МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ

ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 6-05-0612-01 Программная инженерия

**ОТЧЕТ**

**К ЛАБОРАТОРНОЙ РАБОТЕ 2 НА ТЕМУ:**

«Знакомство с командной оболочкой операционной системы»

Выполнил студент Херунцева Дарья Юрьевна

Группа 3 курс 6-1 группа

Минск 2025

**Введение**

Данная работа посвящена изучению и практическому применению командных оболочек и скриптинга в операционных системах Windows и Linux Mint, выполненной в рамках учебного задания уровня 7-8. Целью работы стало освоение базовых и продвинутых инструментов управления файлами, каталогами и переменными окружения, а также разработка скриптов для автоматизации задач. В ходе выполнения были исследованы аналоги команд cmd.exe и bash, что позволило выявить ключевые различия в синтаксисе, подходах к наименованию и параметрам, характерных для Windows и Unix-подобных систем. Особое внимание уделено созданию пяти bash-скриптов, эквивалентных bat-файлам из Windows-задания, с использованием особенностей синтаксиса и возможностей bash, а также анализу переменных окружения в Linux по сравнению с Windows. Кроме того, были разработаны два специализированных скрипта для Linux Mint: один для подсчёта исполняемых и неисполняемых команд в PATH, и другой для рекурсивного вывода структуры каталогов с деталями о файлах и их размерах. Работа проводилась с использованием терминала Linux Mint и текстового редактора nano, что обеспечило практическую реализацию всех задач. Результаты демонстрируют понимание различий между операционными системами и способность адаптировать решения под специфические требования Unix-систем, таких как Linux Mint, используемая в образовательных целях.

### Основные этапы установки Windows

1. Перейти в домашний каталог вашего пользователя (созданного в прошлой лабораторной работе) и создать каталог OS2
2. Выполнить в нём бат-файл prep.bat (лежит вместе с условием на diskstation)
3. Открыть cmd в данном каталоге путём указания слова cmd в адресной строке проводника

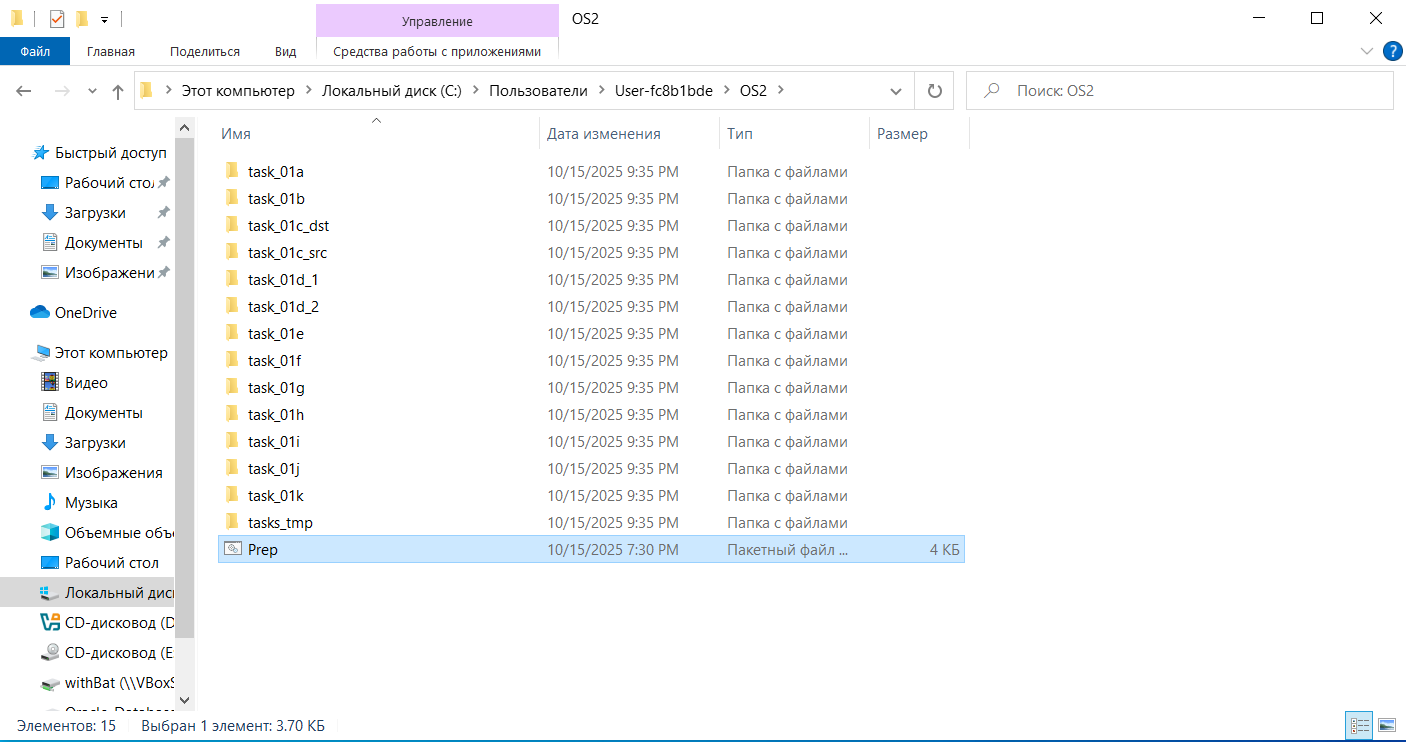


Рисунок 1 – Раскрыли файл Prep в каталоге OS2

1. На основе данных файлов и каталогов выполнить следующее (в скобках указана используемая в задании команда. Также уточняю, все задания выполняются ровно в одну команду за раз!):

(**cd**) Название команды cd является сокращением от английского "Change Directory" (сменить директорию). Оно отражает основную функцию — изменение текущей рабочей директории. Краткое название выбрано для удобства ввода, так как смена директорий — одна из самых частых операций в командной строке. Например, команда cd \Users переходит в папку Users, а cd .. поднимается на уровень выше. Двухбуквенное название минимизирует усилия при частом использовании, что типично для DOS-команд, где экономия символов была важна из-за ограничений интерфейса.

(**cd**) перейти в каталог taks\_01i/Русский с использованием абсолютного пути;

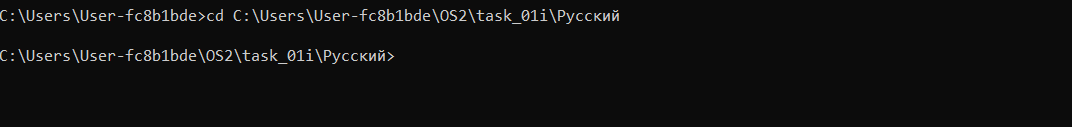


Рисунок 2 – перейти в каталог Русский

(**cd**) перейти в каталог taks\_01i/English с использованием относительных путей;

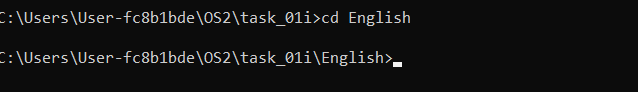


Рисунок 3 – перейти в каталог English

(**copy**) cкопировать все файлы, имеющие расширение txt из каталога tasks\_tmp в каталог task\_01a;

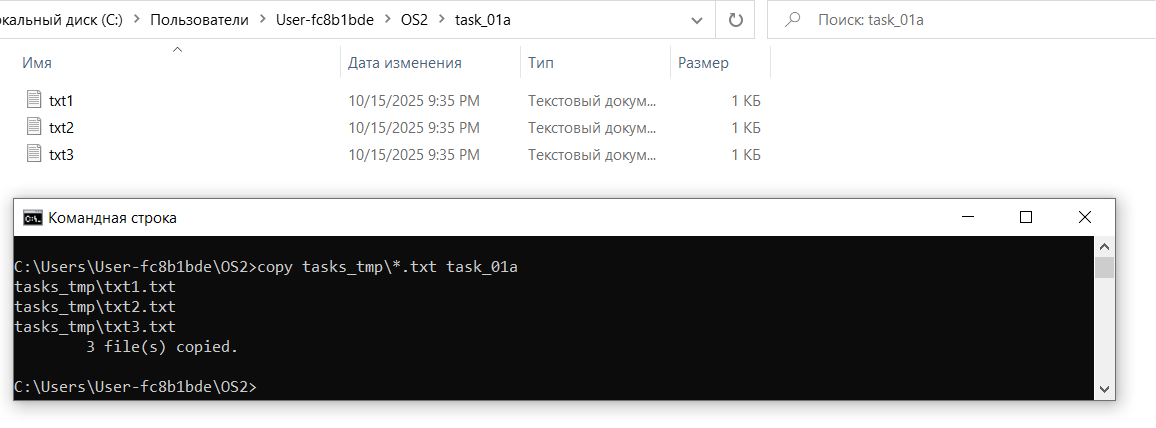


Рисунок 4 – Скопировать все файлы

(**copy**) Команда copy названа напрямую от английского слова "copy" (копировать), что точно описывает её назначение — создание копии файла или файлов в указанном месте. Название интуитивно понятно и отражает простую операцию. Например, copy file.txt D:\ копирует файл file.txt на диск D:. Однословное название без сокращений выбрано для ясности, так как копирование — базовая файловая операция, и команда должна быть легко узнаваемой для пользователей.

(**copy**) повторить туже операцию, но при этом команда должна запрашивать подтверждение на перезапись существующих файлов;

.

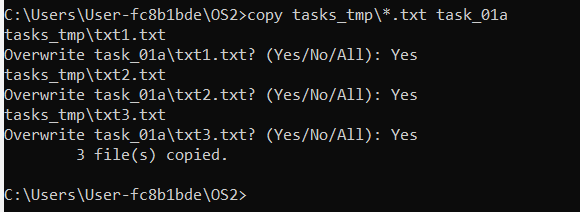


Рисунок 5 – Повторить туже операцию с подтверждением

(**copy**) скопировать файл из каталога task\_01a в каталог task\_01a, но с новым именем;

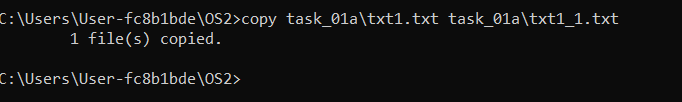


Рисунок 6– скопировать файл из каталога task\_01a

(**copy**) объединить файлы txt1.txt и txt2.txt в файл merged.txt;

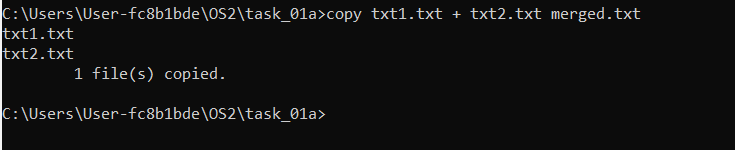


Рисунок 7 – Объединить файлы

(**copy**) скопировать текст, введенный в консоль в файл с именем console.txt (подсказка – Con)

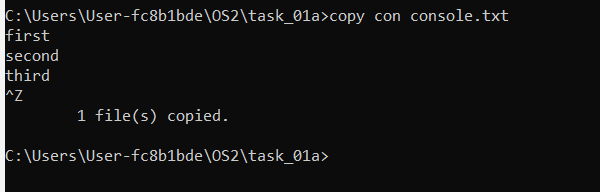


Рисунок 8 – Копировать текст

(**copy**) добавить в конец файла txt3 информацию вводимую с консоли;

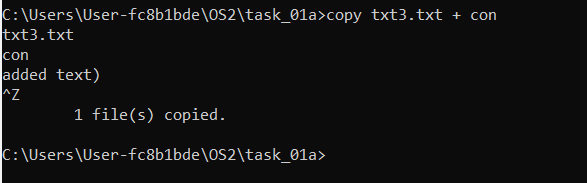


Рисунок 9 – Добавить в конец файла txt3 информацию

(**xcopy**) Название xcopy расшифровывается как "extended copy" (расширенное копирование). Префикс "x" (от "extended") указывает на улучшенную версию команды copy, добавляющую возможности, такие как рекурсивное копирование директорий, фильтрация по датам или атрибутам. Например, xcopy C:\source D:\dest /s копирует директории с подкаталогами. Префикс "x" — распространённый способ в DOS/Windows для обозначения расширенных версий (ср. fdisk и xfsck), а название остаётся коротким, сохраняя связь с copy.

(**xcopy**) скопировать все файлы, каталоги и подкаталоги, включая пустые, а также скрытые и системные из task\_01b в task\_01b\_dst\_1

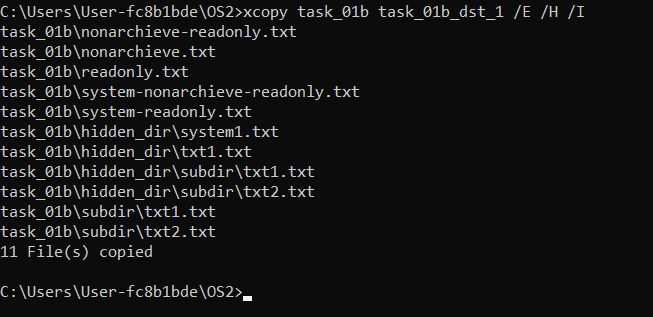


Рисунок 10 –Скопировать все файлы каталоги и подкаталоги

(**xcopy**) скопировать все файлы с атрибутами «архивный» и «только для чтения» с сохранением этого атрибута для файлов-результатов из task\_01b в task\_01b\_dst\_2;

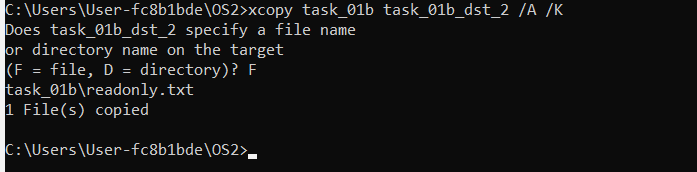


Рисунок 11 –Скопировать все файлы с атрибутами «архивный»

(**move**) Команда move названа от английского слова "move" (переместить), что прямо указывает на её функцию — перемещение или переименование файлов и директорий. Простое и описательное название выбрано для интуитивности. Например, move file.txt D:\ перемещает файл, а move old.txt new.txt переименовывает его. Однословное название отражает универсальность команды, которая объединяет перемещение и переименование, минимизируя количество команд для пользователя.

(**move**) переместить все текстовые файлы с расширением .txt из каталога task\_01c\_src в каталог task\_01c\_dst, зайти в каталог task\_01d\_2 и удалить текст в файлах;

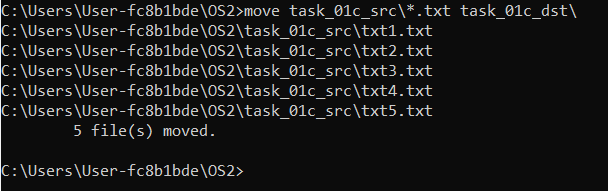


Рисунок 12 –Переместить все текстовые файлы

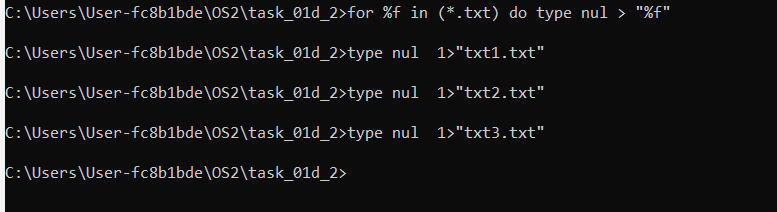


Рисунок 13 –Удалить файлы

(**replace**) Команда replace названа от английского "replace" (заменить), так как она заменяет файлы в целевой директории файлами из источника. Название чётко описывает действие, например, replace source.txt D:\dest заменяет файл в папке dest. Использование полного слова вместо сокращения подчёркивает специфичность операции, отличая её от copy, и делает команду понятной для пользователей, выполняющих целевую замену файлов.

(**replace**) заменить файлы в каталоге task\_01d\_2 файлами, расположенными в каталоге task\_01d\_1;



Рисунок 14 –Заменить файлы

(**ren**) Команда ren — это сокращение от "rename" (переименовать), отражающее её функцию переименования файлов или директорий. Короткое название выбрано для экономии ввода, так как переименование — частая операция. Например, ren old.txt new.txt меняет имя файла. Сокращение до трёх букв соответствует традиции DOS использовать минималистичные названия для базовых операций, сохраняя интуитивную связь с действием.

(**ren**) изменить расширение всех файлов в каталоге task\_01e на .bmp;



Рисунок 15 – изменить расширение

(**ren**) изменить имя файла txt1.bmp на «имя вашего пользователя». Ren

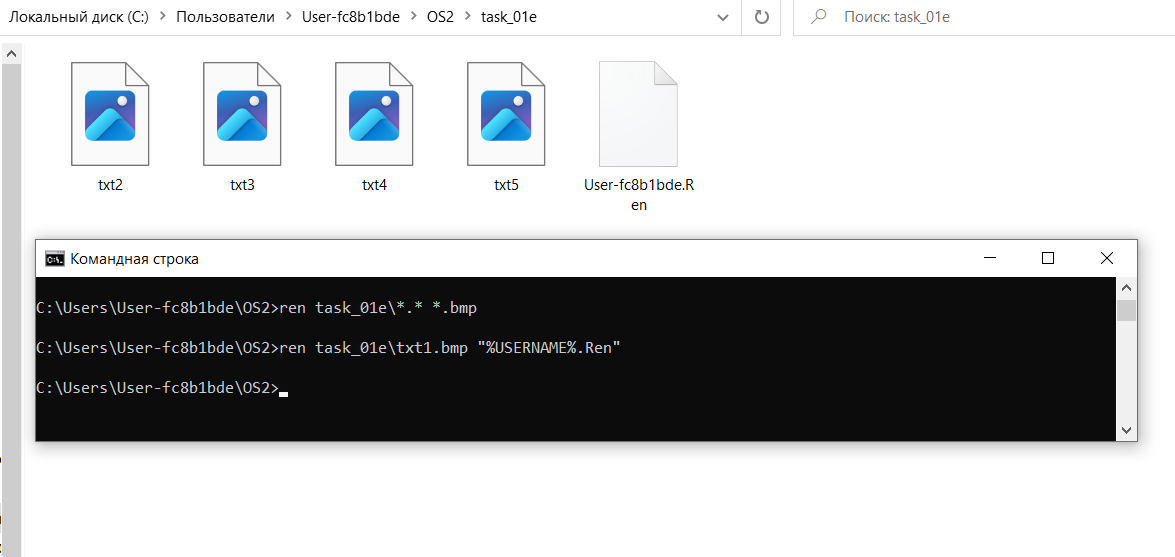


Рисунок 16 – изменить имя файла txt1.bmp

(**fc**). Название fsutil расшифровывается как "file system utility" (утилита файловой системы). Оно отражает её назначение — выполнение операций с файловой системой, таких как управление томами или настройка атрибутов. Например, fsutil fsinfo drives показывает доступные диски. Название комбинирует сокращение "fs" (file system) с "util" (utility), что типично для утилит Windows, где описательное название указывает на специализированный инструмент.

(**fc**) сравнить два файла в каталоге task\_01f и сохранить результат сравнения в result.txt;

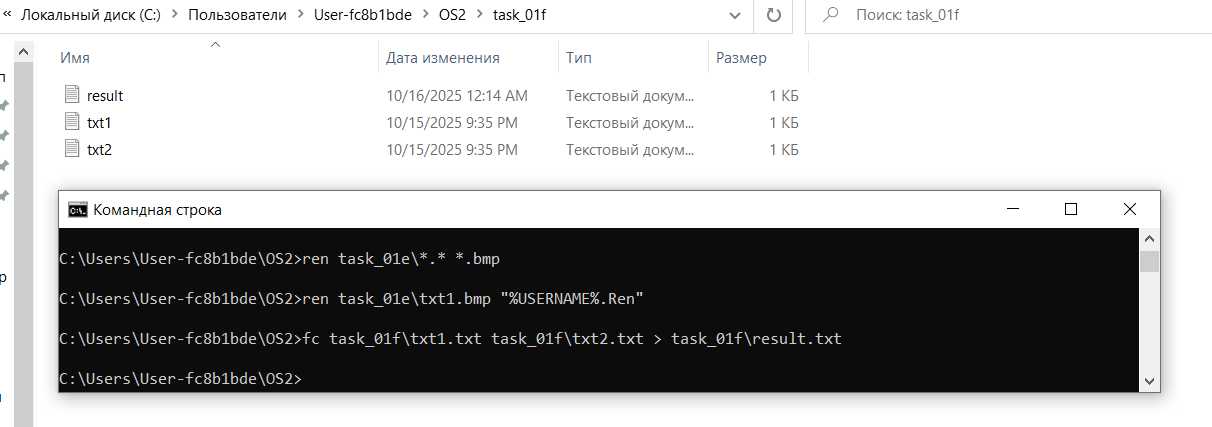


Рисунок 17 – Сравнить два файла в каталоге task\_01f

(**del**) Команда del — сокращение от "delete" (удалить), описывающее её функцию удаления файлов. Краткое название выбрано для удобства, так как удаление — частая операция. Например, del file.txt удаляет файл. Трёхбуквенное сокращение следует традиции DOS, где команды минимизированы для быстрого ввода, но остаются узнаваемыми благодаря прямой связи с английским глаголом.

(**del**) удалить файл txt1 из каталога task\_01g при этом запросив подтверждение на удаление

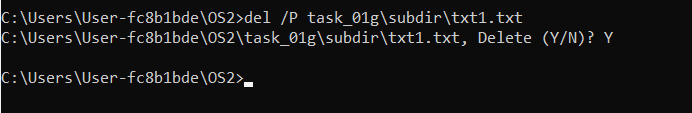


Рисунок 18 – Удалить файл txt1 из каталога task\_01g

(**del**) удалить все файлы с атрибутом «Системный» из каталога task\_01g

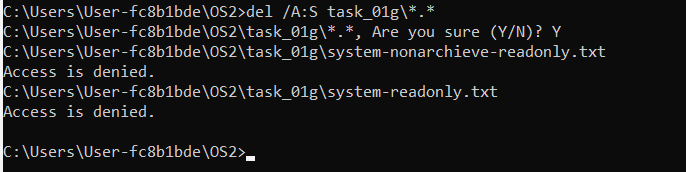


Рисунок 19 – Удалить все файлы с атрибутом «Системный»

(**del**) удалить все файлы из каталога task\_01g так, чтобы остались только файлы с атрибутом «Только для чтения»;

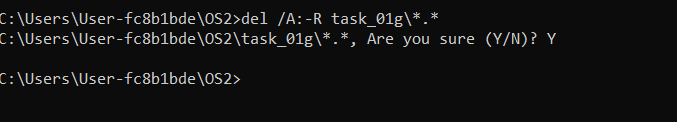


Рисунок 20 –Удалить все файлы из каталога task\_01g

(**del**) удалить все оставшиеся файлы и дочерние каталоги из каталога task\_01g



Рисунок 21 – Удалить все оставшиеся файлы

(**dir**) Команда dir — сокращение от "directory" (директория), обозначающее её функцию — отображение списка файлов и папок в текущей директории. Например, dir /p выводит содержимое с паузой. Название выбрано как сокращение ключевого слова, связанного с файловой системой, и является одним из самых узнаваемых в DOS/Windows благодаря своей простоте и частому использованию.

(**dir**) постранично вывести содержимое каталога C:\Windows, включая вложенные каталоги и файлы (постранично – информация выводится частями с ожиданием нажатия клавиши для вывода следующей части информации);

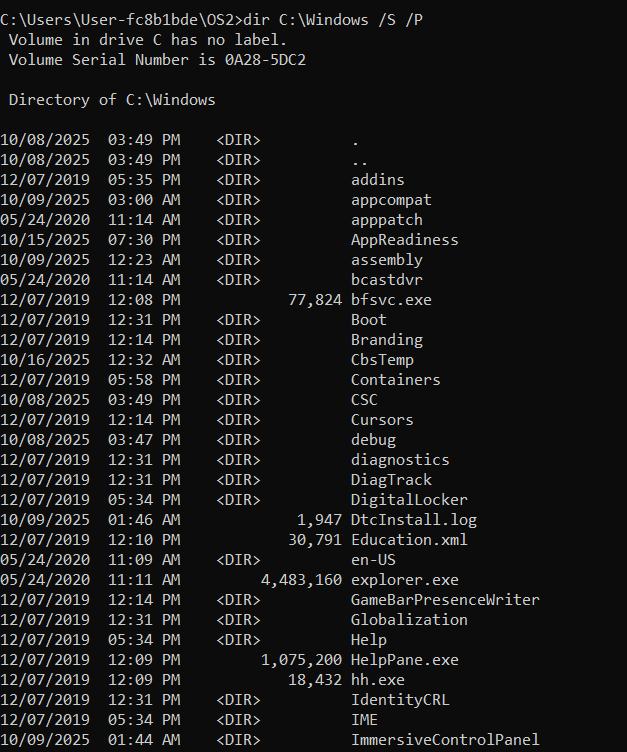


Рисунок 22 – постранично вывести содержимое каталога C:\Windows

(**dir**) постранично вывести 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;

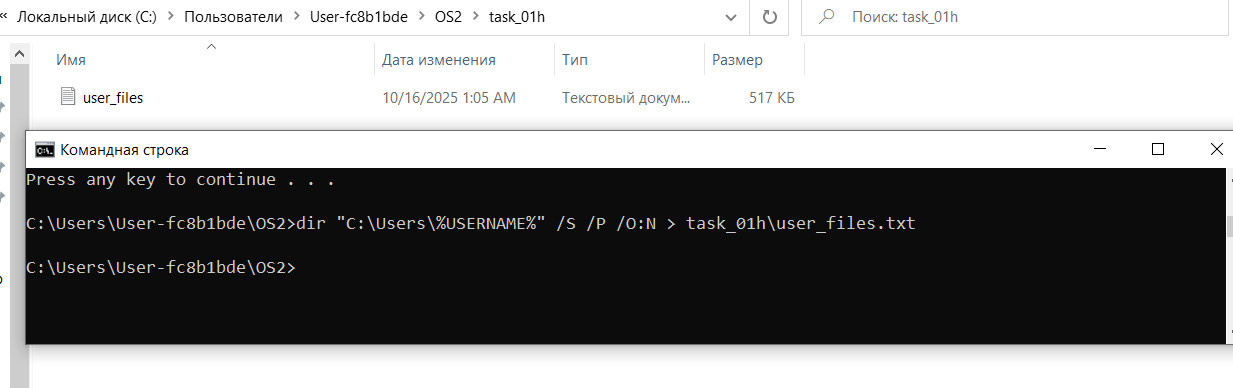


Рисунок 23 – Команда

(**md**) Команда md — сокращение от "make directory" (создать директорию), что отражает её назначение — создание новой папки. Например, md newfolder создаёт папку newfolder. Двухбуквенное название максимально укорочено для удобства ввода, следуя традиции DOS, где базовые операции обозначались минимальными сокращениями, сохраняющими связь с функцией.

(**md**) создать каталог task\_01j в вашем каталоге OS2



Рисунок 25 – создание каталога

(**md**) создать иерархию каталогов task\_01j\sub\dir\full\path;

(**rd**) Команда rd — сокращение от "remove directory" (удалить директорию), указывающее на её функцию удаления папок. Например, rd emptyfolder удаляет пустую папку, а rd /s folder — непустую. Как и md, двухбуквенное название выбрано для краткости, что типично для парного набора команд создания/удаления в DOS, где md и rd логически связаны.

(**rd**) удалить пустой каталог \task\_01k\empty\_subdir;

(**rd**) удалить не пустой каталог \task\_01k\subdir.

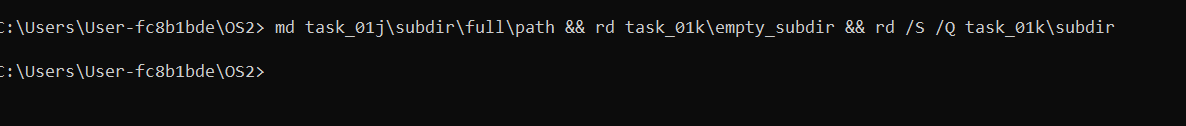


Рисунок 26 – Создать иерархию каталогов, удалить пустой и непустой каталоги

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

* вернуться в папку OS2;
* создать текстовый файл со списком студентов группы и их номерами книжек (например, Иванов 1) используя команду **type** иввод с консоли;
* выполнить поиск студента в файле по его фамилии;
* добавить ещё студентов в конец файла;
* выведите список студентов в отсортированном порядке и после этого выйдите из папки OS2;

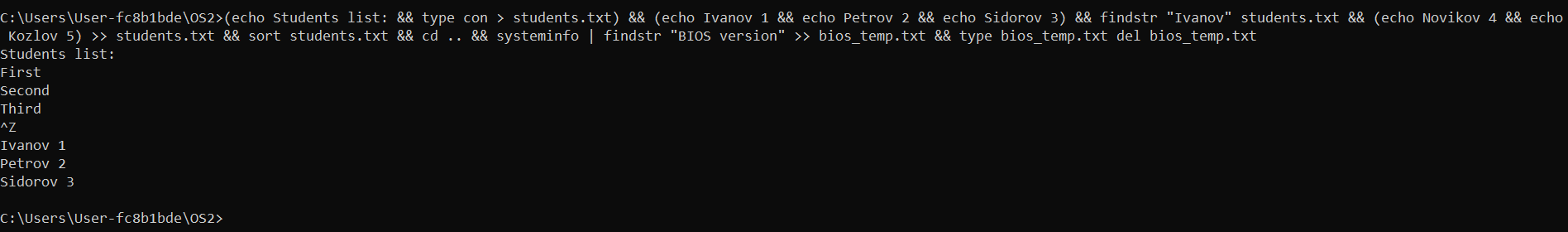


Рисунок 27 – Создать список студентов

1. Создать 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.*

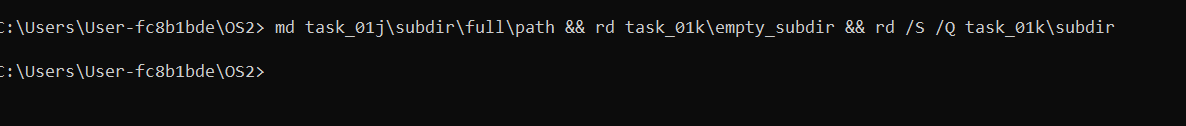
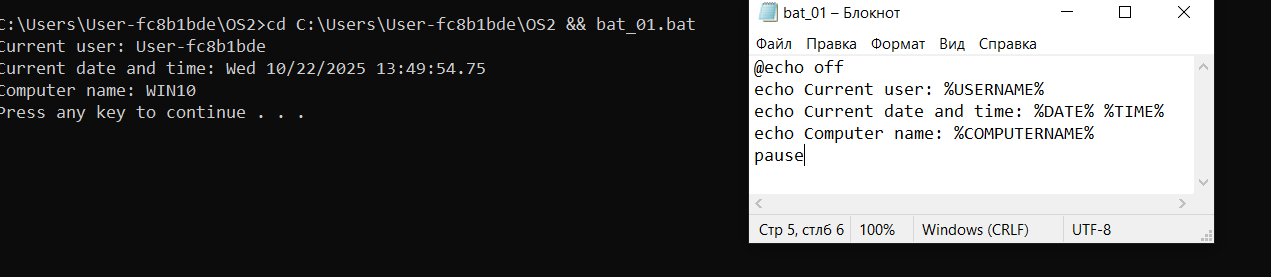


Рисунок 28 – файл bat\_01

1. Создать 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.*

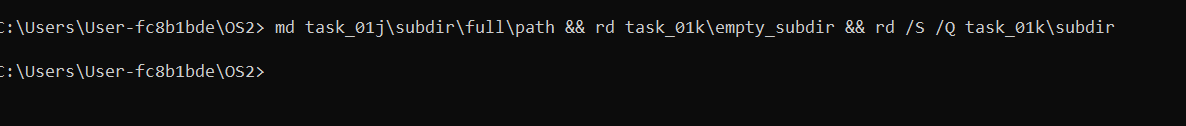
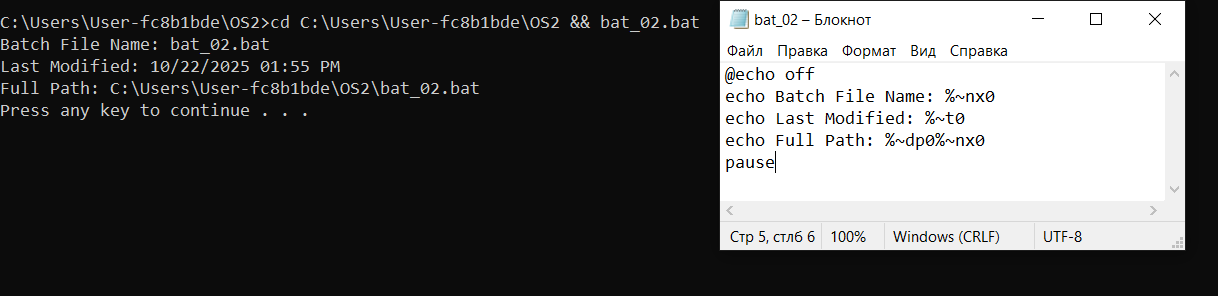


Рисунок 29 – файл bat\_02

1. Создать 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.*

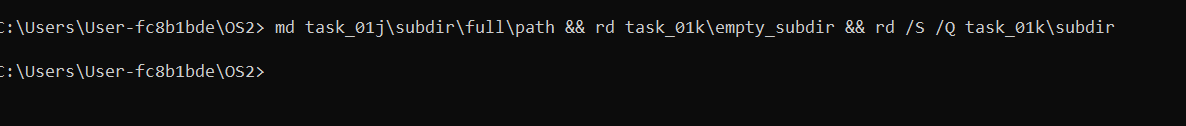
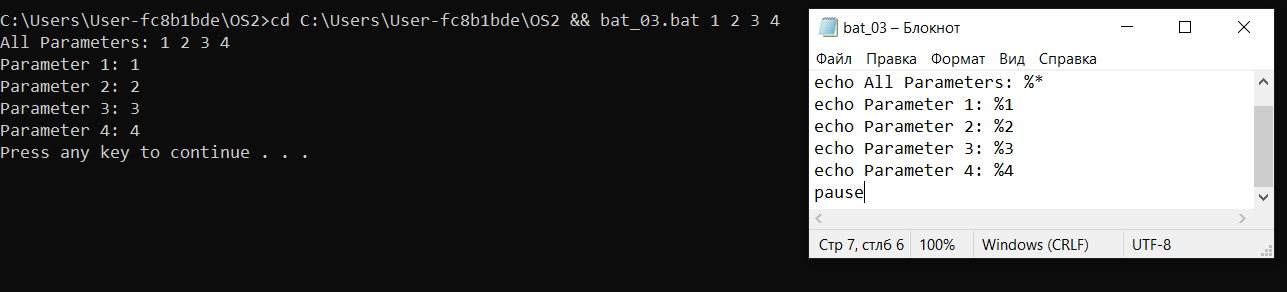


Рисунок 30 – файл bat\_03

1. Создать 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.*

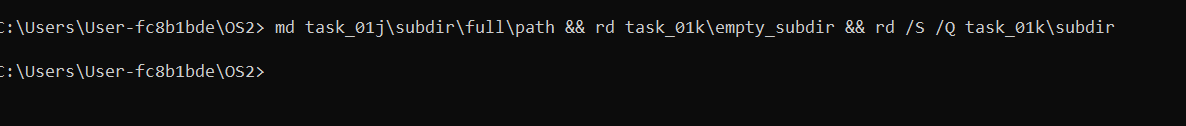
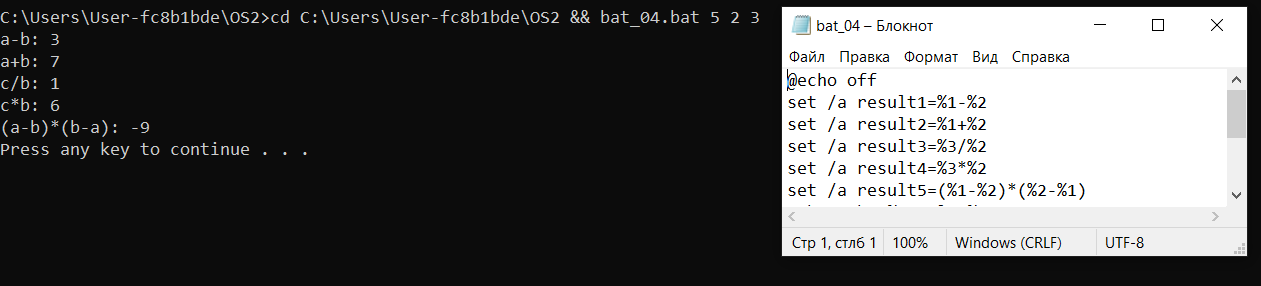


Рисунок 31 – файл bat\_04

1. Создать 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.

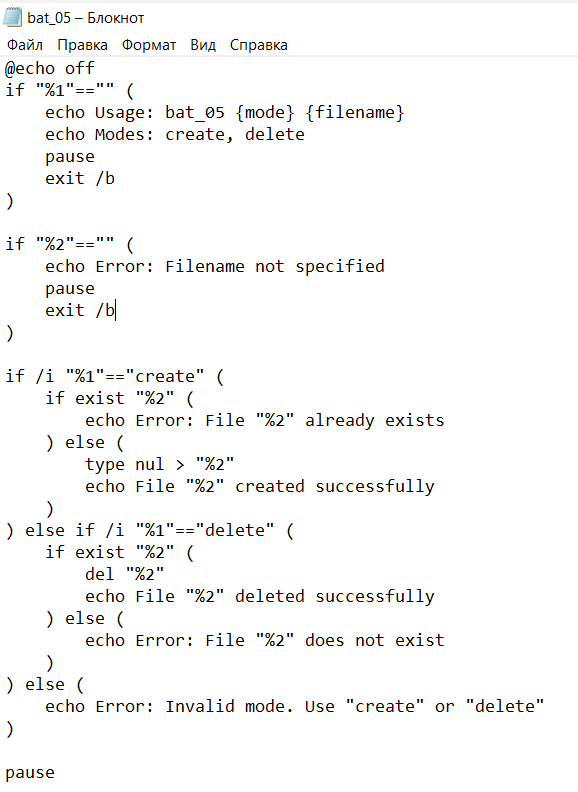


Рисунок 38 – bat\_05

1. 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.

Этот скрипт создаёт интерактивный калькулятор в PowerShell с поддержкой арифметических операций (+, -, \*, /), возведения в степень (^), извлечения корня (через возведение в дробную степень), тригонометрических функций (sin, cos, tan), десятичного (log10) и натурального (log) логарифмов, а также остатка от деления (mod)

|  |
| --- |
| param (  [switch]$Exit  )  if ($Exit) {  exit  }  while ($true) {  Write-Host "`nInteractive Calculator" -ForegroundColor Green  Write-Host "Available operations: +, -, \*, /, ^ (power), mod (remainder), sin, cos, tan, log (ln), log10, sqrt (root)" -ForegroundColor Yellow  Write-Host "Examples: 2 + 3, 5 ^ 2, sin 0.5, sqrt 16, 10 mod 3" -ForegroundColor Yellow  Write-Host "Enter expression (or 'exit' to quit):" -ForegroundColor Cyan  $input = Read-Host  if ($input -eq "exit") {  break  }  try {  # Разбиваем входные данные на части  $parts = $input -split ' '  if ($parts.Length -eq 0) {  Write-Host "Error: Enter an expression." -ForegroundColor Red  continue  }  $operation = $parts[0].ToLower()  $num1 = $null  if ($parts.Length -gt 1) {  $num1 = [double]$parts[1]  }  $num2 = $null  if ($parts.Length -gt 2) {  $num2 = [double]$parts[2]  }  $result = $null  # Проверяем однозначные функции  if ($num1 -ne $null -and $num2 -eq $null -and $operation -in "sin", "cos", "tan", "log", "log10", "sqrt") {  switch ($operation) {  "sin" { $result = [math]::Sin($num1) }  "cos" { $result = [math]::Cos($num1) }  "tan" { $result = [math]::Tan($num1) }  "log" { if ($num1 -le 0) { throw "Logarithm undefined for non-positive numbers" } $result = [math]::Log($num1) }  "log10" { if ($num1 -le 0) { throw "Logarithm undefined for non-positive numbers" } $result = [math]::Log10($num1) }  "sqrt" { if ($num1 -lt 0) { throw "Square root undefined for negative numbers" } $result = [math]::Sqrt($num1) }  }  }  # Проверяем бинарные операции  elseif ($num1 -ne $null -and $num2 -ne $null -and $operation -in "+", "-", "\*", "/", "^", "mod") {  switch ($operation) {  "+" { $result = $num1 + $num2 }  "-" { $result = $num1 - $num2 }  "\*" { $result = $num1 \* $num2 }  "/" { if ($num2 -eq 0) { throw "Division by zero" } $result = $num1 / $num2 }  "^" { $result = [math]::Pow($num1, $num2) }  "mod" { $result = $num1 % $num2 }  }  }  else {  throw "Error: Invalid expression format. Use: operation number (e.g., sin 0.5) or number operation number (e.g., 2 + 3)."  }  Write-Host "Result: $result" -ForegroundColor Green  }  catch {  Write-Host "Error: $($\_.Exception.Message)" -ForegroundColor Red  }  } |

Листинг1 – Скрипт калькулятора

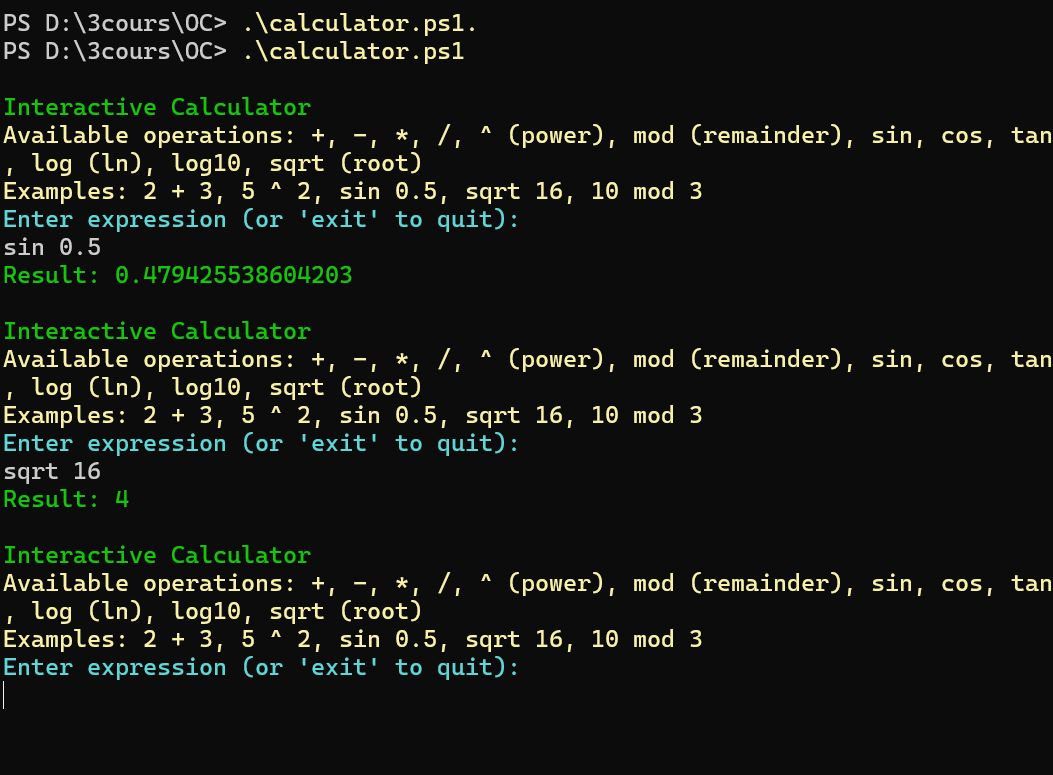


Рисунок 39 – Вывод

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).

#### Скрипт для PowerShell: Перевод температуры

Этот скрипт переводит температуру между Кельвинами (K), Цельсиями (C) и Фаренгейтами (F) на основе введённого формата (например, 100K, 100C, 100F).

|  |
| --- |
| param (  [string]$Temperature  )  function Convert-Temperature {  param ([string]$tempStr)  if (-not $tempStr) {  Write-Host "Enter temperature in format: 100K (Kelvin), 100C (Celsius), 100F (Fahrenheit)" -ForegroundColor Red  return  }  $value = [double]::Parse($tempStr -replace '[^0-9\.]')  $unit = $tempStr -replace '[0-9\.]'  switch ($unit.ToUpper()) {  "K" {  $celsius = $value - 273.15  $fahrenheit = $celsius \* 9/5 + 32  Write-Host "$value K = $celsius C = $fahrenheit F" -ForegroundColor Green  }  "C" {  $kelvin = $value + 273.15  $fahrenheit = $value \* 9/5 + 32  Write-Host "$value C = $kelvin K = $fahrenheit F" -ForegroundColor Green  }  "F" {  $celsius = ($value - 32) \* 5/9  $kelvin = $celsius + 273.15  Write-Host "$value F = $celsius C = $kelvin K" -ForegroundColor Green  }  default {  Write-Host "Invalid unit format. Use K, C, or F." -ForegroundColor Red  }  }  }  if ($Temperature) {  Convert-Temperature -tempStr $Temperature  } else {  while ($true) {  $tempInput = Read-Host "Enter temperature (or 'exit' to quit)"  if ($tempInput -eq "exit") { break }  Convert-Temperature -tempStr $tempInput  }  } |

Листинг 2 – Скрипт для перевода тепмературы

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).

|  |
| --- |
| param (  [string]$Temperature  )  function Convert-Temperature {  param ([string]$tempStr)  if (-not $tempStr) {  Write-Host "Enter temperature in format: 100K (Kelvin), 100C (Celsius), 100F (Fahrenheit)" -ForegroundColor Red  return  }  $value = [double]::Parse($tempStr -replace '[^0-9\.]')  $unit = $tempStr -replace '[0-9\.]'  switch ($unit.ToUpper()) {  "K" {  $celsius = $value - 273.15  $fahrenheit = $celsius \* 9/5 + 32  Write-Host "$value K = $celsius C = $fahrenheit F" -ForegroundColor Green  }  "C" {  $kelvin = $value + 273.15  $fahrenheit = $value \* 9/5 + 32  Write-Host "$value C = $kelvin K = $fahrenheit F" -ForegroundColor Green  }  "F" {  $celsius = ($value - 32) \* 5/9  $kelvin = $celsius + 273.15  Write-Host "$value F = $celsius C = $kelvin K" -ForegroundColor Green  }  default {  Write-Host "Invalid unit format. Use K, C, or F." -ForegroundColor Red  }  }  }  if ($Temperature) {  Convert-Temperature -tempStr $Temperature  } else {  while ($true) {  $tempInput = Read-Host "Enter temperature (or 'exit' to quit)"  if ($tempInput -eq "exit") { break }  Convert-Temperature -tempStr $tempInput  }  } |

Листинг 2 – Выбор региона

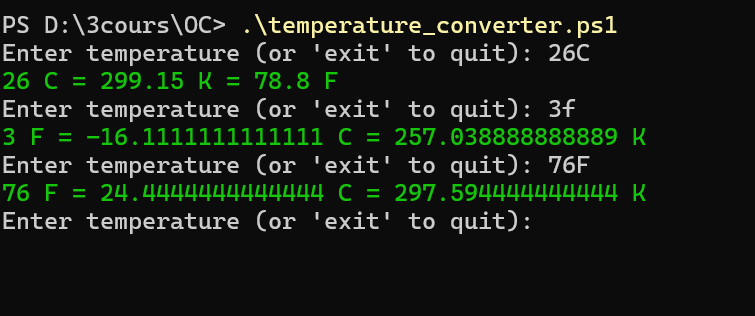


Рисунок 40 – Вывод

1. Исследуйте что такое переменные окружения и зачем они нужны. Какая самая важная переменная окружения для нас при работе в консоли?

Переменные окружения в Windows — это именованные значения, которые хранят информацию, используемую операционной системой, приложениями и командной строкой для настройки их поведения. Они представляют собой пары ключ-значение, доступные для процессов в системе, и содержат данные, такие как пути к директориям, настройки пользователя или системные параметры. Переменные окружения задаются на уровне пользователя или системы и могут быть временными (для текущей сессии) или постоянными (сохранёнными в реестре Windows). Их основное назначение — обеспечивать гибкость и переносимость программ, позволяя им адаптироваться к конфигурации системы без жёстко прописанных значений.

Самая важная переменная окружения для работы в консоли Windows — это PATH. Она определяет список директорий, в которых система ищет исполняемые файлы (например, .exe, .bat, .cmd) при вводе команды без указания полного пути. Без PATH пользователю пришлось бы каждый раз указывать полный путь к программам.

**Постановка задачи для Linux:**

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

В Linux, особенно в Linux Mint, существуют команды, аналогичные командам Windows, таким как cd, copy, xcopy, move, replace, ren, fs, del, dir, md, rd и systeminfo, но они отличаются по синтаксису, названиям и подходам, отражая философию Unix-подобных систем. Команды Linux, основанные на Unix, обычно короткие, чувствительны к регистру и модульные, тогда как команды Windows, унаследованные от DOS, более описательные и нечувствительны к регистру. Ниже описаны аналоги каждой команды Windows в Linux, их использование и различия в подходах к именованию и параметрам с использованием стандартных утилит Linux, как описано на ss64.

Команда Windows cd (change directory) имеет прямой аналог в Linux — команду cd. Обе изменяют текущую рабочую директорию, принимая путь в качестве аргумента (например, cd /home/user в Linux или cd \Users\user в Windows). В Linux cd чувствительна к регистру, пути используют прямой слэш (/), а не обратный (), как в Windows. Параметры, такие как cd .. для перехода на уровень выше, работают одинаково, но Linux поддерживает дополнительные возможности, например cd - для возврата в предыдущую директорию.

Для команды Windows copy, которая копирует файлы, эквивалентом в Linux является cp. Команда cp копирует файлы или директории (с опцией -r или --recursive для директорий) из источника в назначение (например, cp file1.txt /home/user/dest/). В отличие от copy, где опция /y подавляет запрос на перезапись, cp требует -f (force) для перезаписи или -i (interactive) для запроса подтверждения. В Linux cp чувствительна к регистру для имен файлов и опций, использует одинарный (-f) или двойной (--force) дефис для параметров, в отличие от слэшей Windows (/y).

Команда Windows xcopy, расширенная утилита для копирования директорий и выборочного копирования, аналогична cp с опцией -r для рекурсивного копирования (например, cp -r source\_dir/ dest\_dir/ соответствует xcopy source\_dir dest\_dir /s). Однако для продвинутых функций xcopy, таких как копирование по атрибутам (/d для новых файлов), в Linux используются комбинации cp с find или rsync. Например, rsync -a source/ dest/ обеспечивает синхронизацию, аналогичную xcopy /s /d. Linux предпочитает комбинировать модульные утилиты, в отличие от интегрированного подхода xcopy, и использует опции с дефисами (--archive), а не слэши.

Команда Windows move, перемещающая файлы или директории, соответствует команде Linux mv. Обе переименовывают или перемещают элементы (например, mv file1.txt /home/user/ или move file1.txt \Users\user). В Linux mv по умолчанию перезаписывает файлы, в отличие от move, которая запрашивает подтверждение без /y. Для mv используются -i для запроса подтверждения или -f для принудительной перезаписи, с чувствительными к регистру путями и опциями, в отличие от нечувствительного к регистру подхода Windows.

Команда Windows replace, заменяющая файлы в целевой директории файлами из источника, не имеет прямого аналога в Linux. Ближайшая функциональность достигается с помощью cp -f для перезаписи (например, cp -f source/file.txt dest/) или rsync --delete для синхронизации и замены. В Linux функциональность replace разделена между утилитами, а опции cp и rsync используют дефисы, требуя явного указания источника и назначения, в отличие от слэшей Windows.

Команда Windows ren (rename) также соответствует mv в Linux, так как mv выполняет как переименование, так и перемещение (например, mv oldname.txt newname.txt эквивалентно ren oldname.txt newname.txt). В Linux mv чувствительна к регистру, и нет специальных опций только для переименования, в отличие от ren, что отражает унифицированный подход Linux к файловым операциям. Параметры mv используют дефисы (например, -i), в отличие от слэшей ren.

Команда Windows fs не является стандартной; предположительно, имеется в виду fsutil или операции с файловой системой. В Linux для информации о файловой системе используется df (например, df -h для удобочитаемых размеров) или lsblk для блоковых устройств, в отличие от fsutil fsinfo. Для управления файловой системой используются tune2fs или mkfs, аналогичные функциям fsutil. Команды Linux используют короткие имена и опции с дефисами (например, df -h), в отличие от многословных подкоманд Windows (fsutil fsinfo drives).

Команда Windows del, удаляющая файлы, эквивалентна команде Linux rm. Например, rm file.txt удаляет файл, как del file.txt. Для удаления директорий rm требует -r (recursive), в отличие от del, который работает только с файлами. В rm используются -f для подавления запросов (как del /f) и -i для интерактивного удаления, с чувствительными к регистру именами файлов и опциями с дефисами, в отличие от нечувствительных к регистру путей и слэшей Windows.

Команда Windows dir, отображающая содержимое директории, соответствует команде Linux ls. Например, ls -l даёт подробный список, аналогичный dir. В ls есть опции -a (все файлы, включая скрытые) и -h (удобочитаемые размеры), схожие с dir /a или dir /s. Команда ls использует однобуквенные, чувствительные к регистру опции с дефисами (например, -l), тогда как dir использует слэши и подкоманды (например, /w для широкого формата), а пути в Linux чувствительны к регистру.

Команда Windows md (make directory) эквивалентна Linux mkdir. Обе создают директории (например, mkdir newdir или md newdir). В Linux mkdir поддерживает -p для создания родительских директорий (например, mkdir -p dir1/dir2), в отличие от md, который создаёт только один уровень без дополнительных команд. Опции mkdir используют дефисы и чувствительны к регистру, в отличие от простого, нечувствительного к регистру синтаксиса md.

Команда Windows rd (remove directory) соответствует Linux rmdir для пустых директорий (например, rmdir dir) или rm -r для непустых (например, rm -r dir). В отличие от rd /s, удаляющего непустые директории, rmdir ограничен пустыми, а rm -r более мощный, но требует осторожности. Linux использует опции с дефисами (например, -f для rm) и чувствительные к регистру пути, в отличие от подхода rd со слэшами.

Команда Windows systeminfo, отображающая системную информацию, соответствует нескольким командам Linux. Для общего обзора системы uname -a показывает ядро и имя хоста (аналог OS Name и Host Name в systeminfo). Для информации об оборудовании lscpu, lsmem или lshw показывают данные о процессоре и памяти, схожие с Processor и Total Physical Memory в systeminfo. Команда dmidecode (с sudo) предоставляет подробные данные о BIOS и оборудовании. Для информации о дисках используются df -h или lsblk. Linux разделяет функциональность systeminfo на несколько утилит, используя короткие имена команд и опции с дефисами (например, lscpu --all), в отличие от единой команды systeminfo с разделённым запятыми выводом.

Различия в именовании и параметрах заключаются в том, что команды Linux обычно короткие, в нижнем регистре и следуют Unix-конвенциям (например, cp, mv, ls), что обеспечивает быстрый ввод и модульность. Команды Windows, основанные на DOS, часто являются аббревиатурами (например, dir, del, ren) и нечувствительны к регистру, иногда более описательные. Параметры Linux используют одинарный (-r) или двойной (--recursive) дефис, чувствительны к регистру и требуют явного указания путей. Windows использует слэши (/s, /f), нечувствительные к регистру, с необязательными параметрами и подразумеваемыми значениями по умолчанию. Linux предпочитает модульность, комбинируя утилиты (например, find с cp для задач xcopy), тогда как Windows интегрирует больше функций в одну команду. Чувствительность к регистру в Linux распространяется на пути и аргументы, в отличие от Windows, а Linux использует прямые слэши (/), а не обратные ().

В Linux Mint эти команды являются частью стандартных пакетов (coreutils для cp, mv, ls, mkdir, rm, rmdir; util-linux для df, lsblk; procps для uname).

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))

Вывести имя текущего пользователя, текущие дату и время, имя хоста, используя системные переменные:

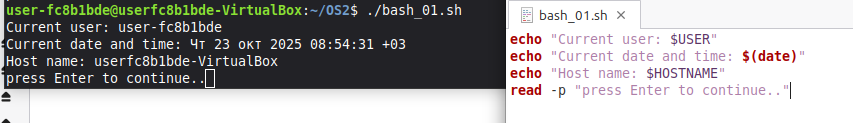


Рисунок 1 – bash\_01

Вывести имя текущего скрипта, время последней модификации и полный путь к скрипту, используя расширения параметров:

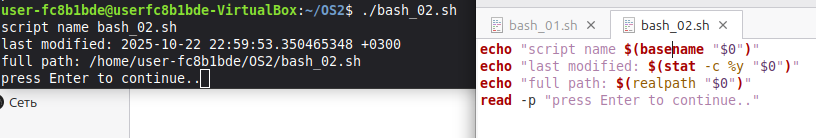


Рисунок 2 –bash\_02

Вывести строку со всеми переданными параметрами и первые четыре параметра по отдельности:

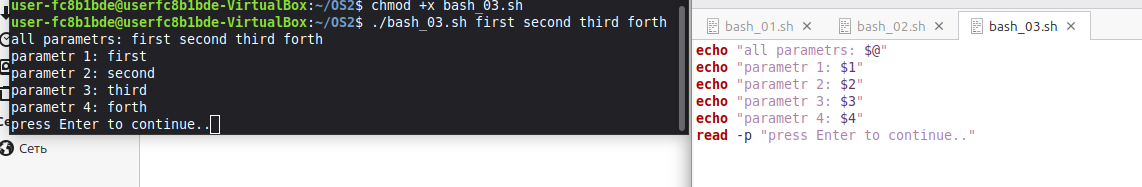


Рисунок 3 – bash\_03

Вычислить выражения a-b, a+b, c/b, c\*b, (a-b)\*(b-a), где a, b, c — первый, второй и третий параметры:

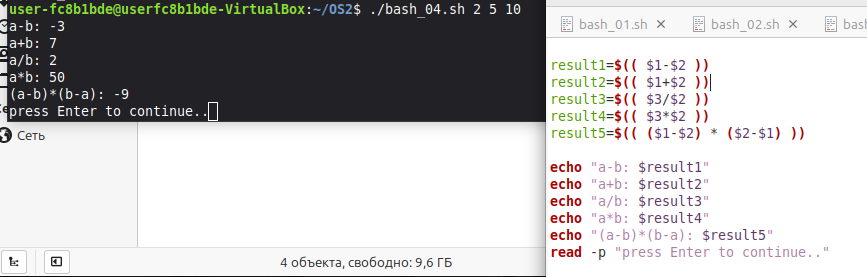


Рисунок 4 – bash\_04

Скрипт вызывается как bash\_05.sh {режим} {имя\_файла}, поддерживает режимы create и delete, с соответствующими проверками и сообщениями:

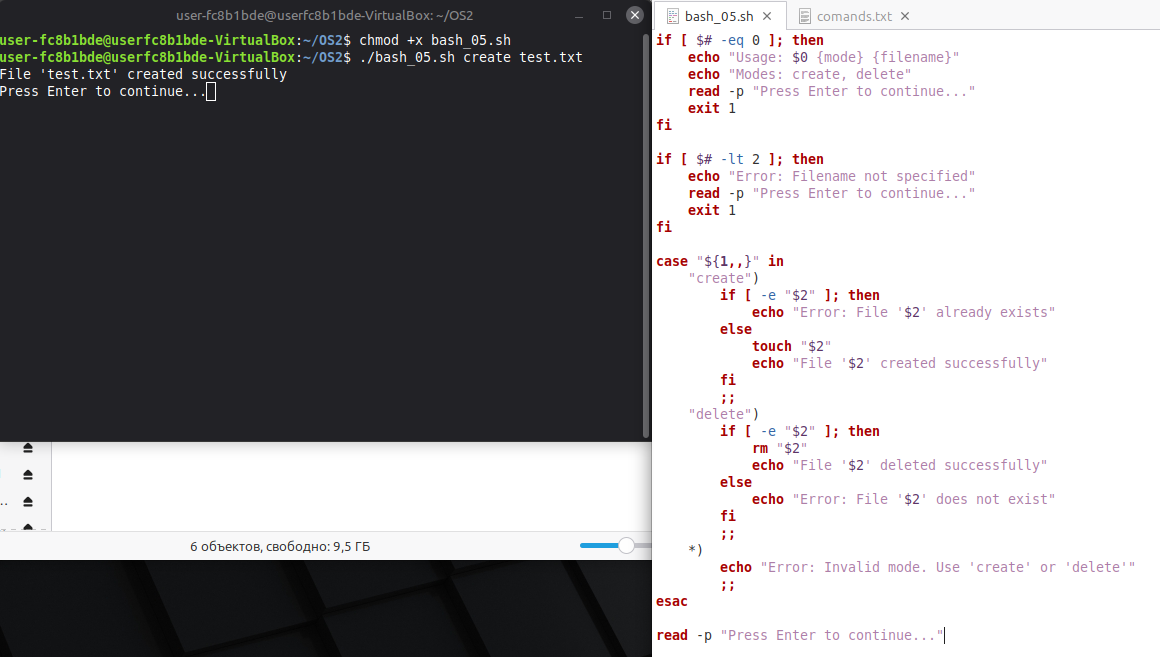


Рисунок 5 – bash\_05

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

Переменные окружения в Linux — это пары ключ-значение, содержащие конфигурационные данные, которые операционная система, приложения и оболочка, такая как bash, используют для настройки своего поведения. Они хранят информацию, например, пути к исполняемым файлам, домашнюю директорию пользователя или языковые настройки, и доступны процессам в текущей сессии или глобально. Переменные могут быть временными, действующими только в текущей оболочке, или постоянными, заданными в конфигурационных файлах, таких как ~/.bashrc или /etc/environment. Их основная роль заключается в обеспечении гибкости и переносимости программ, позволяя им адаптироваться к системе без жёстко прописанных значений. Переменные окружения используются для указания директорий поиска команд, настройки приложений, предоставления системной информации и поддержки скриптов, упрощая автоматизацию задач.

В Linux переменные окружения хранятся в памяти текущей сессии оболочки и передаются дочерним процессам. Временные переменные задаются в текущей оболочке и исчезают при её закрытии, а постоянные сохраняются в текстовых файлах, таких как ~/.bashrc для пользователя или /etc/environment для всей системы. Это делает редактирование переменных простым и удобным для скриптов. Доступ к переменным осуществляется через синтаксис $ИМЯ, они чувствительны к регистру, а в PATH директории разделяются двоеточиями. Управление осуществляется командами, такими как export для установки, printenv или env для просмотра и unset для удаления. Каждая оболочка, например bash или zsh, имеет свои конфигурационные файлы, что даёт гибкость, но может вызывать путаницу при выборе правильного файла для постоянных изменений.

В Windows переменные окружения также хранят конфигурационные данные, но они сохраняются в реестре, в ветках HKEY\_LOCAL\_MACHINE для системных и HKEY\_CURRENT\_USER для пользовательских переменных. Это делает их редактирование менее удобным для скриптов, так как требуется доступ через графический интерфейс или команду setx, которая применяет изменения только в новых сессиях. В командной строке Windows переменные вызываются через %ИМЯ% или $Env:ИМЯ в PowerShell, они нечувствительны к регистру, а в PATH директории разделяются точками с запятой. Временные переменные, заданные через set, действуют в текущей сессии и передаются дочерним процессам, но постоянные изменения требуют перезапуска консоли.

Основные отличия заключаются в способе хранения: Linux использует текстовые файлы, что упрощает автоматизацию, тогда как Windows полагается на реестр, что менее гибко. Linux чувствителен к регистру, а Windows — нет. Разделитель в PATH в Linux — двоеточие, в Windows — точка с запятой. Linux требует явного экспорта переменных для передачи дочерним процессам, в то время как в Windows переменные, заданные через set, автоматически наследуются в текущей сессии. Linux разделяет конфигурацию по оболочкам, что даёт больше контроля, но требует понимания их различий, тогда как Windows использует единый реестр, упрощая системное управление, но ограничивая гибкость. Команды управления в Linux, такие как export и unset, встроены в оболочку и проще в использовании, чем set и setx в Windows, которые менее интуитивны из-за разделения на временные и постоянные изменения.

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

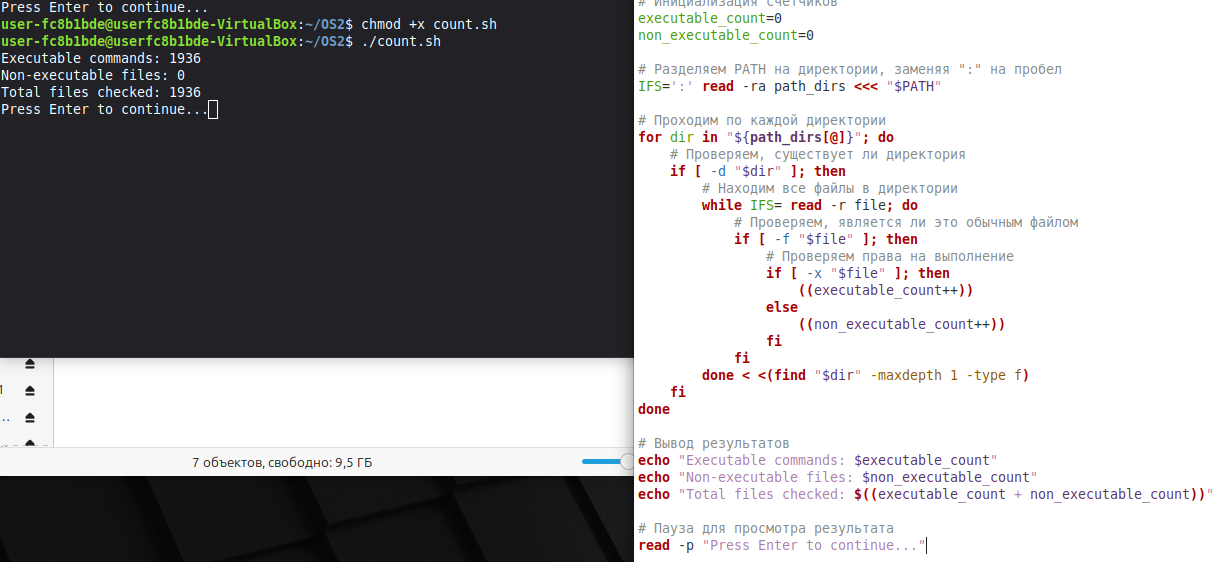


Рисунок 6 – Исполняемый код

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).

|  |
| --- |
| if [ $# -eq 0 ]; then  echo "Usage: $0 <directory\_path>"  echo "Example: $0 /home/user/documents"  read -p "Press Enter to continue..."  exit 1  fi  if [ ! -d "$1" ]; then  echo "Error: Directory '$1' does not exist"  read -p "Press Enter to continue..."  exit 1  fi  dir\_path=$(realpath "$1")  echo "Listing contents of: $dir\_path"  echo "--------------------------------"  convert\_size() {  local size\_bytes=$1  local size  local unit  if [ $size\_bytes -lt 1024 ]; then  size=$size\_bytes  unit="B"  elif [ $size\_bytes -lt $((1024 \* 1024)) ]; then  size=$(echo "scale=2; $size\_bytes / 1024" | bc)  unit="KB"  elif [ $size\_bytes -lt $((1024 \* 1024 \* 1024)) ]; then  size=$(echo "scale=2; $size\_bytes / (1024 \* 1024)" | bc)  unit="MB"  else  size=$(echo "scale=2; $size\_bytes / (1024 \* 1024 \* 1024)" | bc)  unit="GB"  fi  size=$(echo "$size" | sed 's/\.00$//')  echo "$size $unit"  }  while IFS= read -r item; do  if [ "$item" = "$dir\_path" ]; then  continue  fi  rel\_path=${item#"$dir\_path/"}  if [ -d "$item" ]; then  file\_count=$(find "$item" -type f | wc -l)  echo "Directory: $rel\_path ($file\_count files)"  elif [ -f "$item" ]; then  # Получаем размер файла в байтах  size\_bytes=$(stat -c %s "$item" 2>/dev/null)  if [ $? -eq 0 ]; then  # Конвертируем размер  size\_formatted=$(convert\_size $size\_bytes)  echo "File: $rel\_path ($size\_formatted)"  else  echo "File: $rel\_path (size unavailable)"  fi  fi  done < <(find "$dir\_path" -print)  read -p "Press Enter to continue..." |

Листинг 1 – Код исполняемый

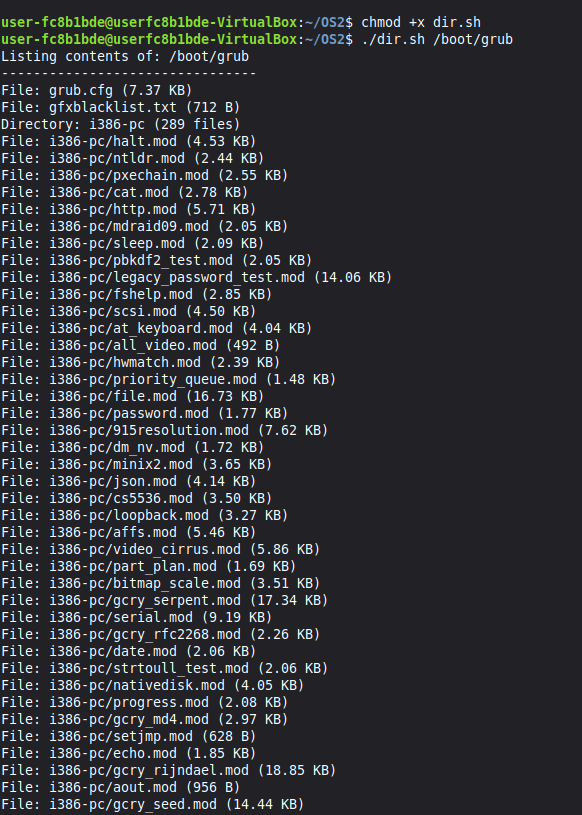


Рисунок 7- Вывод

**Заключение**

В ходе лабораторной работы по теме «Знакомство с командной оболочкой операционной системы» были изучены и применены основы работы с командной оболочкой в Linux Mint. Разработаны пять bash-скриптов, аналогичных bat-файлам Windows, для вывода системной информации, обработки параметров, выполнения арифметических операций и управления файлами, с использованием конструкций bash, таких как subshell ($( )) и case. Исследованы переменные окружения, особенно PATH, подчёркивающая гибкость Linux благодаря текстовым конфигурационным файлам в отличие от реестра Windows. Скрипты для подсчёта команд в PATH и анализа структуры каталогов с размерами файлов успешно реализовали требуемый функционал. Работа подтвердила практическую значимость изучения командной оболочки для автоматизации задач в Linux Mint.