Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Лабораторная работа 2

По дисциплине «Операционные системы»

На тему «Знакомство с командной оболочкой операционной системы»

Выполнил:

Студент 3 курса 9 группы

Хуторцов Кирилл Владимирович

Преподаватель: Савельева М.Г.

Минск, 2025

# **Введение**

**Цель работы:** получение практических навыков в работе с командными оболочками такими как cmd, PowerShell, bash.

**Постановка задачи:**

1. Требуется создать каталог OS2, запустить prep.bat.
2. Требуется выполнить команды: cd, copy, xcopy, move, replace, ren, fc, del, dir, md, rd.
3. Требуется использовать >, >>, |, &.
4. Требуется написать 5 bat-файлов (bat\_01–bat\_05) с переменными, параметрами, математикой и логикой.
5. Требуется исследовать: названия команд, параметры, переменные окружения (PATH — ключевая).
6. Требуется написать скрипты для калькулятора и перевода температуры для PowerShell.

**Используемые инструменты**:

* Oracle VirtualBox
* Командные оболочки: cmd, PowerShell, bash

# **Постановка задачи для Windows**

## **Уровень 4-5**

1. Зашли в домашний каталог пользователя, созданного в предыдущей лабораторной работе, и создали каталог OS2.

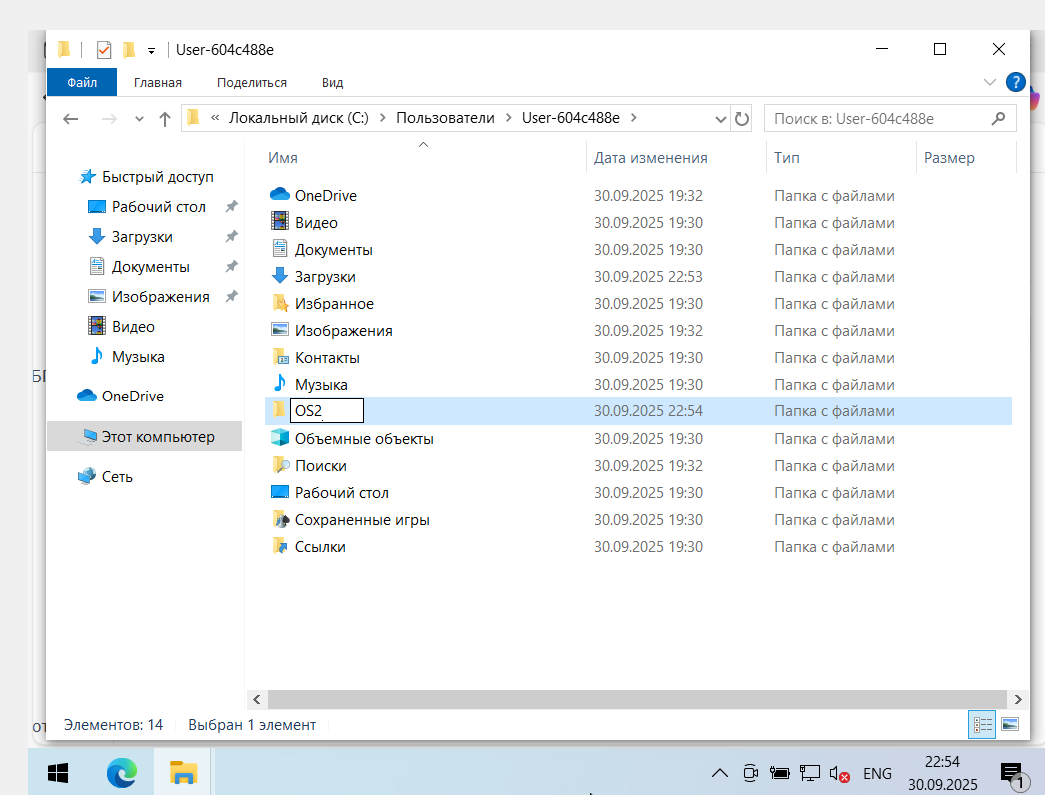


Рисунок 1.1 – Создали каталог

1. Выполнили бат-файл prep.bat, который подготовил нам среду: создал все необходимые каталоги и заполнил их файлами.

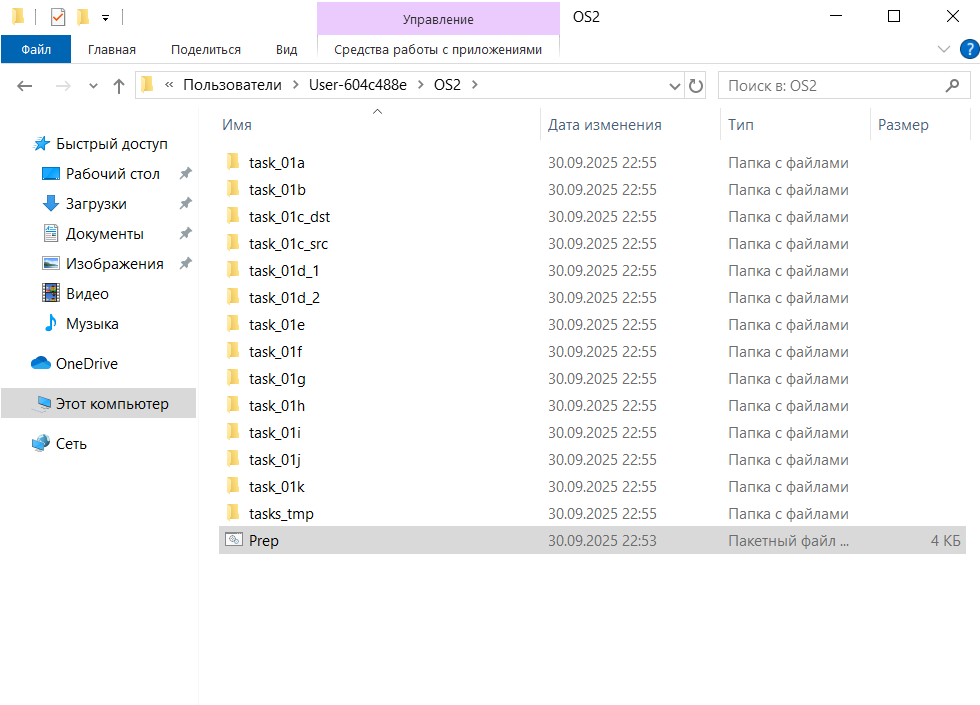


Рисунок 1.2 – Выполнение бат-файла

1. Затем вводим в адресную строку проводника команду для открытия cmd.

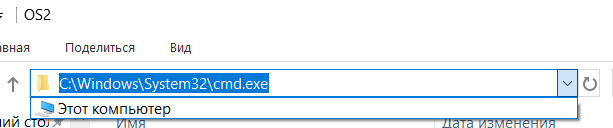


Рисунок 1.3 – Открытие cmd

1. Переходим в каталог taks\_01i/Русский с использованием абсолютного пути.

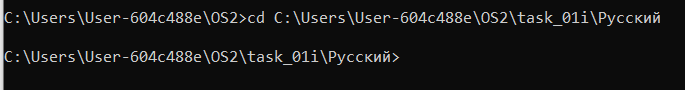


Рисунок 1.4 – Переход в каталог по абсолютному пути

Переходим в каталог taks\_01i/English с использованием относительных путей.

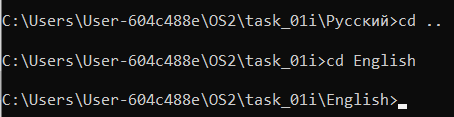


Рисунок 1.5 – Переход в каталог по относительному пути

Копируем все файлы, имеющие расширение txt из каталога tasks\_tmp в каталог task\_01a.

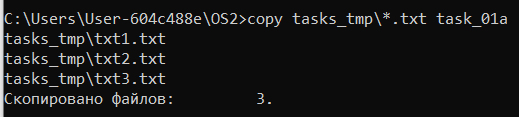


Рисунок 1.6 – Копирование файлов

Повторяем ту же операцию, но при этом команда запрашивает подтверждение на перезапись существующих файлов.

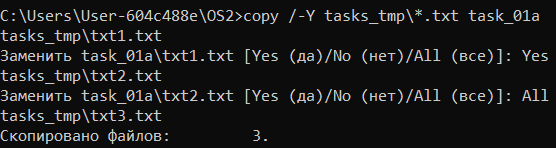


Рисунок 1.7 – Копирование файлов с запросом на подтверждение

Копируем файл из каталога task\_01a в каталог task\_01a, но с новым именем.

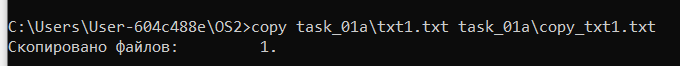


Рисунок 1.8 – Копирование файла внутри каталога под другим названием

Объединяем файлы txt1.txt и txt2.txt в файл merged.txt.

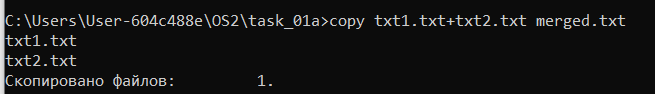


Рисунок 1.9 – Объединение файлов

Копируем текст, введенный в консоль в файл с именем console.txt.

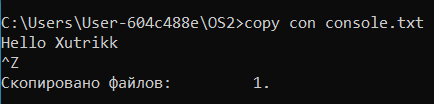


Рисунок 1.10 – Копирование текста из консоли

Добавляем в конец файла txt3 информацию, вводимую с консоли.

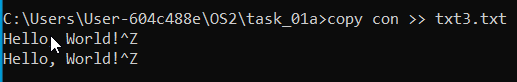


Рисунок 1.11 – Добавление информации в конец файла

Копируем все файлы, каталоги и подкаталоги, включая пустые, а также скрытые и системные из task\_01b в task\_01b\_dst\_1.

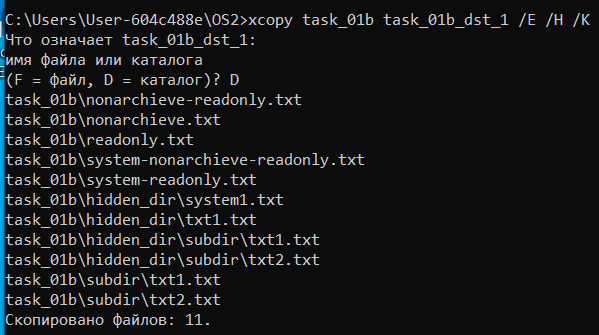


Рисунок 1.12 – Копирование всех файлов

/E — копирует все подкаталоги, включая пустые. /H — копирует скрытые и системные файлы. /K — сохраняет атрибуты файлов.

Копируем все файлы с атрибутами «архивный» и «только для чтения» с сохранением этого атрибута для файлов-результатов из task\_01b в task\_01b\_dst\_2.

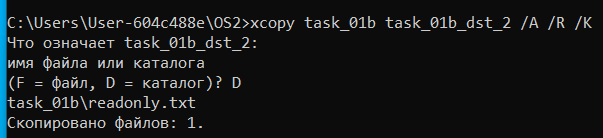


Рисунок 1.13 – Копирование архивных файлов и для чтения

/A — копирует только файлы с атрибутом "архивный". /R — копирует файлы "только для чтения". /K — сохраняет атрибуты у файлов-результатов.

Перемещаем все текстовые файлы с расширением .txt из каталога task\_01c\_src в каталог task\_01c\_dst.

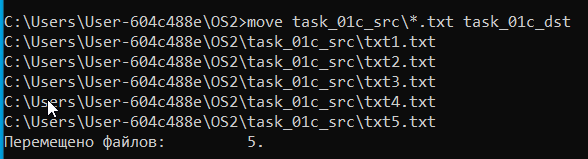


Рисунок 1.14 – Перемещение файлов

Заходим в каталог task\_01d\_2 и удаляем текст в файлах.

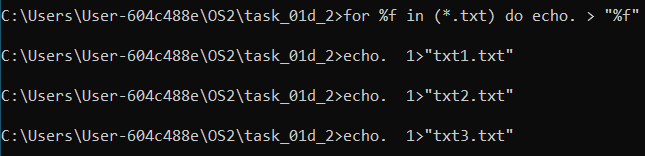


Рисунок 1.15 – Удаление текста в файлах

for %f in (\*.txt) — перебирает все .txt файлы. echo. > "%f" — перезаписывает каждый файл пустым содержимым.

Заменяем файлы в каталоге task\_01d\_2 файлами, расположенными в каталоге task\_01d\_1.

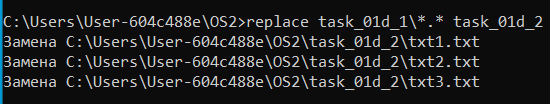


Рисунок 1.16 – Замена файлов

Изменяем расширение всех файлов в каталоге task\_01e на .bmp.



Рисунок 1.17 – Изменение расширения

Изменяем имя файла txt1.bmp на User-c121f4dd.Ren.



Рисунок 1.18 – Изменение имени файла

Сравниваем два файла в каталоге task\_01f и сохраняем результат сравнения в result.txt.



Рисунок 1.19 – Сравнение файлов

Удаляем файл txt1 из каталога task\_01g, при этом запросив подтверждение на удаление.



Рисунок 1.20 – Удаление файла с запросом на подтверждение

Удаляем все файлы с атрибутом «Системный» из каталога task\_01g.



Рисунок 1.21 – Удаление файлов с атрибутом «Системный»

for /R "task\_01g" — рекурсивно перебирает все файлы в task\_01g и подкаталогах. attrib "%f" — проверяет атрибуты. find /i "S" — ищет системный атрибут. attrib -s -h "%f" — снимает системный и скрытый. del "%f" — удаляет файл.

Удаляем все файлы из каталога task\_01g так, чтобы остались только файлы с атрибутом «Только для чтения».

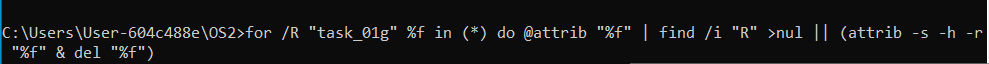


Рисунок 1.22 – Удаляем файлы

for %f in (task\_01g\\*) — перебирает все файлы в каталоге task\_01g. attrib "%f" — проверяет атрибуты файла. find /i "R" — ищет атрибут Read-only. || del "%f" — если атрибут не найден, файл удаляется.

Удаляем все оставшиеся файлы и дочерние каталоги из каталога task\_01g.

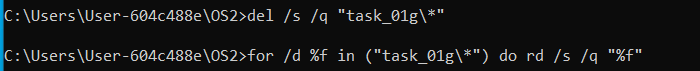


Рисунок 1.23 – Удаление оставшихся файлов

Постранично выводим содержимое каталога C:\Windows, включая вложенные каталоги и файлы.

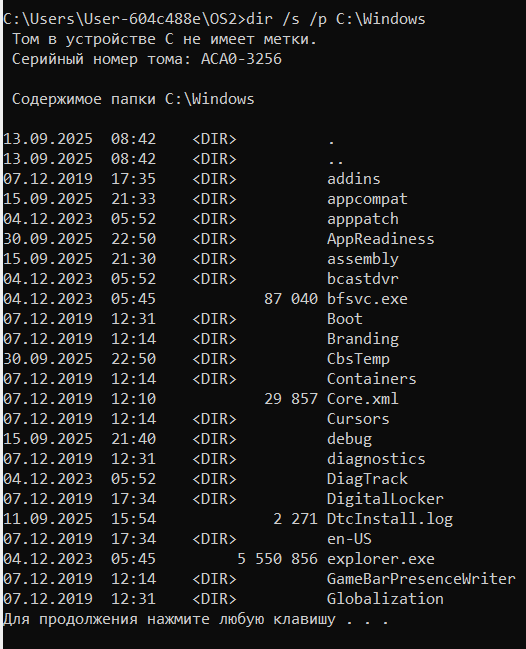


Рисунок 1.24 – Просмотр содержимого

/s — рекурсивно показывает все файлы и папки, включая вложенные. /p — выводит постранично, останавливаясь после каждой страницы и ожидая нажатия клавиши.

Постранично выводим все каталоги и файлы, включая вложенные подкаталоги и файлы каталога C:\Users\«ваш пользователь» в отсортированном по алфавиту порядке с записью результата в файл user\_files.txt в каталоге task\_01h.



Рисунок 1.25 – Вывод содержимого по алфавиту и запись в файл

Создаем каталог task\_01j в вашем каталоге OS2.



Рисунок 1.26 – Создание каталога

Создаем иерархию каталогов task\_01j\sub\dir\full\path



Рисунок 1.27 – Создание иерархии

Удаляем пустой каталог \task\_01k\empty\_subdir.



Рисунок 1.28 – Удаление пустого каталога

Удаляем не пустой каталог \task\_01k\subdir.



Рисунок 1.29 – Удаление не пустого каталога

/s — удаляет всё содержимое, включая вложенные файлы и папки. /q — тихий режим, без подтверждений.

1. Выполняем следующие задания с использованием различных команд, но с обязательным использованием таких операторов как перенаправление вывода (>, >>), конвейеризации (|) и соединения (&).

Возвращаемся в папку OS2.



Рисунок 1.30 – Возврат в папку OS2

Создаем текстовый файл со списком студентов группы и их номерами книжек, используя команду **type** иввод с консоли.

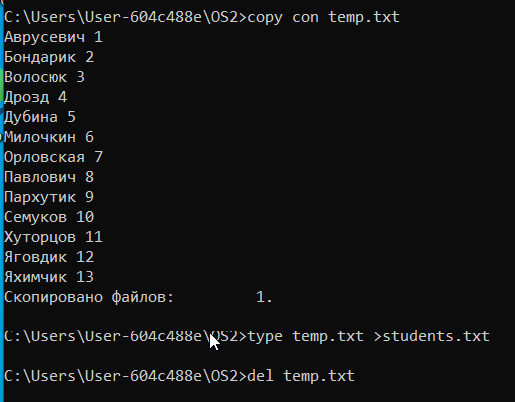


Рисунок 1.31 – Создание и заполнение файла

Выполняем поиск студента в файле по его фамилии.



Рисунок 1.32 – Поиск студента по фамилии

Добавляем ещё студентов в конец файла.



Рисунок 1.33 – Добавление студентов

Выводим список студентов в отсортированном порядке и после этого выходим из папки OS2.

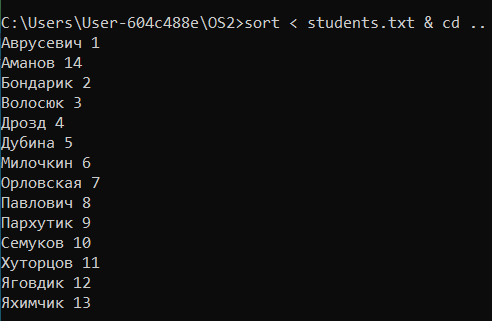


Рисунок 1.34 – Вывод списка студентов

Выводим версию BIOS через файл.



Рисунок 1.35 – Вывод версии BIOS

1. Создаем bat-файл bat\_01, который выводит следующую информацию, используя системные переменные ([ss64](https://ss64.com/nt/syntax-variables.html)): *Имя текущего пользователя, текущие дату и время, имя компьютера.*

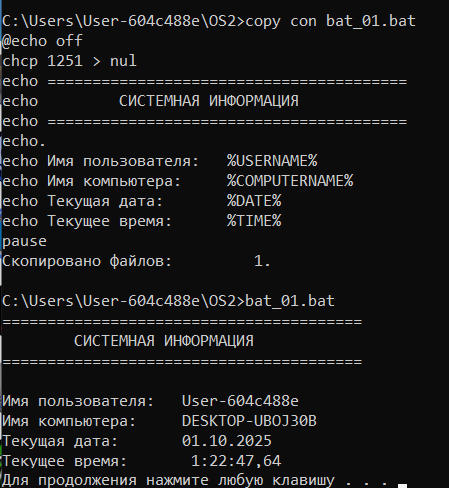


Рисунок 1.36 – Создание bat-файла bat\_01

1. Создаем bat-файл bat\_02, который выводит следующую информацию, используя расширения параметров ([ss64](https://ss64.com/nt/syntax-args.html)): *имя текущего бат-файла, время последней модификации и полный путь к текущему бат-файлу.*

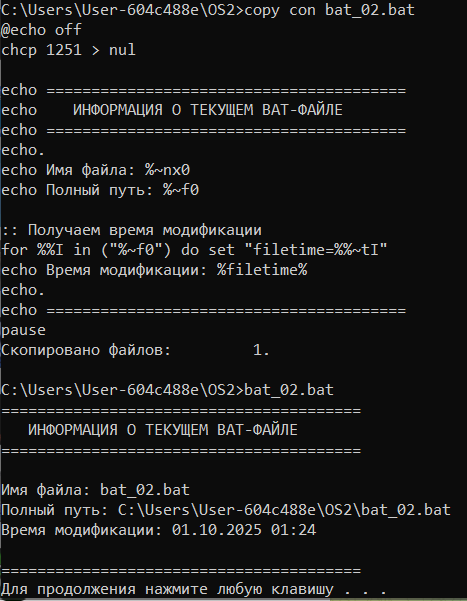


Рисунок 1.37 – Создание bat-файла bat\_02

1. Создаем bat-файл bat\_03, который выводит следующую информацию, используя параметры ([ss64](https://ss64.com/nt/syntax-args.html)): *строку со всеми переданными параметрами, а также первые 4 параметра по отдельности.*

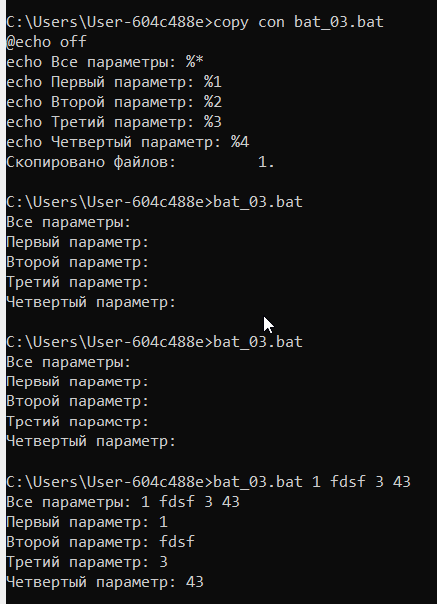


Рисунок 1.38 – Создание bat-файла bat\_03

1. Создаем bat-файл bat\_04, который выводит следующую информацию ([ss64](https://ss64.com/nt/set.html)): *результаты следующих математических выражений a-b, a+b, c/b, c\*b, (a-b)\*(b-a), где a,b,c – 1,2 и 3 параметры передаваемые бат-файлу.*

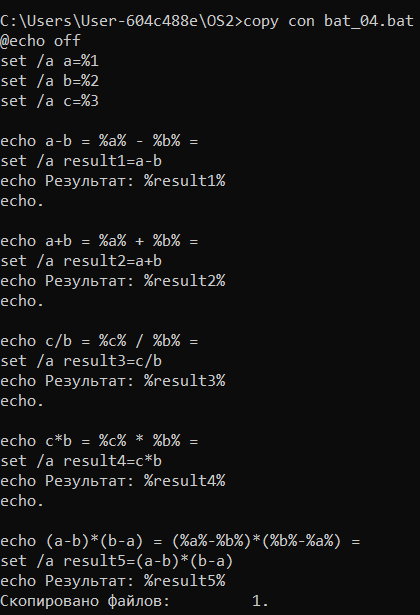


Рисунок 1.39 – Создание bat-файла bat\_04

Вызываем наш файл с параметрами и просматриваем результат.

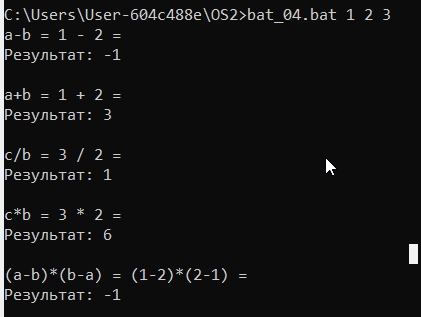


Рисунок 1.40 – Просмотр результата

1. Создаем bat-файл bat\_05, который работает следующим образом:

* вызывается в формате bat\_05 {режим работы} {имя файла};
* поддерживает режимы работы – создать, удалить (так и пишется в консоли);
* имя файла – любое;
* если не задан ни режим, ни файл, то выводится подсказка по использованию файла;
* если не задано имя файла, то выводится соответствующее сообщение;
* режим «создать» – создаёт файл, указанный вторым параметром, но если файл уже существует, то выводится соответствующее сообщение;
* режим «удалить» – удаляет файл, указанный вторым параметром, но если файл отсутствует, то выводится соответствующее сообщение.

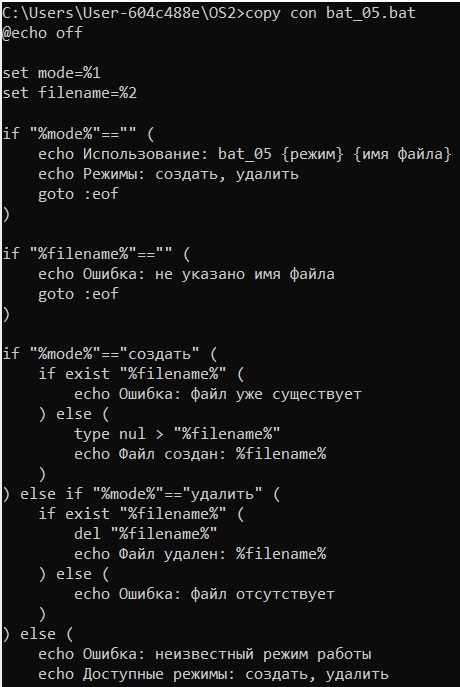


Рисунок 1.41 – Создание bat-файла bat\_05

После создания файла проверяем, как он работает.

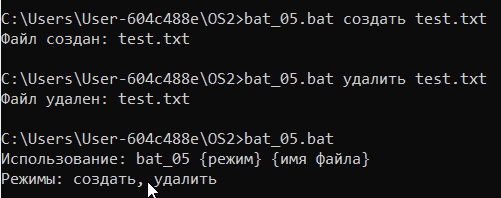


Рисунок 1.42 – Проверка использования

1. На основе рассмотренных ранее консольных команд и с использованием документации по ним выделяем правила их наименования. И тоже самое проворачиваем относительно параметров данных команд.

Правила наименования команд. Команды часто происходят от английских слов, описывающих действие, стремятся быть короткими и сохраняют совместимость с MS-DOS и ранними версиями Windows.

Таблица 1.1 – Примеры наименований команд

|  |  |  |
| --- | --- | --- |
| **Команда** | **Происхождение** | **Значение** |
| dir | directory | Показать содержимое каталога |
| cd | change directory | Перейти в другой каталог |
| md / mkdir | make directory | Создать каталог |
| rd / rmdir | remove directory | Удалить каталог |
| del | delete | Удалить файл |
| copy | copy | Копировать файл |
| type | type out | Вывести содержимое файла |
| echo | echo | Отобразить строку |
| set | set variable | Задать переменную |
| pause | pause execution | Приостановить выполнение |
| find | find string | Найти строку в файле |
| sort | sort lines | Отсортировать строки |
| systeminfo | system information | Вывести сведения о системе |

Правила наименования параметров. Параметры начинаются с /, часто являются сокращениями от английских слов, могут быть одиночными или составными.

Таблица 1.2 – Примеры наименований параметров

|  |  |  |
| --- | --- | --- |
| **Команда** | **Параметр** | **Значение** |
| dir | /s | subdirectories — показать вложенные каталоги |
| dir | /p | page — постраничный вывод |
| copy | /y | yes — не спрашивать подтверждение |
| rd | /s | subtree — удалить всё содержимое |
| rd | /q | quiet — без подтверждений |
| icacls | /grant | grant access — выдать права |
| icacls | /T | tree — рекурсивно по дереву |
| attrib | -s | system — снять системный атрибут |
| attrib | -h | hidden — снять скрытый атрибут |
| set | /a | arithmetic — режим вычислений |

1. Переменные окружения — это специальные переменные, которые хранят системную информацию и доступны всем процессам в операционной системе. Они задают контекст выполнения команд, скриптов и программ.

Переменные окружения позволяют:

* Получать пути к системным каталогам (%SystemRoot%, %TEMP%).
* Узнавать имя пользователя (%USERNAME%).
* Определять имя компьютера (%COMPUTERNAME%).
* Управлять путями к исполняемым файлам (%PATH%).
* Использовать переменные в скриптах (%1, %\*, %DATE%, %TIME%).
* Передавать параметры между процессами и скриптами.

Таблица 1.3 – Переменные окружения

|  |  |
| --- | --- |
| **Переменная** | **Назначение** |
| PATH | Список каталогов, в которых система ищет исполняемые файлы |
| PATHEXT | Расширения файлов, считающихся исполняемыми |
| COMSPEC | Путь к интерпретатору командной строки (cmd.exe) |
| TEMP / TMP | Путь к временной папке пользователя |
| USERNAME | Имя текущего пользователя |
| USERPROFILE | Путь к домашнему каталогу пользователя |
| HOMEPATH | Относительный путь к домашнему каталогу |
| HOMEDRIVE | Диск, на котором расположен домашний каталог |
| LOCALAPPDATA | Путь к папке Local для хранения локальных данных приложений |
| NUMBER\_OF\_PROCESSORS | Количество логических процессоров |
| PROCESSOR\_ARCHITECTURE | Архитектура процессора |
| PROCESSOR\_IDENTIFIER | Строка с названием процессора |
| PROCESSOR\_LEVEL | Уровень процессора |
| OS | Название операционной системы |
| LOGONSERVER | Сервер, через который выполнен вход в систему |
| COMPUTERNAME | Имя компьютера |
| USERDOMAIN | Домен пользователя |
| PROMPT | Настройка приглашения командной строки |
| ERRORLEVEL | Код возврата последней выполненной команды |
| CMDLINE | Путь к интерпретатору командной строки (cmd.exe) |

Параметр COMSPEC указывает на главную командную строку компьютера, а CMDCMDLINE указывает на командную строку, которая запущена.

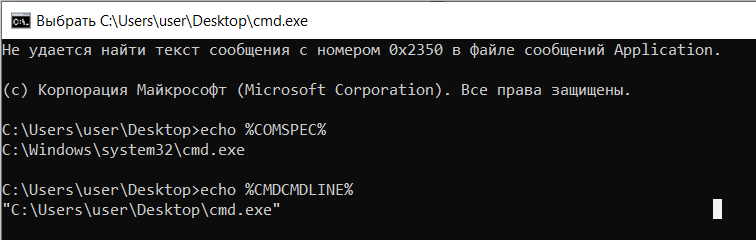


Рисунок 1.43 – Сравнение параметров

Самая важная переменная для работы в консоли – %PATH%, потому что она определяет, где Windows ищет исполняемый файл при вводе команды.

## **Уровень 7-8**

1. Необходимо написать скрипт для PowerShell, который будет представлять функционал интерактивного калькулятора. Поддерживаемые математические операции должны включать основные арифметические операции, возведение в степень, извлечение корня заданной степени, основные тригонометрические функции, десятичный и натуральный логарифмы, остаток от деления.

Открываем PowerShell и создаем при помощи команды файл.

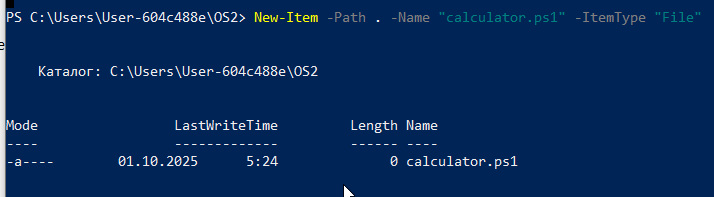


Рисунок 1.43 – Создание файла

Пишем внутри файла скрипт для калькулятора.

function Show-Menu {

Write-Host "`n=== ИНТЕРАКТИВНЫЙ КАЛЬКУЛЯТОР ===" -ForegroundColor Cyan

Write-Host "1. Сложение (+)"

Write-Host "2. Вычитание (-)"

Write-Host "3. Умножение (\*)"

Write-Host "4. Деление (/)"

Write-Host "5. Возведение в степень (^)"

Write-Host "6. Извлечение корня (v)"

Write-Host "7. Синус (sin)"

Write-Host "8. Косинус (cos)"

Write-Host "9. Тангенс (tan)"

Write-Host "10. Арксинус (asin)"

Write-Host "11. Арккосинус (acos)"

Write-Host "12. Арктангенс (atan)"

Write-Host "13. Десятичный логарифм (log10)"

Write-Host "14. Натуральный логарифм (ln)"

Write-Host "15. Остаток от деления (%)"

Write-Host "16. Выход`n"

}

function Read-Number($prompt) {

while ($true) {

try {

return [double](Read-Host $prompt)

}

catch {

Write-Host "Ошибка! Введите число." -ForegroundColor Red

}

}

}

function Calculate-Root($number, $root) {

if ($root % 2 -eq 0 -and $number -lt 0) {

throw "Ошибка: нельзя извлечь корень четной степени из отрицательного числа"

}

return [Math]::Pow($number, 1/$root)

}

# Основная программа

$angleUnit = Read-Host "Выберите единицы измерения углов (R-радианы/D-градусы)"

$angleUnit = $angleUnit.ToUpper()

while ($true) {

Show-Menu

$choice = Read-Host "Выберите операцию (1-16)"

try {

$result = $null

switch ($choice) {

"1" {

$a = Read-Number "Введите первое число"

$b = Read-Number "Введите второе число"

$result = $a + $b

}

"2" {

$a = Read-Number "Введите первое число"

$b = Read-Number "Введите второе число"

$result = $a - $b

}

"3" {

$a = Read-Number "Введите первое число"

$b = Read-Number "Введите второе число"

$result = $a \* $b

}

"4" {

$a = Read-Number "Введите первое число"

$b = Read-Number "Введите второе число"

if ($b -eq 0) { throw "Ошибка: деление на ноль!" }

$result = $a / $b

}

"5" {

$a = Read-Number "Введите основание"

$b = Read-Number "Введите показатель степени"

$result = [Math]::Pow($a, $b)

}

"6" {

$a = Read-Number "Введите число"

$b = Read-Number "Введите степень корня"

$result = Calculate-Root $a $b

}

"7" {

$x = Read-Number "Введите значение"

if ($angleUnit -eq "D") { $x = $x \* [Math]::PI / 180 }

$result = [Math]::Sin($x)

}

"8" {

$x = Read-Number "Введите значение"

if ($angleUnit -eq "D") { $x = $x \* [Math]::PI / 180 }

$result = [Math]::Cos($x)

}

"9" {

$x = Read-Number "Введите значение"

if ($angleUnit -eq "D") { $x = $x \* [Math]::PI / 180 }

$result = [Math]::Tan($x)

}

"10" {

$x = Read-Number "Введите значение"

if ($x -lt -1 -or $x -gt 1) { throw "Ошибка: арксинус определен только для [-1, 1]" }

$result = [Math]::Asin($x)

if ($angleUnit -eq "D") { $result = $result \* 180 / [Math]::PI }

}

"11" {

$x = Read-Number "Введите значение"

if ($x -lt -1 -or $x -gt 1) { throw "Ошибка: арккосинус определен только для [-1, 1]" }

$result = [Math]::Acos($x)

if ($angleUnit -eq "D") { $result = $result \* 180 / [Math]::PI }

}

"12" {

$x = Read-Number "Введите значение"

$result = [Math]::Atan($x)

if ($angleUnit -eq "D") { $result = $result \* 180 / [Math]::PI }

}

"13" {

$x = Read-Number "Введите число"

if ($x -le 0) { throw "Ошибка: логарифм определен только для положительных чисел" }

$result = [Math]::Log10($x)

}

"14" {

$x = Read-Number "Введите число"

if ($x -le 0) { throw "Ошибка: логарифм определен только для положительных чисел" }

$result = [Math]::Log($x)

}

"15" {

$a = Read-Number "Введите делимое"

$b = Read-Number "Введите делитель"

if ($b -eq 0) { throw "Ошибка: деление на ноль!" }

$result = $a % $b

}

"16" { exit }

default { Write-Host "Неверный выбор!" -ForegroundColor Red; continue }

}

Write-Host "Результат: $result" -ForegroundColor Green

}

catch {

Write-Host $\_.Exception.Message -ForegroundColor Red

}

}

Листинг 1.1 – Скрипт калькулятора

Запускаем калькулятор.

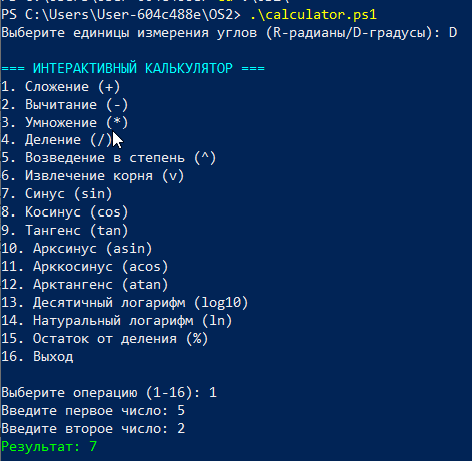


Рисунок 1.44 – Запуск калькулятора

Калькулятор проверили на работу всех функций. Работает корректно.

1. Написать скрипт для PowerShell для перевода температуры между тремя основными единицами измерения, при этом определение исходной единицы измерения происходит на основе введенного параметра (например, 100K – кельвин, 100С или просто 100 – цельсий, 100F – фарингейт).

Как и в прошлом задании, создаем файл и записываем туда скрипт.

function Convert-Temperature {

param (

[string]$InputTemperature

)

$ABSOLUTE\_ZERO\_C = -273.15

$ABSOLUTE\_ZERO\_F = -459.67

$ABSOLUTE\_ZERO\_K = 0

if ($InputTemperature -match "^([-+]?\d\*\.?\d+)\s\*([CFK]?)$") {

try {

$value = [double]$matches[1]

}

catch {

Write-Host "Ошибка: Введите корректное число (например, 25 или -10.5)." -ForegroundColor Red

return

}

$unit = $matches[2].ToUpper()

# Если единица не указана, предполагаем Цельсий

if (-not $unit) {

$unit = "C"

}

# Проверка на абсолютный ноль и конвертация

switch ($unit) {

"C" {

if ($value -lt $ABSOLUTE\_ZERO\_C) {

Write-Host "Ошибка: Температура не может быть ниже абсолютного нуля ($ABSOLUTE\_ZERO\_C °C)." -ForegroundColor Red

return

}

$celsius = $value

$fahrenheit = ($value \* 9/5) + 32

$kelvin = $value + 273.15

}

"F" {

if ($value -lt $ABSOLUTE\_ZERO\_F) {

Write-Host "Ошибка: Температура не может быть ниже абсолютного нуля ($ABSOLUTE\_ZERO\_F °F)." -ForegroundColor Red

return

}

$celsius = ($value - 32) \* 5/9

$fahrenheit = $value

$kelvin = ($value - 32) \* 5/9 + 273.15

}

"K" {

if ($value -lt $ABSOLUTE\_ZERO\_K) {

Write-Host "Ошибка: Температура не может быть ниже абсолютного нуля ($ABSOLUTE\_ZERO\_K K)." -ForegroundColor Red

return

}

$celsius = $value - 273.15

$fahrenheit = ($value - 273.15) \* 9/5 + 32

$kelvin = $value

}

}

# Форматированный вывод результатов

Write-Host "`nРезультаты конвертации:" -ForegroundColor Green

Write-Host " Цельсий: $($celsius.ToString('F2')) °C" -ForegroundColor Cyan

Write-Host " Фаренгейт: $($fahrenheit.ToString('F2')) °F" -ForegroundColor Cyan

Write-Host " Кельвин: $($kelvin.ToString('F2')) K" -ForegroundColor Cyan

}

else {

Write-Host "Ошибка: Неверный формат ввода. Используйте формат: число[единица], например: 25C, 100F, 300K или просто 25." -ForegroundColor Red

}

}

# Основной цикл программы

while ($true) {

Write-Host "`n=== КОНВЕРТЕР ТЕМПЕРАТУРЫ ===" -ForegroundColor Yellow

Write-Host "Введите температуру с единицей измерения (C, F, K)"

Write-Host "Примеры: 25C, 77F, 298K или просто 25 (по умолчанию Цельсий)"

Write-Host "Для выхода введите 'exit'`n"

$input = Read-Host "Введите температуру"

if ($input -eq "exit") {

Write-Host "Выход из программы..." -ForegroundColor Yellow

break

}

Convert-Temperature -InputTemperature $input

}

Листинг 1.2 – Скрипт для перевода температуры

Запускаем наш скрипт и проверяем на работоспособность.

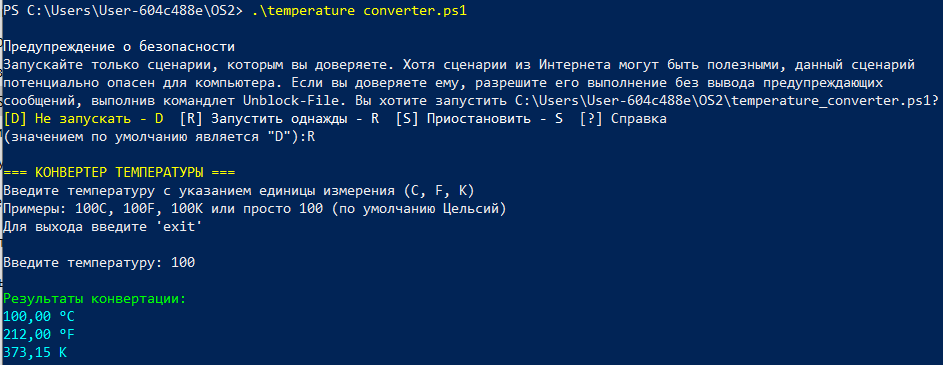


Рисунок 1.45 – Запуск конвертера температур

Скрипт для перевода температуры проверили на работоспособность. Работает корректно.

# **Постановка задачи для Linux**

## **Уровень 4-5**

ХУЙ ПИСЮНА 52!!!!!!

1. Определить команды аналогичные для команд, изученных в задании по Windows. Рассмотреть отличия в подходах к наименованию команд и параметров.

Таблица 2.1 – Аналоги команд

|  |  |  |  |
| --- | --- | --- | --- |
| **Windows CMD** | **Linux Bash** | **Назначение** | **Отличие в стиле** |
| dir | ls | Список файлов | ls — кратко, параметры через - |
| cd | cd | Переход в каталог | одинаково |
| md / mkdir | mkdir | Создание каталога | одинаково |
| rd / rmdir | rmdir | Удаление каталога | одинаково |
| del | rm | Удаление файла | rm — более мощная, может удалять рекурсивно |
| copy | cp | Копирование | cp — параметры через -r, -v |
| type | cat | Просмотр файла | cat — потоковый вывод |
| echo | echo | Вывод строки | одинаково |
| find | grep | Поиск по строке | grep — регулярные выражения |
| sort | sort | Сортировка строк | одинаково |
| systeminfo | uname, lscpu, lsb\_release | Информация о системе | разбито на несколько утилит |

Стиль Linux лаконичный, ориентирован на потоки и конвейеры. А параметры почти всегда начинаются с – ил --.

1. Написать пять bash-скриптов аналогичных пяти bat-файлам из заданий по Windows, но с использованием особенностей синтаксиса и возможностей bash. ([ss64](https://ss64.com/bash/syntax.html))

Создаем первый bash-скрипт и вводим туда код.

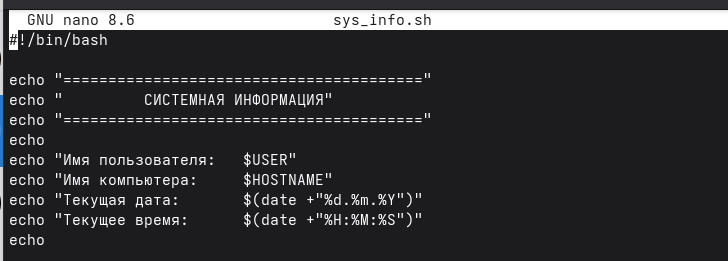


Рисунок 2.1 – Первый скрипт

Запускаем и проверяем наш скрипт.

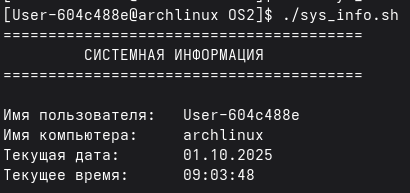


Рисунок 2.2 – Запуск первого скрипта

Создаем второй скрипт.

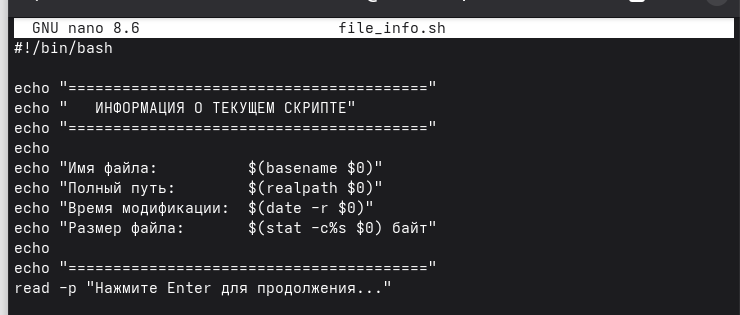


Рисунок 2.3 – Второй скрипт

Запускаем скрипт и проверяем на правильность исполнения.

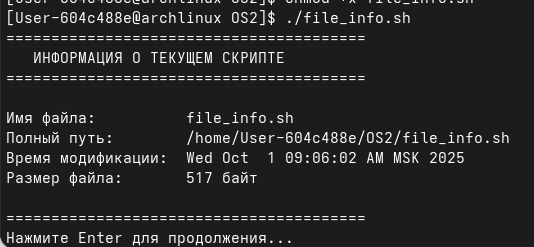


Рисунок 2.4 – Запуск второго скрипта

Создаем третий скрипт.

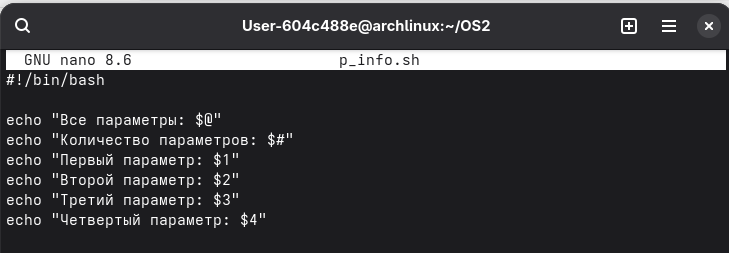


Рисунок 2.5 – Третий скрипт

Запускаем и проверяем на правильность исполнения третий скрипт.

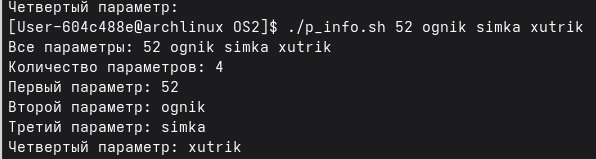


Рисунок 2.6 – Запуск третьего скрипта

Создаем четвертый скрипт.

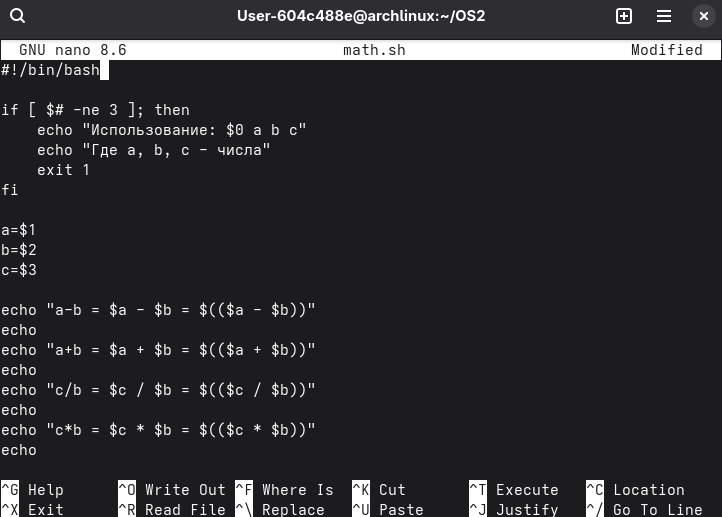


Рисунок 2.7 – Четвертый скрипт

Запускаем четвертый скрипт и проверяем правильность вычислений.

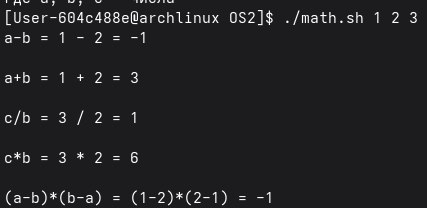


Рисунок 2.8 – Запуск четвертого скрипта

Создаем пятый скрипт.

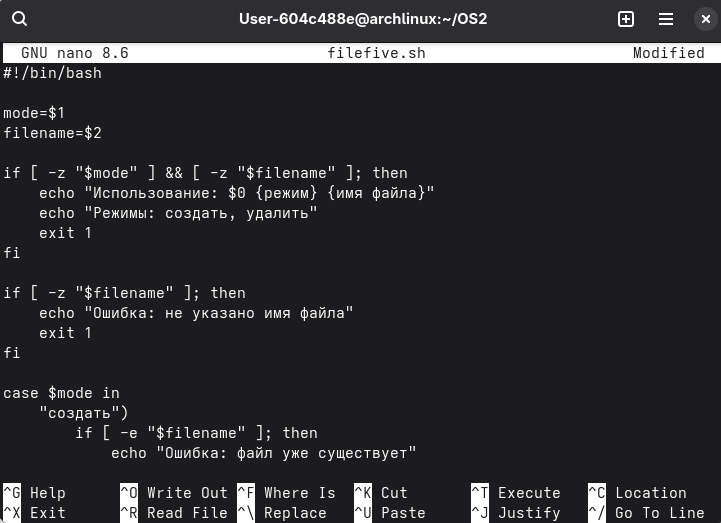


Рисунок 2.9 – Пятый скрипт

Запускаем пятый скрипт.

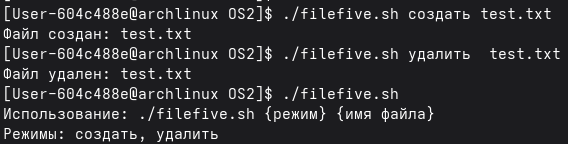


Рисунок 2.10 – Запуск пятого скрипта

Все bash-скрипты аналогичны тем, что были в Windows и работают по тому же принципу.

1. Исследуйте переменные окружения в Linux. В чём отличие от того, как это устроено в Windows?

Таблица 2.2 – Переменные окружения

|  |  |
| --- | --- |
| **Переменная** | **Назначение** |
| PATH | Список директорий, в которых система ищет исполняемые файлы |
| HOME | Домашний каталог текущего пользователя |
| USER | Имя текущего пользователя |
| LOGNAME | Имя пользователя, вошедшего в систему |
| SHELL | Путь к используемой командной оболочке |
| PWD | Текущий рабочий каталог |
| LANG | Язык и локаль по умолчанию |
| LC\_ALL | Переопределение всех локалей |
| TERM | Тип терминала |
| HOSTNAME | Имя компьютера |
| UID | Числовой идентификатор пользователя |
| MAIL | Путь к почтовому ящику пользователя |
| DISPLAY | Адрес графического дисплея |
| SHLVL | Уровень вложенности оболочек |
| PS1 | Основное приглашение командной строки, которое видно каждый раз, когда Bash ждет команду |
| PS2 | Вторичное приглашение, которое появляется, если не завершена команда |
| RANDOM | Генератор случайных чисел |
| GROUPS | Список групп, в которых состоит пользователь |
| BASH\_VERSION | Версия Bash |
| BASH\_SOURCE | Путь к исполняемому скрипту |
| ? | Код возврата последней команды |

В Linux переменные – это ключ=значение. Они используются через $VAR, а в Windows через %VAR%. В Linux устанавливаются через export VAR=value. Хранятся в env, printenv, set, а в Windows хранятся в системной и пользовательской области.

## **Уровень 7-8**

* + - 1. Напишите скрипт, который считает количество исполняемых и неисполняемых команд присутствующих в PATH.

Создаем файл и записываем туда наш скрипт.

#!/bin/bash

exec\_count=0

non\_exec\_count=0

# Разбиваем PATH на отдельные каталоги

IFS=':' read -ra dirs <<< "$PATH"

for dir in "${dirs[@]}"; do

# Проверка: существует ли каталог

if [[ -d "$dir" ]]; then

for file in "$dir"/\*; do

# Проверка: файл существует

[[ -e "$file" ]] || continue

if [[ -f "$file" && -x "$file" ]]; then

((exec\_count++))

elif [[ -f "$file" ]]; then

((non\_exec\_count++))

fi

done

fi

done

echo "Исполняемых файлов: $exec\_count"

echo "Неисполняемых файлов: $non\_exec\_count"

Листинг 2.1 – Скрипт для подсчета файлов

Затем запускаем наш скрипт и проверяем на работоспособность.

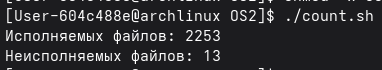


Рисунок 2.11 – Запуск скрипта

Скрипт для подсчета исполняемых и неисполняемых файлов работает корректно.

* + - 1. Напишите скрипт, который выводит список вложенных каталогов и файлов в указанном каталоге. При этом для вложенных каталогов выводится количество файлов внутри них, а для файлов выводится их размер в kb, mb, gb в зависимости от размера (значение размера не может превышать 1024, иначе перевод в следующую единицу измерения, например, kb -> mb).

Создаем файл и записываем туда наш скрипт.

#!/bin/bash

# Проверка: передан ли путь

if [[ -z "$1" ]]; then

echo "Использование: $0 /путь/к/каталогу"

exit 1

fi

target="$1"

# Проверка: существует ли каталог

if [[ ! -d "$target" ]]; then

echo "Ошибка: '$target' не является каталогом"

exit 1

fi

echo "Анализ каталога: $target"

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

# Функция для форматирования размера

format\_size() {

local size=$1

local unit="KB"

if (( size > 1024 )); then

size=$((size / 1024))

unit="MB"

fi

if (( size > 1024 )); then

size=$((size / 1024))

unit="GB"

fi

echo "${size} ${unit}"

}

# Перебор содержимого

for item in "$target"/\*; do

if [[ -d "$item" ]]; then

count=$(find "$item" -type f | wc -l)

echo "[DIR] $(basename "$item") — файлов внутри: $count"

elif [[ -f "$item" ]]; then

size\_bytes=$(stat -c %s "$item")

size\_kb=$((size\_bytes / 1024))

formatted=$(format\_size "$size\_kb")

echo "[FILE] $(basename "$item") — размер: $formatted"

fi

done

Листинг 2.2 – Скрипт, выводящий список файлов

Запускаем наш скрипт и проверяем на работоспособность.

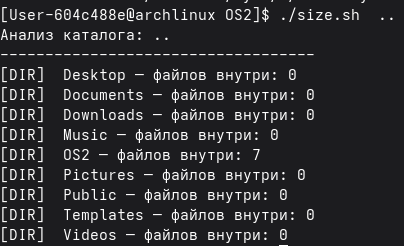


Рисунок 2.12 – Запуск скрипта

Скрипт работает корректно. В результате все по 0, потому файлы отсутствуют.

# **Заключение**

В ходе выполнения лабораторной работы было получено комплексное представление о работе с командными оболочками операционных систем Windows и Linux.

На платформе Windows были освоены команды cd, copy, xcopy, move, del, dir, md, rd, fc, ren, replace, а также операторы перенаправления (>, >>), конвейеризации (|) и соединения (&). Особое внимание уделялось созданию и тестированию .bat-файлов с использованием системных переменных, параметров и расширений. Это позволило автоматизировать рутинные задачи и закрепить понимание структуры командной строки Windows.

На платформе Linux были изучены аналогичные команды (ls, cp, mv, rm, mkdir, rmdir, find, echo, cat, sort, grep, stat, xargs, chmod, df, du) и написаны Bash-скрипты, повторяющие функциональность .bat-файлов. Также были реализованы более сложные скрипты, включая анализ переменной PATH и обработку содержимого каталогов с динамическим форматированием размеров файлов.

Были разработаны интерактивные скрипты на PowerShell и Bash, демонстрирующие работу с пользовательским вводом, математическими операциями, логикой ветвления и форматированием. Это позволило углубить понимание синтаксиса и возможностей каждой оболочки.