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 №1

на тему

**СКРИПТЫ SHELL**

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 8](#_Toc146752071)

[Список использованных источников 9](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 10](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Цели лабораторной работы:

1. Изучение элементов и конструкций скриптов *shell*: переменных, параметров, ветвлений, циклов, вычислений, команд *shell* и вызовов внешних программ.
2. Изучение принципов интеграции *Unix*-программ скриптами *shell.*
3. Выполнение задания – написать скрипт для оболочки *shell*, который обеспечит получение заданным образом организованной выходной информации. Результаты выполнения скрипта записываются в файл для последующего анализа. Используются перенаправление ввода-вывода, внешние утилиты и фильтры, а также переменные и структуры данных *shell* для обработки данных.
4. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

*Bash* (от англ. *Bourne again shell*) – усовершенствованная и модернизированная вариация командной оболочки *Bourne shell*. Одна из наиболее популярных современных разновидностей командной оболочки *UNIX*. Особенно популярна в среде *Linux*, где она часто используется в качестве предустановленной командной оболочки.

Представляет собой командный процессор, работающий, как правило, в интерактивном режиме в текстовом окне. *Bash* также может читать команды из файла, который называется скриптом (или сценарием). Как и все   
*Unix*-оболочки, он поддерживает автодополнение имён файлов и каталогов, подстановку вывода результата команд, переменные, операторы ветвления и цикла. Ключевые слова, синтаксис и другие основные особенности языка были заимствованы из *sh*.

*Bash* в основном соответствует стандарту *POSIX*, но с рядом расширений. Название «*bash*» является акронимом от англ. *Bourne-again-shell* и представляет собой игру слов: *Bourne-shell* – одна из популярных разновидностей командной оболочки для *UNIX (sh),* автором которой является Стивен Борн (1978), усовершенствована в 1987 году Брайаном Фоксом. Фамилия *Bourne* перекликается с английским словом born, означающим «родившийся», отсюда: рождённая-вновь-командная оболочка [1].

Оболочка принимает команды, которые пользователь ввел в командную строку, и интерпретирует их, то есть переводит в машинный код. Операционная система получает код в качестве инструкций и выполняет их.

Синтаксис команды выглядит так: *<команда> [флаги][аргументы]*. Флаги – это короткие входные параметры после команды, которые обычно выглядят как дефис и буква. Это нужно, чтобы команда выполнялась определенным образом, например по-особому выводила результат. Для большинства команд существуют десятки флагов. Аргументы – это данные, которые передаются команде. Например, чтобы создать файл, нужно указать его имя – оно будет аргументом. Флаги и аргументы нужны не всегда и не для всех команд.

Если пользователю нужно выполнить много шаблонных действий по очереди и этот алгоритм будет использоваться не один раз, можно создать скрипты. Команды записываются в файл и выполняются каждый раз, когда он исполняется. С помощью скриптов можно автоматизировать работу с системой и упростить процессы. Чтобы запустить скрипт из консоли, нужно ввести команду: *bash <имя скрипта>* или *sh <имя скрипта>* [2].

*Bash*-скрипты могут быть написаны в любом текстовом редакторе и должны иметь права на выполнение. В начале каждого скрипта должна быть строка, называемая шебанг (*shebang*), которая сообщает операционной системе, какой интерпретатор использовать для выполнения скрипта.   
Шебанг должен начинаться с символа решетки и следующим за ним символ восклицания, после которых указывается путь к интерпретатору. Для использования интерпретатора *Bash*, шебанг будет выглядеть следующим образом: *#!/bin/bash*.

Также во время написания скрипта можно оставлять комментарии, которые начинаются с символа решетки и продолжаются до конца строки. Комментарии не будут выполнены интерпретатором и используются для описания функциональности скрипта.

Параметры командной строки позволяют передавать аргументы в скрипты *Linux* при его запуске. Параметры командной строки могут быть доступны в скрипте как *$1, $2, $3* и т.д., где *$1* – это первый параметр,   
*$2* – второй параметр и т.д.

Переменные в *Bash* используются для хранения данных, таких как строки и числа. Они могут быть определены явно, путем присвоения значения, или неявно, путем автоматического присвоения значения при выполнении определенных операций. Для создания переменной в *Bash* необходимо присвоить ей значение, используя знак равенства.

Значение переменной можно получить, указав ее имя после команды *echo* и знака *$*. Также возможно присвоить значение переменной через ввод с клавиатуры с помощью команды *read*.

Условные операторы – это конструкции, которые позволяют выполнить определенный набор действий в зависимости от истинности или ложности какого-то условия. В *Bash*-скриптах условия записываются в скобках и передаются в команду *if*.

Циклы в *Bash* используются для выполнения повторяющихся действий. Существуют два типа циклов: цикл *for* и цикл *while*. Цикл *for* используется для выполнения команд для каждого элемента из списка. Цикл *while* используется для выполнения команд до тех пор, пока условие не станет ложным.

Функции в *Bash* используются для группировки команд в логически связанные блоки. Функции могут быть вызваны из скрипта с помощью имени функции. Название функции должно начинаться с буквы или символа подчеркивания и может содержать только буквы, цифры и символы подчеркивания [3].

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате лабораторной работы была создана консольная игра «Пятнашки», код которой представлен в приложении А.

Суть игры заключается в перемещении числовых плиток по игровому полю с целью упорядочивания их по возрастанию. Игровое поле представляет собой квадратную сетку 4х4. Цель игры – добиться упорядоченного расположения чисел от 1 до 15, перемещая плитки по полю, используя минимальное количество ходов.

При запуске программы в нее можно передать сид игрового   
поля – начальное значение, которое используется для инициализации генератора случайных чисел. Затем инициализированный генератор случайных чисел используется для создания игровой карты. Передача сида осуществляется двумя способами:

* через аргумент командной строки;
* через файл *seed.txt* (обработка файла происходит только в том случае, если аргумент не передан в командную строку).

Если сид игрового поля явно не задается, то генератор случайных чисел инициализируется автоматически.

На рисунке 1 приведен пример игрового поля, полученного путем передачи через аргументы командной строки значения сида «15350061».

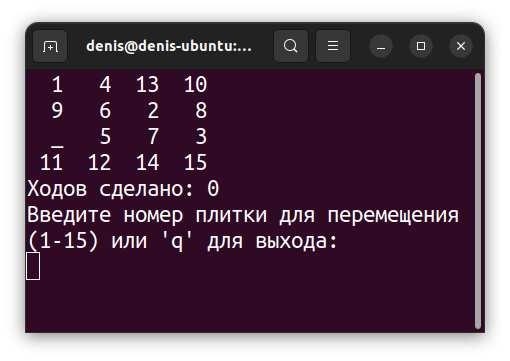


Рисунок 1 – Пример игрового поля

Для взаимодействия с программой необходимо вводить номер плитки, которую нужно передвинуть в соседнюю пустую ячейку, либо символ «q» для выхода. Если ввести номер плитки, возле которой нет свободного места, то ничего не произойдет.

При каждом перемещении увеличивается число сделанных ходов с соответствующим выводом. В случае ввода несуществующего номера плитки программа информирует пользователя об этом (рисунок 2).

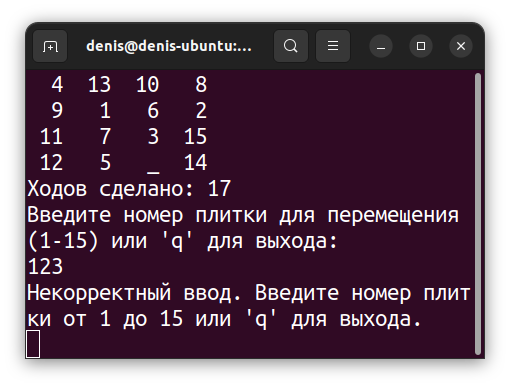


Рисунок 2 – Обработка некорректного ввода номера плитки

Также вся информация о происходящем в игре (начало, конец, исходное состояние поля, перемещения плиток, значение сида) логируется в файл *game\_log.txt* (рисунок 3).

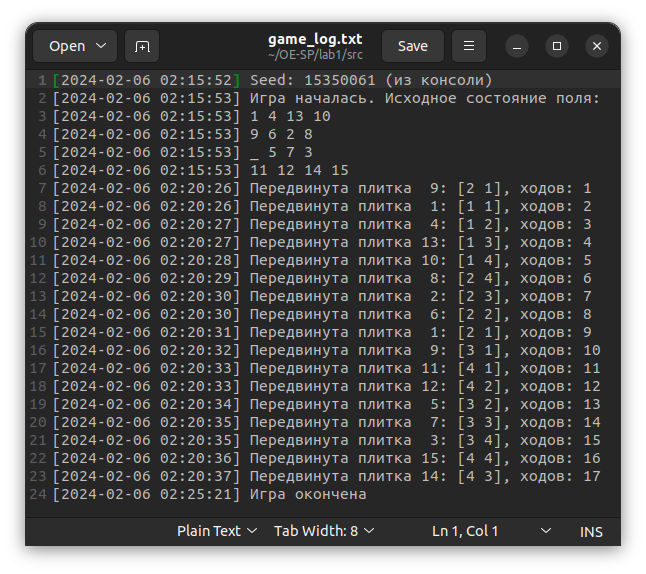


Рисунок 3 – Файл с логом игры

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были приобретены теоретические знания и практические навыки по работе со скриптами *shell*: были изучены основные элементы и конструкции, такие как переменные, параметры, ветвления, циклы, вычисления, команды.

Практический результат работы – создание консольной игры «Пятнашки» с использованием оболочки *shell*. Игра предоставляет пользователю возможность перемещать числовые плитки по игровому полю 4x4 с целью упорядочивания их по возрастанию. Весь процесс игры логируется в файл, включая начало и конец игры, исходное состояние поля, перемещения плиток и значение сида.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Bash [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Bash.
2. BASH (Bourne-Again Shell) [Электронный ресурс]. – Режим доступа: https://blog.skillfactory.ru/glossary/bash/.
3. Инструкция по Bash [Электронный ресурс]. – Режим доступа: timeweb.cloud/tutorials/linux/instrukciya-po-napisaniyu-skriptov-v-linux-bash.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Код программы

#!/bin/bash

# Функция для отображения текущего состояния игрового поля

function display\_board {

    clear

    for ((i = 0; i < 4; i++))

    do

        for ((j = 0; j < 4; j++))

        do

            printf "%3s " "${board[i\*4 + j]}"

        done

        echo

    done

}

# Функция для перемешивания элементов на игровом поле

function shuffle\_board {

    for ((i = 0; i < 1000; i++))

    do

        random=$((RANDOM % 15))

        move\_tile $((random / 4 + 1)) $((random % 4 + 1))

    done

    moves=0

}

# Функция для проверки, является ли заданная позиция корректной

function is\_valid\_position {

    local row=$1

    local col=$2

    if ((row >= 1 && row <= 4 && col >= 1 && col <= 4))

    then

        return 0

    else

        return 1

    fi

}

# Функция для обмена пустой ячейки с соседней плиткой

function move\_tile {

    local row=$1

    local col=$2

    if is\_valid\_position $row $col

    then

        if [ ${board[(row-1)\*4 + col-1]} == "\_" ]

        then

            return

        fi

        # Проверка возможности перемещения плитки в пустую ячейку

        if is\_valid\_position $((row - 1)) $col && [ ${board[((row-2)\*4)+col-1]} == "\_" ] # вверх

        then

            swap\_tiles $row $col $((row-1)) $col

            ((moves++))

        elif is\_valid\_position $((row + 1)) $col && [ ${board[(row\*4)+col-1]} == "\_" ] # вниз

        then

            swap\_tiles $row $col $((row+1)) $col

            ((moves++))

        elif is\_valid\_position $row $((col - 1)) && [ ${board[(row-1)\*4+col-2]} == "\_" ] # влево

        then

            swap\_tiles $row $col $row $((col-1))

            ((moves++))

        elif is\_valid\_position $row $((col + 1)) && [ ${board[(row-1)\*4+col]} == "\_" ] # вправо

        then

            swap\_tiles $row $col $row $((col+1))

            ((moves++))

        fi

    fi

}

# Функция для обмена плиток между двумя ячейками

function swap\_tiles {

    local row1=$1

    local col1=$2

    local row2=$3

    local col2=$4

    local temp=${board[(row1-1)\*4 + col1-1]}

    board[(row1-1)\*4 + col1-1]=${board[(row2-1)\*4 + col2-1]}

    board[(row2-1)\*4 + col2-1]=$temp

}

# Функция для нахождения координат ячейки с заданным номером

function find\_tile\_coordinates {

    local target=$1

    for ((row = 0; row < 4; row++))

    do

        for ((col = 0; col < 4; col++))

        do

            if [[ ${board[row\*4 + col]} == "$target" ]]

            then

                echo "$((row+1)) $((col+1))"

                return

            fi

        done

    done

}

# Функция проверка победы

function check\_win {

    local win\_board=($(seq 1 15))

    win\_board+=("\_")

    for ((i = 0; i < 16; i++))

    do

        if [ "${board[i]}" != "${win\_board[i]}" ]

        then

            return 1  # Игрок не выиграл

        fi

    done

    echo "Поздравляем! Вы выиграли!"

    echo "Ходов сделано: $moves"

    log\_event "Победа"

    exit 0  # Игрок выиграл

}

# Функция для логирования событий

function log\_event {

    tag=$(echo "[DATE\_PLACEHOLDER]" | sed "s/DATE\_PLACEHOLDER/$(date +'%Y-%m-%d %H:%M:%S')/")

    echo "$tag $1" >> "$log\_file"

}

# Глобальные переменные

board=($(seq 1 15))

board+=("\_")

#board=($(seq 1 14))

#board+=("\_")

#board+=("15")

moves=0

log\_file="game\_log.txt"

function main {

    # Перемешивание плиток

    shuffle\_board

    log\_event "Игра началась. Исходное состояние поля:"

    for ((i = 0; i < ${#board[@]}; i+=4)); do

        log\_event "${board[i]} ${board[i+1]} ${board[i+2]} ${board[i+3]}"

    done

    # Основной игровой цикл

    while true

    do

        display\_board

        check\_win

        echo "Ходов сделано: $moves"

        echo "Введите номер плитки для перемещения (1-15) или 'q' для выхода:"

        read input

        if [ "$input" == "q" ]

        then

            echo "Игра завершена. До свидания!"

            break

        fi

        if ((input >= 1 && input <= 15))

        then

            local coordinates=$(find\_tile\_coordinates $input)

            if [ -n "$coordinates" ]

            then

                local row=${coordinates[0]}

                local col=${coordinates[1]}

                move\_tile $row $col

                log\_event "$(printf "Передвинута плитка %+2s: [%s%s], ходов: %d" "$input" "$row" "$col" "$moves")"

            else

                echo "Некорректный ввод. Плитка с номером $input не найдена."

                sleep 2

            fi

        else

            echo "Некорректный ввод. Введите номер плитки от 1 до 15 или 'q' для выхода."

            sleep 2

        fi

    done

    log\_event "Игра окончена"

}

# Инициализация лога

echo -n > "$log\_file"

if [ -n "$1" ];

then

    RANDOM=$1

    log\_event "Seed: $1 (из консоли)"

elif [ -s "seed.txt" ]

then

    SEED=$(< "seed.txt")

    RANDOM=$SEED

    log\_event "Seed: $SEED (из файла seed.txt)"

else

    log\_event "Seed: не задан"

fi

# Перенаправление ошибок

exec 2>>$log\_file

main