/;
s/string:/SeqStart/' <<< "$line")
```

```
                              (sed -n -e "s/^.*\(SeqStart.*\)/\1/p" <<<
"$edited") > /tmp/buffer_1
                              flag=true
                              break
                        fi
                  done
            fi
      done < "$file_1"

      flag=false
      while read -r line
      do
            if [ "$flag" = "true" ]; then
                  echo "$line" >> /tmp/buffer_2
            else
                  for word in $line; do
                        if [[ $word =~ $string ]]; then
                              edited=$(sed 's/SeqStart/NotSeq/;
s/string:/SeqStart/' <<<"$line")
                              (sed -n -e "s/^.*\(SeqStart.*\)/\1/p" <<<
"$edited") > /tmp/buffer_2
                              flag=true
                              break
                        fi
                  done
            fi
      done < "$file_2"

if [ -s /tmp/buffer_1 -o -s /tmp/buffer_2 ]; then
      if cmp -s /tmp/buffer_1 /tmp/buffer_2; then
```

```
            echo 0
      else
            echo 1
      fi
else
      echo "Ни в одном из файлов нет подстроки 'string:'."
fi

rm /tmp/buffer_1 /tmp/buffer_2
```

Описание скрипта comparator2.sh

- Переменная “flag” имеет значение “false” по умолчанию. При данном ее значении каждая строка файла сканируется пословно. При обнаружении подстроки “string:” она переходит в значение “true” и все последующие строки копируются в буфер без “пословного расчленения”, так как нам больше не важно их конкретное содержимое.
- При использовании команды “sed” для замены подстрок в строке задействован оператор “<<<”, направляющий в “sed” конкретную строку, а не файл.
- При обнаружении первого вхождения подстроки “string:”:

  1. Выполняется следующая замена: `s/string:/SeqStart/`

  2. С помощью команды (`sed -n -e "s/^.*\(SeqStart.*\)/\1/p" <<< "$edited"`) > /tmp/buffer_2 выполняется копирование в буфер только той части строки, которая следует после вхождения подстроки “SeqStart” (Т.е. После подстроки “string:”)

НО!

Команда из п. 2 сработает некорректно, если в текущей строке изначально будет расположена подстрока “SeqStart”. Потому, перед выполнением п. 1, 2 необходимо заменить “SeqStart”, если такая подстрока имеется, на другую подстроку:

s/SeqStart/NotSeq/

Данный метод гарантирует сравнение текста после первого вхождения подстроки “string:”.

- В случае, если ни один из файлов не содержит подстроки “string”, в терминал выводится соответствующее сообщение.

Демонстрация тестирования:

В левой колонке приведено содержимое файла, справа - символы, отправленные в буфер для последующего сравнения.

Содержимое файла	Выделенная последовательность
This is a testring:t string:txtSeqStart string string file string SeqStart for comparing string	SeqStartt string:txtNotSeq string string file string SeqStart for comparing string
;This is a testring:t string:txtSeqStart string string file string SeqStart for	SeqStartt string:txtNotSeq string string file string SeqStart for comparing string

comparing string	

Результат работы test2.sh: 1 (Символьные последовательности различаются количеством пробельных символов)

Заметим, что скрипт:

- Корректно обработал первое вхождение подстроки “string:”.
- Сохранил исходное количество пробелов в каждой строке.

2.3. Третий вариант сравнения (По float в стандартной форме).

Листинг скрипта comparator3.sh

```
#!/bin/bash
regex="^[+-]?([0-9]+)(\.([0-9]+))?$"
file_1=$1
```

```
file_2=$2
touch /tmp/buffer_1
touch /tmp/buffer_2

if [ -z "$file_1" -o -z "$file2" ]; then
	echo "Некорректная строка запуска (При запуске укажите оба файла)."
else
	while IFS= read -r line
	do
		for word in $line; do
			if [[ $word =~ $regex ]]; then
				echo $word >> /tmp/buffer_1
			fi
		done
	done < "$file_1"

	while IFS= read -r line
	do
		for word in $line; do
			if [[ $word =~ $regex ]]; then
				echo $word >> /tmp/buffer_2
			fi
		done
	done < "$file_2"

	if cmp -s /tmp/buffer_1 /tmp/buffer_2; then
		echo 0
	else
		echo 1
fi
```

Описание скрипта comparator3.sh

- Алгоритм полностью повторяет вариант сравнения по целым числам. Различие состоит только в регулярном выражении.
- Регулярное выражение по сравнению с п.1 расширено:
  Повторяет функционал регулярного выражения п.1 + допускает наличие дробной части (Опционально, так как INT - это подкласс FLOAT)

Демонстрация тестирования:

Содержимое файла	Выделенная для сравнения последовательность
-0.9This is t0.1he -0.1 0.2 3 4 1 the11 test for testing my 3rd script 7.77654321	-0.1 0.2 3 4 1 7.77654321
Some random words to test how 0-0.7 0.4 0000.11 -9.9. my script work	0.4 0000.11

Результат работы test3.sh: 1 (Последовательности различаются)

Заметим, что скрипт:

- Успешно отбирает и добавляет в буферный файл только те ЧПТ, что отделены от остальных символов пробельными символами.
- Успешно обрабатывает нужные ЧПТ вне зависимости от их положения в строке (Сбоку, в центре).
- Учитывает знак ЧПТ.
- Включает числа типа INT в последовательность.

Последовательности ЧПТ отобраны верно. Фактический код возврата совпал с предполагаемым. Скрипт работает успешно.

2.4. Четвертый вариант сравнения (По float в стандартной и экспоненциальной форме).

Листинг скрипта comparator4.sh

```
#!/bin/bash
regex="^[-]?[0-9]+[.]?[0-9]+([e][+-]?[0-9]+)?$"
file_1=$1
file_2=$2
touch /tmp/buffer_1
touch /tmp/buffer_2

if [ -z "$file_1" -o -z "$file2" ]; then
      echo "Некорректная строка запуска (При запуске укажите оба файла)."
else
      while IFS= read -r line
```

```
    do
        for word in $line; do
            if [[ $word =~ $regex ]]; then
                echo $word >> /tmp/buffer_1
            fi
        done
    done < "$file_1"

    while IFS= read -r line
    do
        for word in $line; do
            if [[ $word =~ $regex ]]; then
                echo $word >> /tmp/buffer_2
            fi
        done
    done < "$file_2"

    if cmp -s /tmp/buffer_1 /tmp/buffer_2; then
        echo 0
    else
        echo 1
    fi

    rm /tmp/buffer_1 /tmp/buffer_2
fi
```

Описание скрипта comparator4.sh

- Функционал скрипта повторяет comparator4.sh, однако регулярное выражение расширено для учета ЧПТ в том числе и в экспоненциальной форме.
- Шаблон предполагает запись экспоненты в нижнем регистре, а также учитывает потенциальное наличие знака “+” или “-” после экспоненты для учета знака степени.
- Шаблон также поддерживает числа типа INT

Демонстрация тестирования:

Содержимое файла	Выделенная последовательность
-0.3Good 00.1 moring! .1 Th9.1is is 10,1 the last 1.0e-10 test in my work. 22.22e10 65 Hope, 12.3 100e10 that my code is 1.1e10 56.4e1 not too messy. I tried to structure everything as good as I could. Wish a happy New Year and more hard-working students.	00.1 1.0e-10 22.22e10 65 12.3 100e10 1.1e10 56.4e1
This is the 2nd part of the test. It's really doesn't matter, what I 00.1 will write there. What's really matter, is the floating point sequence. All previous 1.0e-10 test files were different in addition to the	00.1 1.0e-10 22.22e10 65 12.3

needed sequences. So return code was a 22.22e10 lw 65 ays 1. But now I'll put the same sequence 12.3 100e10 in this 1.1e10 file to show that my script can also return 0. 56.4e1	100e10 1.1e10 56.4e1

Результат работы test4.sh: 0

3. Заключение

В результате выполнения задания были разработаны и реализованы четыре различных сценария командной оболочки BASH, осуществляющих сравнение двух текстовых файлов.

Каждый сценарий (скрипт) был протестирован на различных, потенциально-опасных с точки зрения корректности работы программы, случаях. Программы прошли тесты.

Задание выполнено.

4. Список литературы

1. Теоретический материал курса в Moodle.
2. Bash text processing. (https://tldp.org/LDP/abs/html/textproc.html)
3. Text Processing - Linux command line (https://learnbyexample.gitbooks.io/linux-command-line/content/Text_Processing.html)
4. Bash scripting cheat-sheet. (https://devhints.io/bash)