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

ОТЧЁТ

к лабораторной работе №1

на тему

**СКРИПТЫ SHELL(УГЛУБЛЕННАЯ ВЕРСИЯ)**

Выполнил: студент гр.253504 Сапроненко В.В.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2025

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc183307053)

[2 Краткие теоритические сведения 4](#_Toc183307054)

[3 Описание функций программы 5](#_Toc183307055)

[Заключение 7](#_Toc183307056)

[Список использованных источников 8](#_Toc183307057)

[Приложение А (обязательное) исходный код программы 9](#_Toc183307058)

# **1 ФОРМУЛИРОВКА ЗАДАЧИ**

В данной лабораторной работе необходимо изучить и освоить основные элементы и конструкции, используемые в shell-скриптах, а также получить практические навыки работы с командным интерпретатором Bash. Это включает в себя изучение переменных, параметров, операторов ветвления, циклов, выполнения арифметических вычислений и использования команд оболочки Linux.

Целью работы является не только написание конкретного скрипта, но и понимание того, как различные команды и инструменты Unix могут быть интегрированы в единую программу. Bash-скрипты являются мощным инструментом автоматизации, позволяющим создавать гибкие и эффективные решения для повседневных задач.

В рамках данной лабораторной работы предлагается реализовать консольную версию игры "Крестики-нолики", предназначенную для двух игроков. Игроки поочередно делают ходы, вводя номер клетки, в которую хотят поставить свой символ.

В ходе выполнения работы необходимо разобраться с обработкой пользовательского ввода, управлением состоянием игры, а также научиться организовывать логику проверки победителя и ничьей. Реализация данной задачи позволит закрепить знания по использованию базовых элементов программирования в Bash и даст представление о принципах написания скриптов.

# **2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ**

Shell (интерпретатор командной строки) [1] является важной частью операционных систем семейства Unix/Linux и используется для взаимодействия пользователя с системой. Он позволяет выполнять команды, управлять файлами и процессами, а также создавать скрипты для автоматизации задач. Один из наиболее популярных интерпретаторов — Bash (Bourne Again Shell), который представляет собой усовершенствованную версию оригинального Bourne Shell и включает в себя поддержку переменных, условных операторов, циклов, работы с массивами, строковыми операциями и взаимодействием с внешними программами.

Bash широко применяется в администрировании серверов, автоматизации рутинных задач, анализе данных и создании различных утилит. Важной особенностью Bash является возможность интеграции с утилитами Unix, такими как sed (потоковый редактор), awk (инструмент обработки текстовых данных), wget (утилита для загрузки файлов из сети), а также различными фильтрами и командами для работы с текстом. Эти инструменты позволяют создавать мощные и гибкие скрипты для решения широкого спектра задач.

В основе Bash лежит работа с переменными, которые могут хранить строки, числа или массивы значений. Объявление переменной происходит без указания типа, а доступ к значению осуществляется через знак $. Например, переменная может быть задана командой name="Привет" и выведена с помощью echo $name. Для организации логики выполнения кода используются условные операторы (if, elif, else) [2], позволяющие проверять различные условия и принимать решения на их основе. Например, в игре реализована проверка допустимости хода игрока и проверка условий победы с помощью конструкции if [[ условие ]]; then. Также активно применяются циклы (while, for), которые позволяют повторять определенные действия.

Пользовательский ввод осуществляется с помощью команды read, позволяющей программе получать данные от пользователя. Например, в коде используется read move, что ожидает ввода номера клетки и сохраняет его в переменную move. После этого введенное значение корректируется (let "move=$move-1") для соответствия индексам массива, начинающимся с 0.

В скрипте реализована функция valid\_move, проверяющая, свободна ли выбранная клетка, и функция check\_win, определяющая победителя. Игра продолжается в цикле while true, где игроки поочередно вводят свои ходы. После каждого хода проверяется, завершилась ли игра победой или ничьей.

Таким образом, написание скрипта для игры «Крестики-нолики» продемонстрировало применение ключевых элементов программирования на Bash: массивов, функций, условных операторов, циклов и пользовательского ввода.

# **3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

Программа реализует консольную версию игры «Крестики-нолики» с использованием языка Bash. В ее основе лежит несколько функций, каждая из которых выполняет определенную задачу, обеспечивая корректное взаимодействие с пользователем, обновление игрового поля и проверку условий победы. Функции:

1 Функция инициализации игрового поля: При запуске программы создается массив board, представляющий игровое поле. Каждая ячейка инициализируется пробелом (" "), что означает, что клетка свободна для хода. Эта функция отвечает за подготовку игры перед началом раундов.

2 Функция отображения игрового поля: Функция print\_board отвечает за визуализацию текущего состояния игрового поля. Она использует команду echo для форматированного вывода строк в консоль, отображая текущее расположение символов «X» и «O» на поле.

3 Функция обработки ввода: Программа запрашивает у игрока номер клетки с помощью команды read move, после чего проверяет введенные данные. Значение уменьшается на 1 (let "move=$move-1") для соответствия индексам массива. Если число выходит за пределы 1-9 или клетка уже занята, программа выводит сообщение об ошибке и предлагает повторить ввод.

4 Функция обновления игрового поля: После успешного хода введенное значение заменяется на соответствующий символ (X или O). Обновленное поле сохраняется в массиве board, после чего вызывается print\_board для отображения изменений.

5 Функция проверки победителя: Функция check\_win проверяет победные комбинации (строки, столбцы и диагонали). Если три клетки содержат одинаковый символ (X или O), программа объявляет победителя и завершает выполнение. В противном случае игра продолжается до момента заполнения всех клеток, после чего объявляется ничья.

6 Функция управления игровым процессом: Основной игровой цикл реализован с помощью while true, который выполняется, пока не будет выявлен победитель или не произойдет ничья. Внутри цикла вызываются все ключевые функции: обработки ввода, обновления поля, отображения результатов и проверки победителя. Очередность ходов меняется, переключая turn между X и O. На рисунке 3.1 продемонстрирована работа программного продукта.

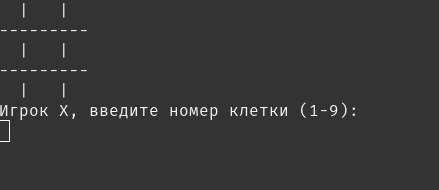


Рисунок 3.1 – Результат работы программы.

Таким образом, программа содержит ряд функций, каждая из которых выполняет свою роль, обеспечивая стабильную работу игрового процесса. Они демонстрируют принципы программирования на Bash, такие как работа с переменными и массивами, обработка ввода и управление логикой выполнения.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы была разработана консольная версия игры "Крестики-нолики" с использованием скриптов shell. Программа успешно реализует основные функции игры, включая отображение игрового поля, обработку ходов двух игроков, проверку на победу и ничью, а также создание скриншотов текущего состояния поля.

Работа над проектом позволила закрепить знания и навыки работы с основными элементами shell-скриптов, такими как переменные, условные операторы, циклы и функции. Были изучены принципы организации логики программы, включая обработку пользовательского ввода и управление состоянием игрового поля.

Особое внимание было уделено реализации проверки на победу и ничью, что потребовало тщательной проработки алгоритмов анализа игрового поля. Дополнительно была реализована функция создания скриншотов, которая демонстрирует возможности перенаправления вывода в файл.

Программа была протестирована на корректность работы. Все функции, включая отображение поля, выполнение ходов, проверку условий завершения игры и создание скриншотов, работают в соответствии с ожиданиями. Игра предоставляет удобный интерфейс для двух игроков и может быть легко расширена для добавления новых функций, таких как игра против компьютера или улучшенное оформление.

Таким образом, поставленные задачи были успешно выполнены. Разработанная программа демонстрирует возможности использования shell-скриптов для создания простых, но функциональных приложений. Данная работа также подчеркивает важность структурированного подхода к разработке программного обеспечения, даже для небольших проектов.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Introduction to Linux Shell and Shell Scripting [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/introduction-linux-shell-shell-scripting/

[2] Bash скрипты [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/726316/

# **ПРИЛОЖЕНИЕ А (обязательное) Исходный код программного продукта**

#!/bin/bash

board=(" " " " " " " " " " " " " " " ")

function print\_board() {

echo "${board[0]} | ${board[1]} | ${board[2]}"

echo "---------"

echo "${board[3]} | ${board[4]} | ${board[5]}"

echo "---------"

echo "${board[6]} | ${board[7]} | ${board[8]}"

}

function check\_win() {

for i in 0 3 6; do

if [[ ${board[$i]} != " " && ${board[$i]} == ${board[$((i+1))]} && ${board[$i]} == ${board[$((i+2))]} ]]; then

return 0

fi

done

for i in 0 1 2; do

if [[ ${board[$i]} != " " && ${board[$i]} == ${board[$((i+3))]} && ${board[$i]} == ${board[$((i+6))]} ]]; then

return 0

fi

done

if [[ ${board[0]} != " " && ${board[0]} == ${board[4]} && ${board[0]} == ${board[8]} ]]; then

return 0

fi

if [[ ${board[2]} != " " && ${board[2]} == ${board[4]} && ${board[2]} == ${board[6]} ]]; then

return 0

fi

return 1

}

function valid\_move() {

if [[ ${board[$1]} != " " ]]; then

echo "Эта клетка уже занята, попробуйте снова."

return 1

fi

return 0

}

turn="X"

while true; do

print\_board

echo "Игрок $turn, введите номер клетки (1-9):"

read move

let "move=$move-1"

if [[ $move -ge 0 && $move -le 8 && $(valid\_move $move) -eq 0 ]]; then

board[$move]=$turn

else

continue

fi

if check\_win; then

print\_board

echo "Игрок $turn выиграл!"

break

fi

if ! [[ " ${board[@]} " =~ " " ]]; then

print\_board

echo "Ничья!"

break

fi

if [[ $turn == "X" ]]; then

turn="O"

else

turn="X"

fi

done