Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЕТ

к лабораторной работе №2

на тему

**ОБРАБОТКА ТЕКСТОВОЙ ИНФОРМАЦИИ.**

**РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ**

Студент Д. С. Кончик

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 6](#_Toc146752070)

[Заключение 7](#_Toc146752071)

[Список использованных источников 8](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 9](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Цели лабораторной работы:

1. Изучение методов и средств обработки текстовой информации, включая регулярные выражения, и использующих их утилит.
2. Выполнение задания – написать скрипт для оболочки *shell*, который обеспечит «пред-сортировку» файлов в текущей директории. Скрипт добавляет префиксы к именам файлов, представляющие собой порядковый номер, и затем сортирует файлы в соответствии с заданными параметрами: прямым или обратным порядком по дате/времени, размеру, алфавиту или случайному порядку. При этом скрипт корректно убирает предыдущие префиксы, если они были добавлены ранее.
3. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Одним из принципов *Unix*-систем является широкое использование текстовых данных: конфигурационные файлы, входные и выходные данные программ в *\*nix* часто организованы в виде обычного текста. Регулярные выражения – это мощный инструмент для манипуляции текстовой информацией.

Регулярные выражения – это специальным образом записанные строки, используемые для поиска символьных шаблонов в тексте. Многие утилиты для работы с текстом в *Linux* и языки программирования включают в себя механизм регулярных выражений. Здесь возникают проблемы: разные программы и языки оперируют различными диалектами регулярных выражений.

Программа *grep* – это основной инструмент для работы с регулярными выражениями. *Grep* анализирует данные со стандартного ввода, ищет совпадения с указанным шаблоном и выводит все подходящие строки. Обычно *grep* предустановлен в большинстве дистрибутивов. *Grep* имеет следующий синтаксис: *grep [параметры] регулярное\_выражение [файл…]*. Самый простой случай использования *grep* – поиск строк, содержащих фиксированную подстроку. У *grep* имеются множество параметров, подробно с ними ознакомиться можно в документации.

Ранее упоминалось, что существует множество диалектов регулярных выражений. В стандарте *POSIX* рассматриваются два вида реализаций. Первый – *Basic Regular Expressions (BRE).* Ему соответствуют практически все *POSIX*-совместимые программы. Второй – *Extended Regular Expression (ERE).* Этот вид позволяет создавать более сложные регулярные выражения, но поддерживается не всеми утилитами.

К примеру, выражение «*zip*» обозначает строку, соответствующую следующим критериям: в строке не меньше трех символов; в строке присутствуют символы *«z», «i», «p»,* причем именно в таком порядке; между ними нет других символов. Символы, соответствующие сами себе (как *«z», «i», «p»*) называются литералами. Кроме того, существуют другая категория символов, называемая метасимволами. Они применяются для составления различных критериев поиска. К метасимволам в *BRE* относятся: *^ $ . [ ] \* \ -*. Чтобы использовать метасимвол в качестве литерала, его нужно экранировать с помощью обратного слэша *(\)*. Метасимвол «точка» *(.)* соответствует любому символу в данной позиции.

Символ «карет» *(^)* и «доллар» *($)* в регулярных выражениях играют роль якорей. Это означает, что в их присутствии совпадение с шаблоном возможно, только если оно будет найдено в начале строки *(^)* или в ее конце *($)*. Регулярное выражение *^$* будет соответствовать пустым строкам.

Кроме описания совпадения с любым символом в заданной позиции *(.)* в регулярных выражениях имеется возможность описать символ из определенного множества. Делается это с помощью квадратных скобок. С помощью дефиса *(-)* можно определять диапазоны символов. Так можно выразить любой диапазон символов и даже нескольких таких диапазонов [1].

Потоковый текстовый редактор *«sed»* встроен в *bash* *Linux Ubuntu*. Он использует построчное чтение, а также позволяет выполнить фильтрацию и преобразование текста. Для работы с потоковым текстовым редактором sed используется синтаксис: *sed [options] instructions [file\_name]* (где «*options»* – ключи-опции для указания метода обработки текста, *«instructions»* – команда, совершаемая над найденным фрагментом текста, *«file\_name»* – имя файла, над которым совершаются действия). Для вывода всех опций потокового текстового редактора нужно воспользоваться командой: *sed --help* [2].

*AWK* – это скриптовый язык, который полезен при работе в командной строке и широко применяется для обработки текста. При использовании *awk* можно выбирать данные – один или более отдельных фрагментов текста – на основе заданного критерия. Например, с помощью *awk* можно выполнять поиск конкретного слова или шаблона во фрагменте текста, а также выбирать определённую строку/столбец в файле.

Простейшая форма команды *awk* подразумевает описание основного действия в одинарных кавычках и фигурных скобках с указанием после него целевого файла. Выглядеть она может так: *awk '{action}' filename.txt*. Когда нужно найти текст, соответствующий конкретному шаблону, или же конкретное слово в тексте, команда принимает следующий вид:   
*awk '/regexpattern/{action}' filename.txt* [3].

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате лабораторной работы была создана консольная программа, осуществляющая «пред-сортировку» файлов, код которой представлен в приложении А. При запуске программы без передачи аргументов появляется информация о возможных параметрах командной строки (рисунок 1).

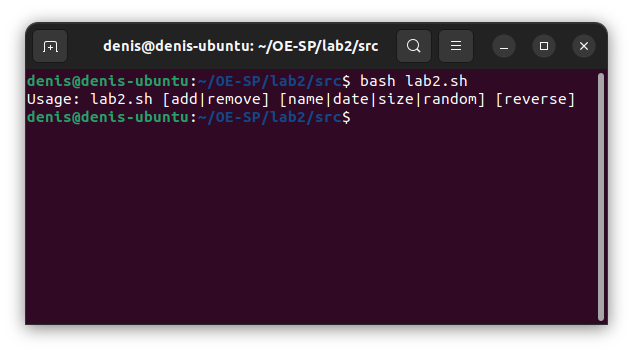


Рисунок 1 – Вывод инструкции

Скрипт добавляет префиксы к именам файлов (рисунок 2), представляющие собой порядковый номер, и в зависимости от переданных параметров затем сортирует их.

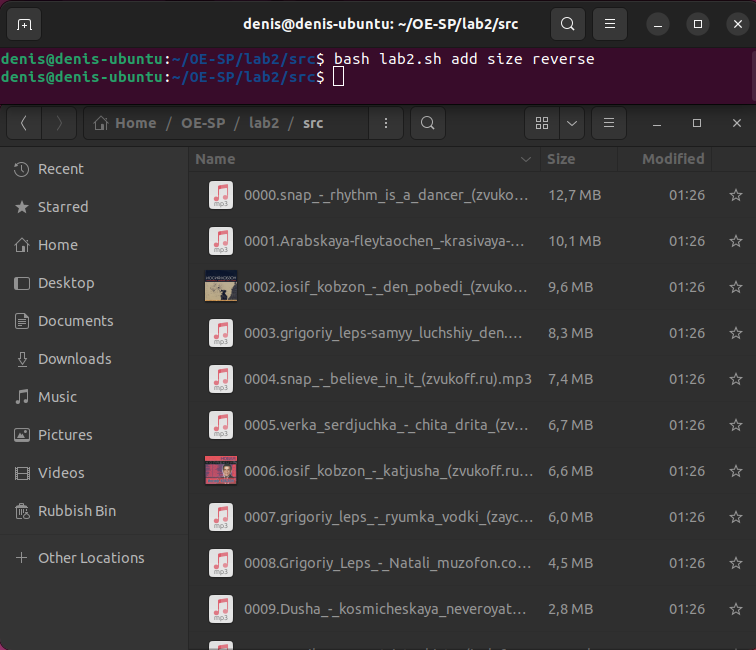


Рисунок 2 – Сортировка по убыванию размера файла

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были изучены методы и средства обработки текстовой информации, включая регулярные выражения, и использование соответствующих утилит.

Практический результат работы – создание *bash* скрипта, обеспечивающего «пред-сортировку» файлов в текущей директории. Скрипт добавляет префиксы к именам файлов, представляющие собой порядковый номер, и затем сортирует файлы в соответствии с заданными параметрами: прямым или обратным порядком по дате/времени, размеру, алфавиту или случайному порядку. При этом скрипт корректно убирает предыдущие префиксы, если они были добавлены ранее. Также есть возможность убрать ранее добавленные префиксы у всех файлов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Регулярные выражения Bash [Электронный ресурс]. – Режим доступа: https://timeweb.cloud/tutorials/linux/regulyarnye-vyrazheniya-bash-gajd.
2. Основные приёмы обработки строк в bash [Электронный ресурс]. – Режим доступа: https://eternalhost.net/base/vps-vds/bash-rabota-so-strokami
3. Команда awk – примеры использования в Linux и Unix [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/ruvds/articles/665084/.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Код программы

#!/bin/bash

add\_prefix() {

    local prefix\_length=4

    local counter=0

    local sort\_option=$1

    local reverse\_option=$2

    # Получаем отсортированный список файлов в зависимости от критерия

    case "$sort\_option" in

        "name")

            local sorted\_files=($(ls -1 | sort))

            ;;

        "date")

            local sorted\_files=($(ls -1t -r))

            ;;

        "size")

            local sorted\_files=($(ls -1S -r))

            ;;

        "random")

            local sorted\_files=($(ls -1 | shuf))

            ;;

        \*)

            echo "Unknown sort option"

            exit 1

            ;;

    esac

    if [ "$reverse\_option" == "reverse" ]; then

        sorted\_files=($(echo "${sorted\_files[@]}" | awk '{for(i=NF;i>0;i--) print $i}'))

    fi

    for file in "${sorted\_files[@]}"; do

        if [ -f "$file" ] && [ "$file" != "$script\_name" ]; then

            # Генерация нового префикса с ведущими нулями

            prefix=$(printf "%0${prefix\_length}d." "$counter")

            # Старое имя файла

            old\_filename=$(basename "$file")

            # Переименование файла с добавлением нового префикса

            mv "$file" "./$prefix${old\_filename}"

            ((counter++))

        fi

    done

}

remove\_prefix() {

    local script\_name="lab2.sh"

    for file in \*; do

        if [ -f "$file" ] && [ "$file" != "$script\_name" ]; then

            # Получение старого имени файла без префикса и точки

            old\_filename="${file##\*/}"

            new\_filename="$(echo "$old\_filename" | sed 's/^[0-9]\*\.//')"

            # Проверка, если старое имя НЕ равно новому, то производим переименование

            if [ "$old\_filename" != "$new\_filename" ]; then

                # Удаление префикса и точки из имени файла

                mv "$file" "$new\_filename"

            fi

        fi

    done

}

# Проверка наличия аргументов

if [ $# -eq 0 ]; then

    echo "Usage: $0 [add|remove] [name|date|size|random] [reverse]"

    exit 1

fi

# Определение действия

action=$1

# Определение сортировки

sort\_option=${2:-"name"}

# Определение направления сортировки

reverse\_option=""

if [ "$3" == "reverse" ]; then

    reverse\_option="reverse"

fi

script\_name="lab2.sh"

IFS=$'\n'

case "$action" in

    "add")

        remove\_prefix

        add\_prefix "$sort\_option" "$reverse\_option"

        ;;

    "remove")

        remove\_prefix

        ;;

    \*)

        echo "Unknown action"

        exit 1

        ;;

esac