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 6 группы

Филипюк Илья Андреевич

Преподаватель: асс. Уласевич Н.И.

2025, Минск

# **Введение**

Командная оболочка – это программа, которая предоставляет текстовый интерфейс для взаимодействия пользователя с операционной системой. Она принимает команды, введённые с клавиатуры, выполняет их или запускает программы, и отображает результаты в командной строке (терминале). Оболочки также позволяют создавать скрипты – файлы с последовательностью команд для автоматизации задач, и обладают функциями программирования.

Цель работы: получить практические навыки в работе с командными оболочками такими как cmd, PowerShell, bash.

Задачи исследования:

* рассмотреть основные команды, используемые в cmd;
* рассмотреть основные команды, используемые в bash;
* рассмотреть основные команды, используемые в Powershell для написания собственных скриптов;
* выполнить задания, используя рассмотренные команды;
* узнать, что такое bat-файлы, для чего они нужны и научиться создавать их;
* узнать, что такое sh-файлы, для чего они нужны и научиться создавать их;

Используемые инструменты:

* cmd;
* bash;
* PowerShell;
* Microsoft Hyper-V;

# **1. Изучение основ cmd в Windows**

В данном разделе рассматриваются основные команды cmd, часто используемые на практике, а также выполнение заданий лабораторной работы для закрепления материала.

# **1.1 Предварительная подготовка**

Перед изучением команд в домашнем каталоге пользователя User-ee4aed31 был создан каталог OS2. На рисунке 1.1 показан результат создания данной директории.

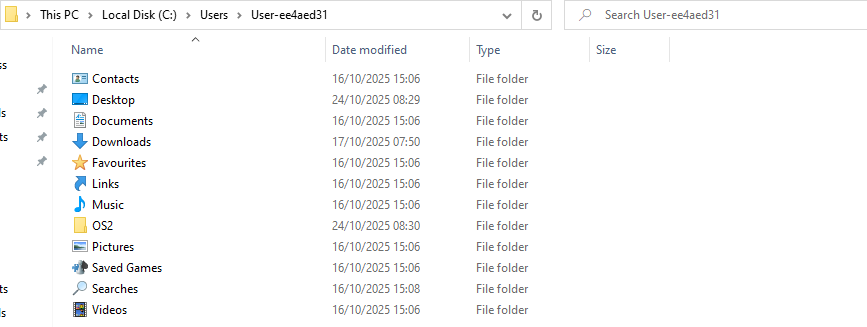


Рисунок 1.1 – Создание директории OS2 в домашнем каталоге пользователя User-ee4aed31

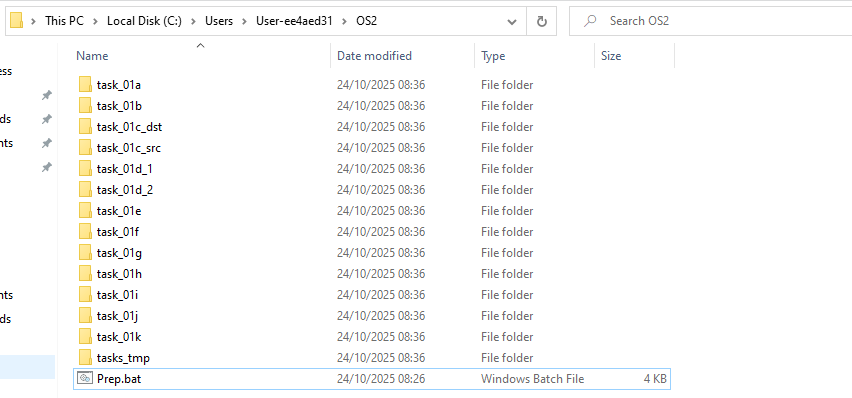
Затем в нем был выполнен файл Prep.bat для формирования структуры папок, необходимой для выполнения заданий. На рисунке 1.2 представлен результат выполнения Prep.bat файла.

Рисунок 1.2 – Результат выполнения файла Prep.bat в директории OS2

Затем был открыт cmd в данном каталоге путём указания слова cmd в адресной строке проводника (рисунок 1.3)

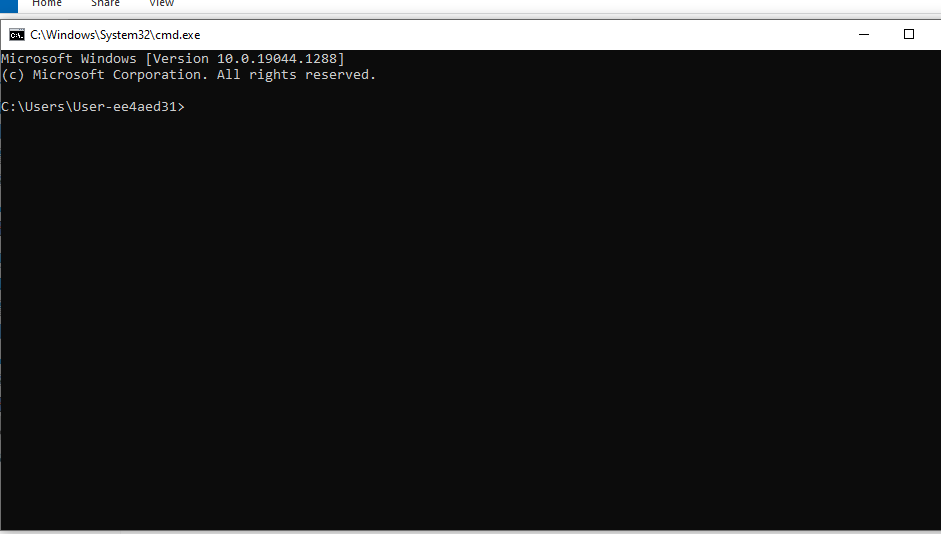


Рисунок 1.3 – Результат открытия cmd в директории OS2

# **1.2 Команда cd**

Команда cd (change directory) предназначена для изменения текущего рабочего каталога. Она поддерживает абсолютные и относительные пути, а также специальные обозначения:

* .. – переход в родительский каталог;
* . – текущий каталог;
* \ – переход к корню диска.

Для работы с путями, содержащими пробелы, рекомендуется заключать их в двойные кавычки. Ключ /d позволяет переключаться между дисками.

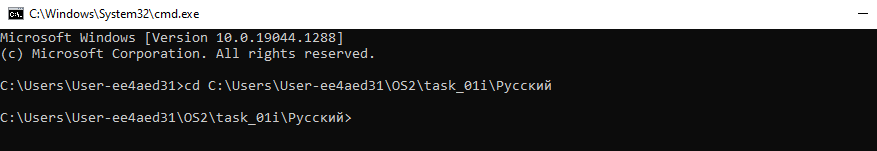
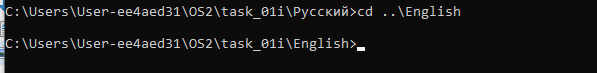


Рисунок 1.4 – Результат выполнения команды cd с указанием абсолютного пути

Рисунок 1.5 – Результат выполнения команды cd с указание относительного пути

# **1.3 Команда copy**

Команда сору выполняет копирование файлов. Ее синтаксис: сору <источник> <назначение>. Поддерживаются шаблоны, например, \*.txt (Рис. 1.6).

Основные ключи:

* /A (копирование текстовых файлов);
* /V (проверка целостности данных);
* /Y и /-Y (автоматический ответ перезаписи или запрос);
* /Z (возобновление копирования после сетевого сбоя).

В случае, если выполнение команды потребует перезаписи существующих файлов в папке назначения и не будет указан ключ /Y, то команда запросит подтверждение перезаписи необходимых файлов (Рис. 1.7).

Для копирования с изменением имени файла следует указать новое имя в параметре назначения (Рис. 1.8).

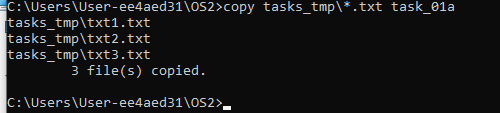


Рисунок 1.6 – Результат выполнения команды copy с шаблоном файла

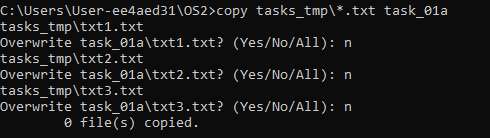
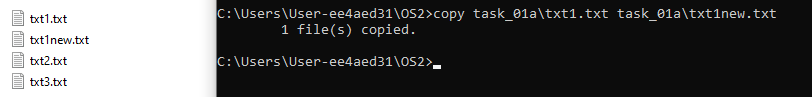
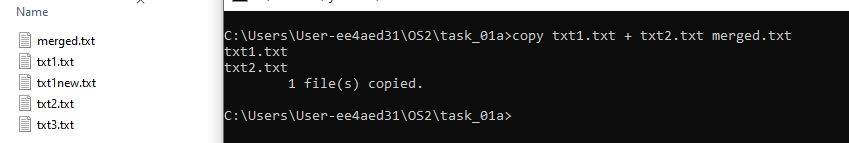
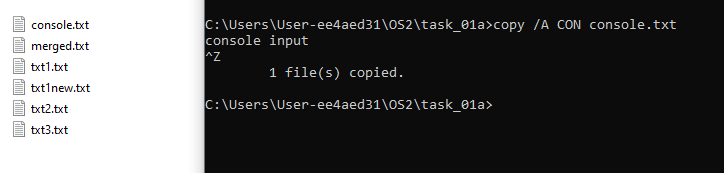


Рисунок 1.7 – Запрос подтверждения на перезапись файлов при повторном выполнении команды copy

Рисунок 1.8 – Результат выполнения команды copy со сменой имени файла

Команда copy также позволяет объединить файлы с помощью символа + (Рис. 1.9) и использовать консоль (CON) как источник или приёмник данных (Рис. 1.10 – Рис. 1.11).

Риснок 1.9 – Результат объединения файлов с помощью команды copy

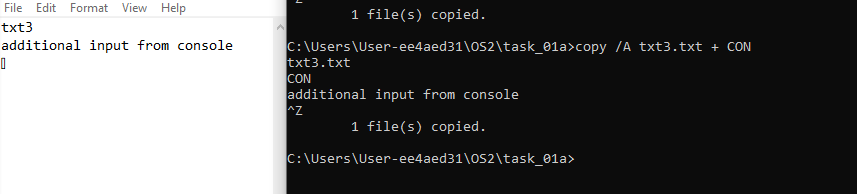
Рисунок 1.10 – Результат выполнения команды copy, где в качестве источника данных выступает консоль

Рисунок 1.11 – Результат выполнения команды copy для записи дополнительного текста, вводимого с консоли, в файл

# **1.3 Команда xcopy**

Команда xcopy предоставляет расширенные возможности копирования. Основные ключи:

* /S (копирование непустых папок и подпапок);
* /E (копирование всех папок, включая пустые);
* /H (включение скрытых и системных файлов);
* /A (копирование файлов с атрибутом «архивный»);
* /L (предварительный просмотр файлов для копирования).

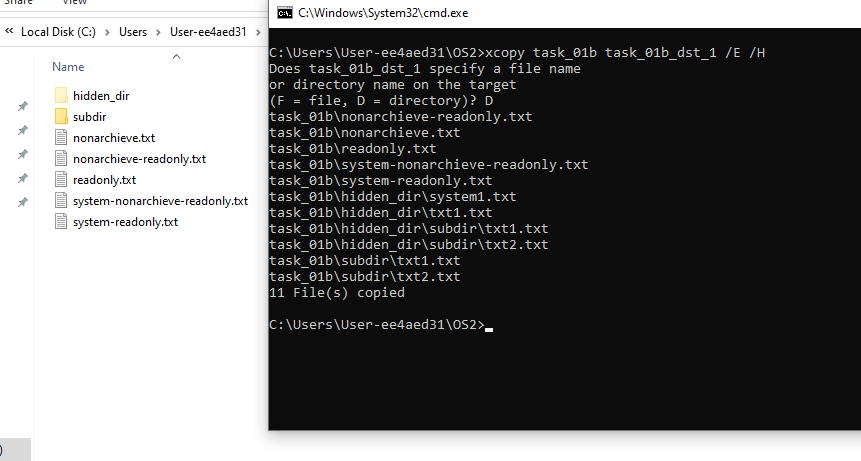
Примеры использования с ключами /H /E и /A /K представлены на Рис. 1.12 и 1.13 соответственно.

Рисунок 1.12 - Результат выполнения команды xcopy с использованием ключей /H и /E

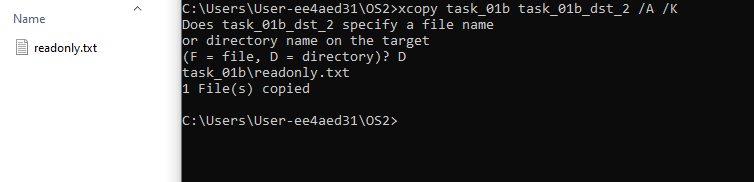


Рисунок 1.13 – Результат выполнения команды xcopy с использованием ключей /A и /K

# **1.4 Команда move**

Команда move предназначена для перемещения или переименования файлов и папок. Поддерживает шаблоны, например, \*.txt (Рис. 1.14). Для очистки каталога можно использовать цикл (Рис. 1.15).

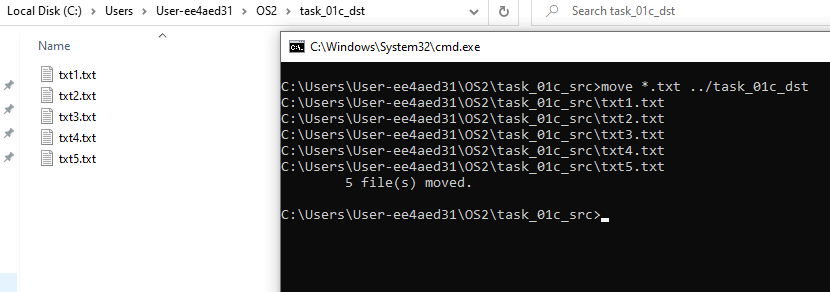
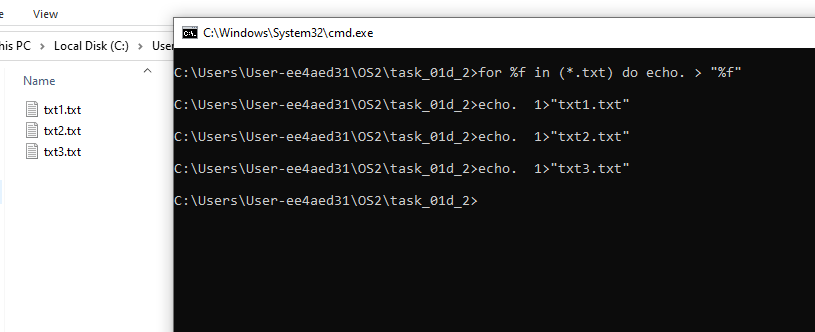
Рисунок 1.14 – Результат выполнения команды move с применением шаблона

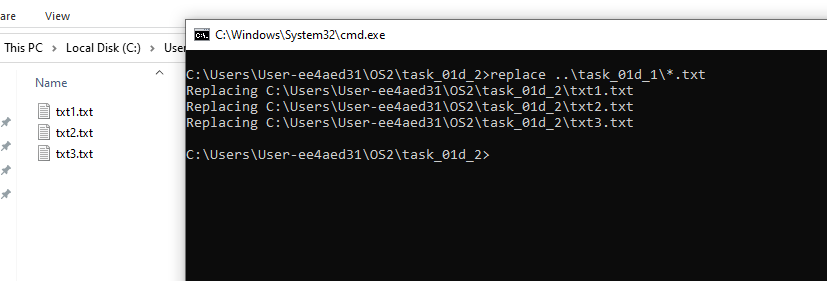
Рисунок 1.15 – Результат выполнения команды очистки файлов

# **1.5 Команда replace**

Команда replace обновляет или добавляет файлы (Рис. 1.16).

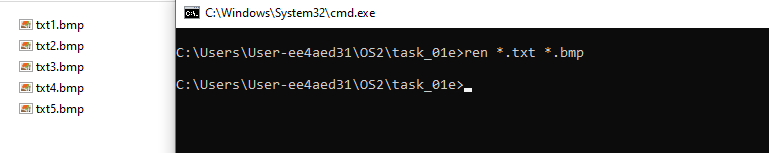
Основные ключи:

* /A (добавление отсутствующих файлов);
* /U (обновление только устаревших файлов);
* /S (рекурсивный поиск и замена).

Рисунок 1.16 – Результат выполнения команды replace

# **1.6 Команда ren**

Команда ren изменяет имена файлов и каталогов. Поддерживает шаблоны, например, смена расширений \*.txt на \*.bmp (Рис. 1.17). Также позволяет одновременно менять имя и расширение (Рис. 1.18).



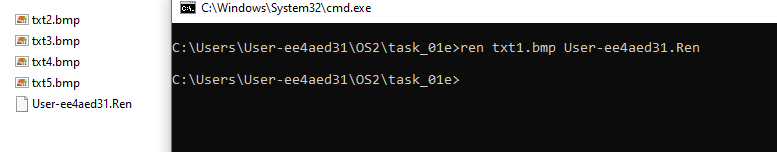
Рисунок 1.17 – Результат выполнения команды ren для смены расширений файлов

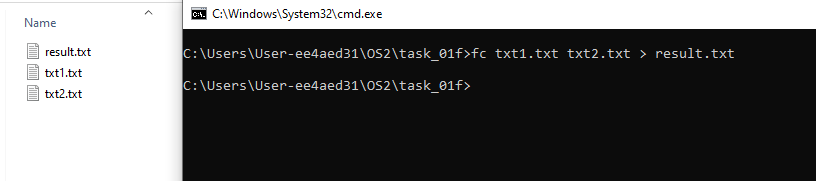
Рисунок 1.18 – Результат выполнения команды ren для смены имени и расширения файла одновременно

# **1.7 Команда fc**

Команда fc сравнивает содержимое файлов (Рис. 1.19).

Основные ключи:

* /A (сокращенный вывод различий);
* /B (двоичное сравнение);
* /L (текстовый режим);
* /N (отображение номеров строк).

Рисунок 1.19 – Результат выполнения команды fc для сравнения двух файлов

# **1.8 Команда del (erase)**

Команда del удаляет файлы (Рис. 1.20 — 1.23).

Основные ключи:

* /P (запрос подтверждения для каждого файла);
* /Q (отключение запросов);
* /S (рекурсивное удаление в подпапках);
* /A (фильтрация по атрибутам).

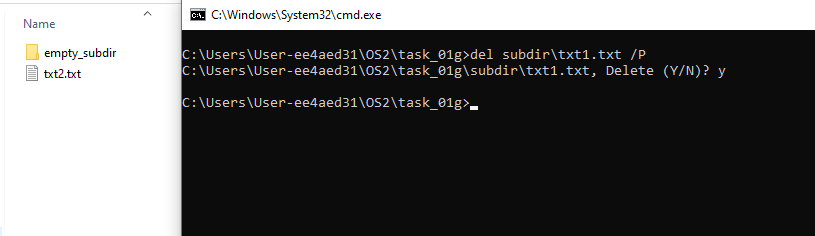


Рисунок 1.20 – Результат выполнения команды del для удаления файла с запросом подтверждения

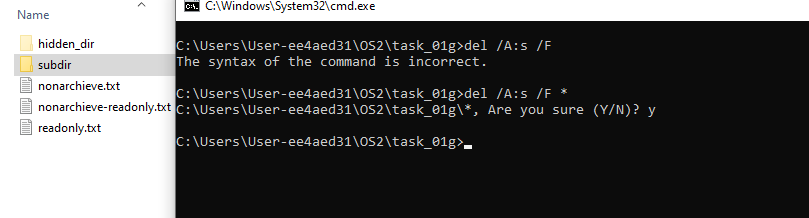
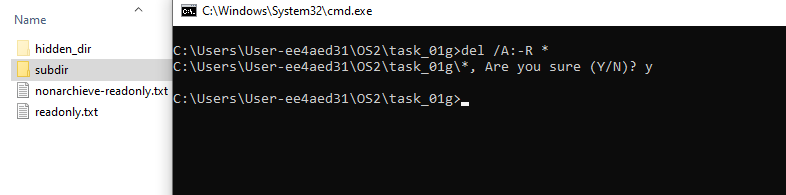


Рисунок 1.21 – Результат выполнения команды del для удаления всех файлов с атрибутом «Системный»

Рисунок 1.22 – Результат выполнения команды del для удаления всех файлов кроме файлов с атрибутом «Только для чтения»

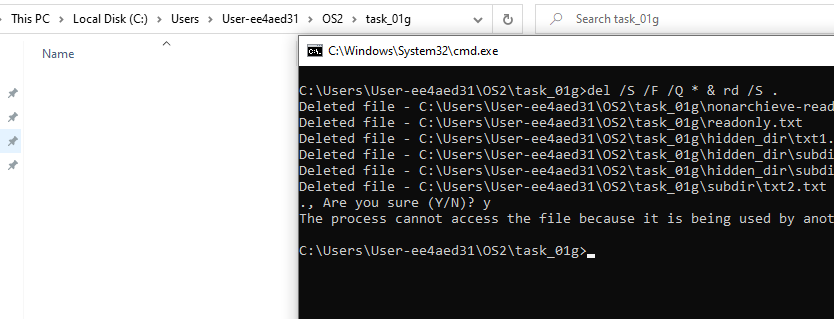


Рисунок 1.23 – Результат выполнения команды del совместно с rd для удаления всех оставшихся файлов и дочерних каталогов

# **1.9 Команда dir**

Команда dir выводит содержимое каталога (Рис. 1.24—1.25).

Основные ключи:

* /A (фильтрация по атрибутам);
* /O (сортировка);
* /S (рекурсивный обход подкаталогов);
* /P (постраничный вывод).

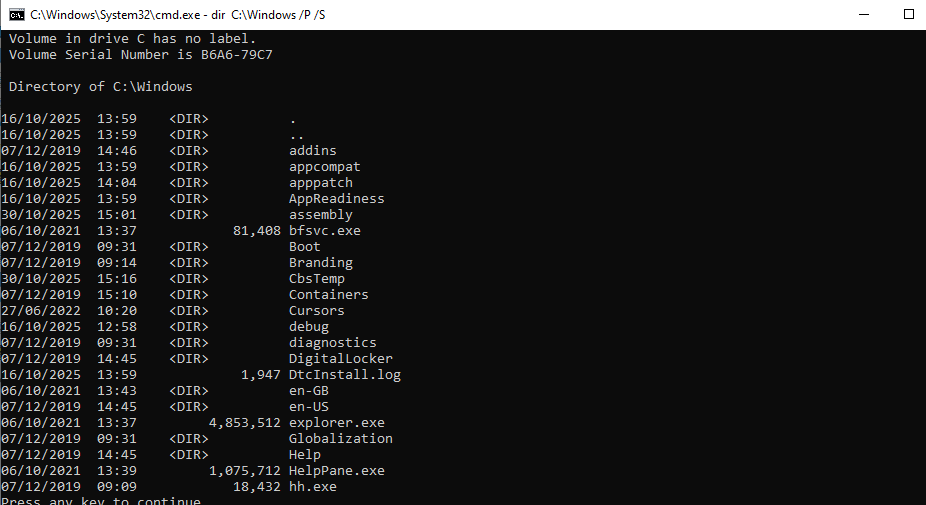
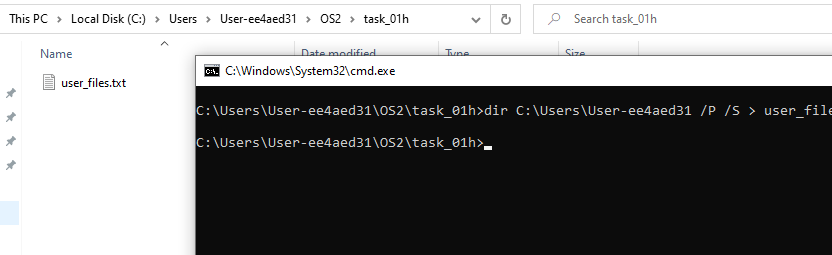
Рисунок 1.24 – Результат выполнения команды dir с ключами /P /S для постраничного вывода содержимого каталога

Рисунок 1.25 – Результат выполнения команды dir с ключами /P /S и с записью в файл

# **1.10 Команда md**

Команда md создает каталоги, включая промежуточные, если они отсутствуют (Рис. 1.26–1.27).

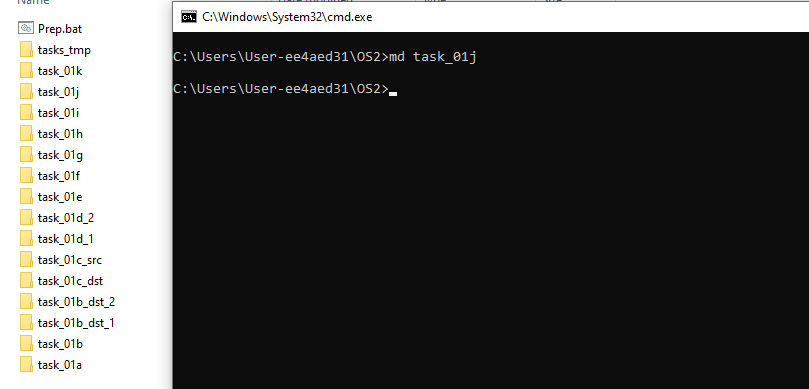
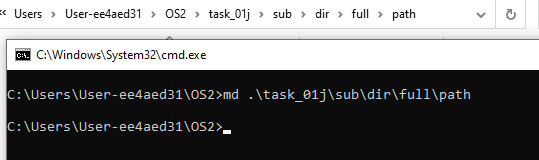
Рисунок 1.26 – Результат выполнения команды md для создания каталога

Рисунок 1.27 – Результат выполнения команды md для создания иерархии папок в предварительно созданном каталоге

# **1.11 Команда rd**

Команда rd удаляет каталоги (Рис. 1.28—1.29).

Основные ключи:

* /S (удаление дерева каталогов);
* /Q (отключение подтверждения);

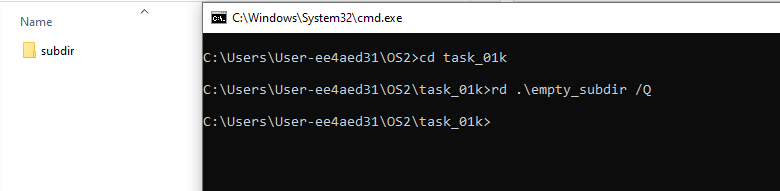
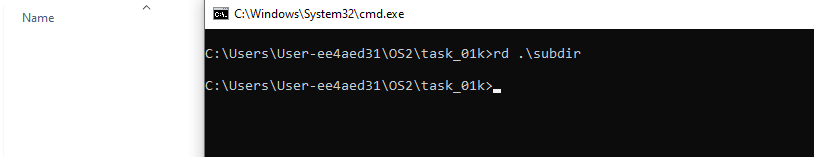


Рисунок 1.28 – Результат выполнения команды rd для удаления пустого каталога

Рисунок 1.29 – Результат выполнения команды rd для удаления непустого каталога

# **1.12 Команда type**

Данная команда предназначена для просмотра содержимого файлов прямо в командной строке. В качестве параметра данная команда принимает название (путь) файла.

# **1.13 Команда systeminfo**

Данная команда предназначена для получения детальной информации о системе, оборудовании и конфигурации Windows. Параметров она не принимает, но у нее есть дополнительные ключи. Вот некоторые мы сейчас и рассмотрим.

Основные ключи:

* /S (удаленный компьютер);
* /FO (формат вывода);
* /U (контекста пользователя);
* /P (пароль пользователя);
* /NH (игнорирование заголовков);

# **1.14 Команда findstr**

Данная команда предназначена для поиска текста в файлах с поддержкой регулярных выражений.

Основные ключи:

* /B (поиск начала по заданному шаблону);
* /I (поиск конца по заданному шаблону);
* /L (использовать шаблон как литерал);
* /I (игнорировать регистр);
* /S (рекурсивный поиск);
* /V (поиск строк, не содержащих шаблон);
* /N (показ номеров строк);
* /M (только полные совпадения);

# **1.15 Операторы**

Для расширения функциональности командной строки используются специальные операторы.

Операторы перенаправления ввода/вывода:

* > – перенаправление вывода с перезаписью файла.
* >> – перенаправление вывода с добавлением в конец файла.

Оператор конвейера (|) передает вывод одной команды на вход другой, позволяя объединять команды в цепочки.

Операторы управления выполнением:

* & – последовательное выполнение команд.
* && – условное выполнение (следующая команда выполняется только при успехе предыдущей).
* || – условное выполнение (следующая команда выполняется только при ошибке предыдущей).
* () – группировка команд.

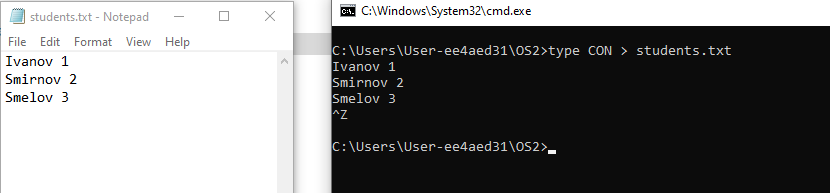


Рисунок 1.30 – Результат выполнения команды type с использованием оператора перенаправления

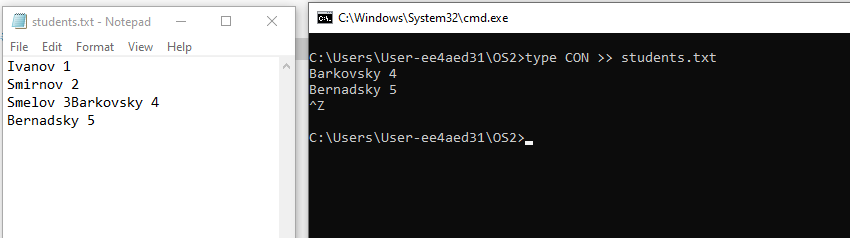
Рисунок 1.31 - Результат выполнения команды findstr с использованием оператора конвейеризации

Рисунок 1.32 – Результат выполнения команды type с использованием оператора перенаправления для записи в конец файла

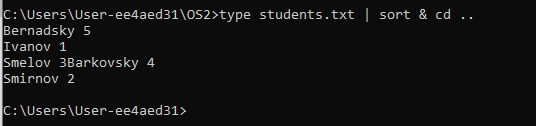
Для вывода в отсортированном порядке нам понадобятся команды type для вывода, sort для сортировки и cd для выхода из папки, а также оператор конвейеризации для того, чтобы вывод type передать команде сортировки, а также оператор соединения команд, чтобы затем еще выполнить команду выхода из папки.

Рисунок 1.33 – Результат выполнения совокупности команд type, sort, cd и операторов | и &

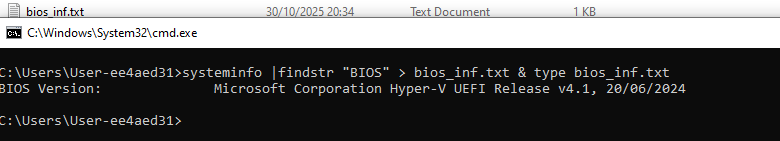
Ну и последним заданием нам требуется получить версию BIOS через файл. Для получения версии BIOS через файл нам понадобится команда systeminfo, чтобы получить всю информацию о системе, команда findstr “BIOS” и команда type для вывода результата на консоль. Используем оператор конвейера, чтобы передать результата вывода информации от systeminfo в findstr, затем оператор перенаправления для записи найденной информации в файл, потом оператор соединения команд, чтобы потом еще вывести эту информацию.

Рисунок 1.34 – Результат выполнения совокупности команд systeminfo, findstr, type и операторов |, & и >

# **1.16 Пакетные файлы (BAT-файлы)**

Пакетный файл (BAT-файл) – это текстовый файл с расширением .bat, содержащий последовательность команд для выполнения в командной строке Windows. Основное назначением является автоматизация рутинных операций.

# **1.17 Системные переменные**

Системные переменные хранят данные о конфигурации системы и пользовательском окружении.

Основные группы переменных:

* системные пути: %SystemRoot%, %ProgramFiles%;
* пользовательские пути: %USERPROFILE%, %APPDATA%, %TEMP%;
* системная информация: %COMPUTERNAME%, %USERNAME%, %OS%.

Для просмотра всех переменных используется команда set. Ключевой является переменная %PATH%, содержащая список каталогов для поиска исполняемых файлов.

# **1.18 Параметры и их расширения**

В BAT-файлах параметры передаются через аргументы командной строки: %0 – имя скрипта, %1-%9 – аргументы, %\* – все аргументы.

Расширения параметров позволяют модифицировать значения преимущественно пути:

* %~f1 (полный путь);
* %~d1 (буква диска);
* %~p1 (путь без имени файла);
* %~n1 (имя файла без расширения);
* %~x1 (расширение файла);
* %~t1 (дата и время изменения);
* %~z1 (размер файла).

# **1.19 Правила именования команд и параметров**

Названия команд чаще всего формируются по следующим принципам:

* аббревиатура: cd (Change Directory), fc (File Compare);
* усечение: del (delete), ren (rename);
* комбинация: xcopy (extended copy), ipconfig (IP configuration);
* прямое указание действия: copy, sort, find.

Параметры команд, как правило, представляют собой одну или несколько букв, указывающих на их функцию: /S (Subdirectories), /Q (Quiet), /A (Attributes).

# **1.20 Написание собственных скриптов**

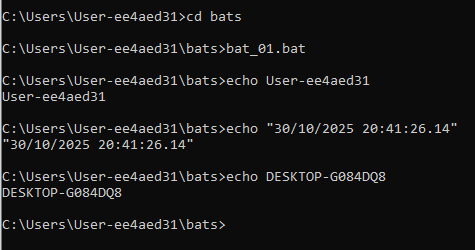
Создаем наш первый скрипт, используя системные переменные для получения данных. В листинге 1.1 представлены команды для вывода требуемой информации. На рисунке 1.35 продемонстрирован результат выполнения.

echo %USERNAME%

echo "%DATE% %TIME%"

echo %COMPUTERNAME%

Листинг 1.1 – Содержимое файла bat\_01.bat

Рисунок 1.35 – Результат выполнения bat\_01.bat

Следующий скрипт выводит метаданные о самом скрипте при помощи расширения параметров. В листинге 1.2 представлены команды, которые как раз и позволяют нам вывести нужную информацию для второго задания. На рисунке 1.36 продемонстрирован результат выполнения.

|  |
| --- |
| echo %~nx0  echo %~t0  echo %~f0 |

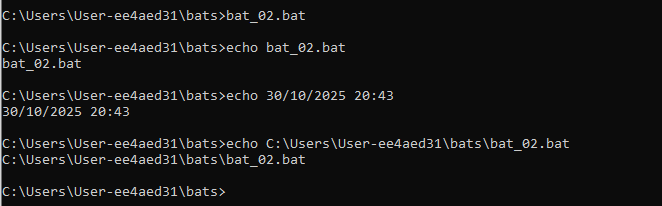
Листинг 1.2 – Содержимое файла bat\_02.bat

Рисунок 1.36 – Результат выполнения файла bat\_02.bat

В третьем задании необходимо работать с параметрами, передаваемые скрипту. В листинге 1.3 представлены команды для вывода наших параметров заданным образом. На рисунке 1.37 продемонстрирован результат выполнения.

|  |
| --- |
| echo %\*  echo %1  echo %2  echo %3  echo %4 |

Листинг 1.3 Содержимое bat-файла bat\_03.bat

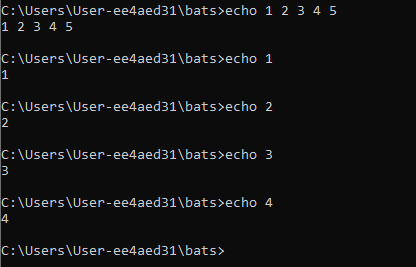


Рисунок 1.37 – Результат выполнения файла bat\_03.bat

В четвертом задании требуется вычислить заданные арифметические выражения. Для этого мы будем использовать команду set с ключом /a, которая позволяет нам задавать значения переменным. В листинге 1.4 представлены команды для выполнения данного задания. На рисунке 1.38 продемонстрирован результат выполнения.

|  |
| --- |
| set /a a=%1  set /a b=%2  set /a c=%3  set /a res1=a-b  echo %a% - %b% = %res1%  set /a res2=a+b  echo %a% + %b% = %res2%  if %b% equ 0 (     echo "Error: attempt to divide by zero!"  ) else (     set /a res3=c/b     echo %c% / %b% = %res3%  )  set /a res4=c\*b     echo %c% \* %b% = %res4%  set /a res5=(a-b)\*(b-a)  echo (%a% - %b%) \* (%b% - %a%) = %res5% |

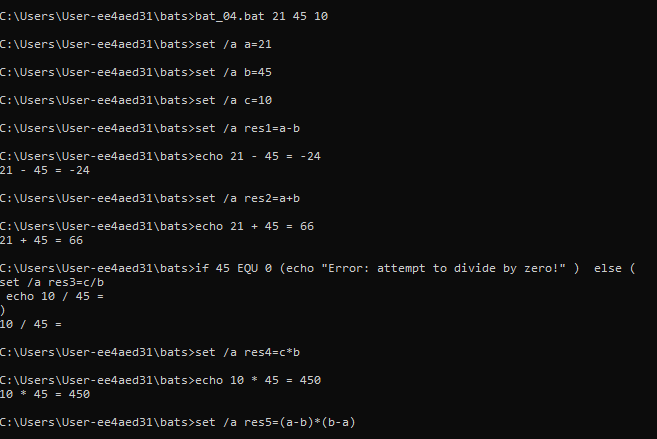
Листинг 1.4 Содержимое bat-файла bat\_04.bat

Рисунок 1.38 – Результат выполнения файла bat\_04.bat

В последнем скрипте требуется написать программу для создания или удаления файла с проверкой корректности аргументов и существования файлов с выводом сообщений при неправильном вводе. В листинге 1.5 представлен полный скрипт для данного задания. На рисунке 1.39 продемонстрирован результат выполнения.

|  |
| --- |
| if "%1" == "" if "%2" == "" (echo "Help: set first param to work mode (create, delete), second to a file name."  goto :eof)  if "%1" == "" (echo "Error: work mode param is missing"  goto :eof)  if not "%1" == "create" if not "%1" == "delete" (echo "Help: wrong work mode input(try create/delete)"  goto :eof)  if "%2" == "" (echo "Error: file name is missing"  goto :eof)  if "%1" == "create" if exist "%2" (echo "Error: file already exists"  goto :eof) else (echo. > "%2"  goto :eof)  if "%1" == "delete" if not exist "%2" (echo "Error: no file was found"  goto :eof) else (del "%2") |

Листинг 1.5 Содержимое bat-файла bat\_05.bat

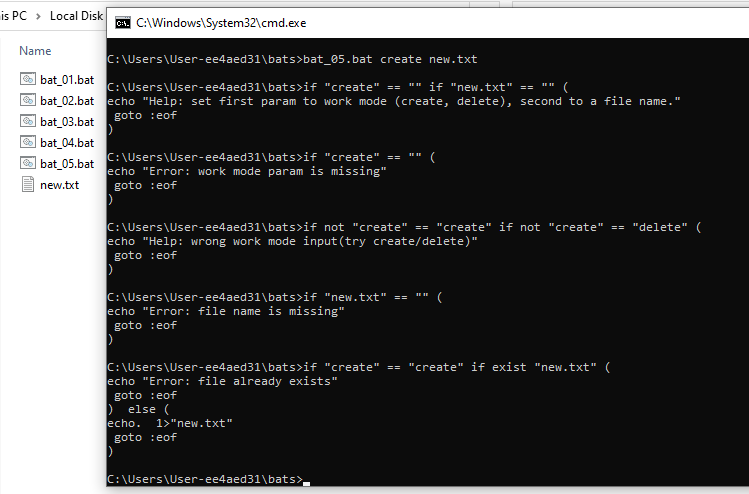


Рисунок 1.39 – Результат выполнения файла bat\_05.bat

# **2. Некоторые основы PowerShell и написание скриптов**

PowerShell – это современная кроссплатформенная оболочка командной строки и язык сценариев, разработанный Microsoft. В отличие от традиционной командной строки (cmd), PowerShell предлагает объектно-ориентированный подход и глубокую интеграцию с платформой .NET.

Ключевые характеристики:

* объектно-ориентированность: Команды возвращают объекты .NET, а не текст, что упрощает обработку и передачу данных;
* кроссплатформенность: работает в операционных системах Windows, Linux и macOS;
* мощный язык сценариев: предоставляет возможности для автоматизации сложных задач;
* интеграция с .NET: позволяет использовать всю мощь библиотек .NET Framework.

# **2.1 Написание собственных скриптов**

В качестве первого задания нам было необходимо написать интерактивный калькулятор для вычисления математических выражений. В приложении А представлен скрипт, написанный для выполнения требуемых задач.

Основные функции:

* Input-Number – обрабатывает вводимые в консоль данные;
* Read-Number – обрабатывает вводимые в консоль числовые данные;
* Read-Int – обрабатывает вводимые в консоль целочисленные данные;
* To-Radians – преобразует введённые данные из градусов в радианы;
* From-Radians – преобразует введённые данные из радиан в градусы;
* Show-Menu – выводит меню доступных операций;
* Do-Trigonometry – обрабатывает введённые тригонометрические функции и проводит вычисления;
* Do-Logarythm – обрабатывает введённые логарифмические функции и проводит вычисления;

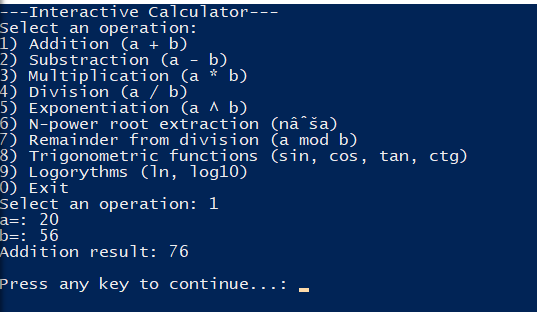
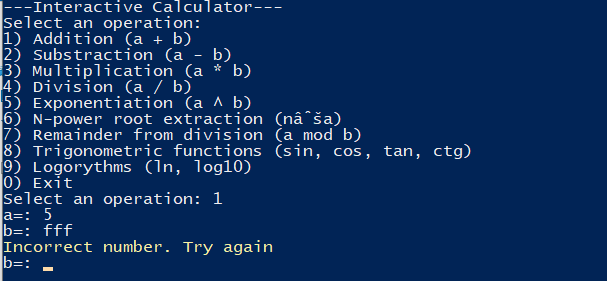
Для обработки ошибок используем структуру try-catch.

Рисунок 2.1 – Результат выполнения скрипта калькулятора

Теперь проверим различные сценарии, которые могут привести к ошибкам.

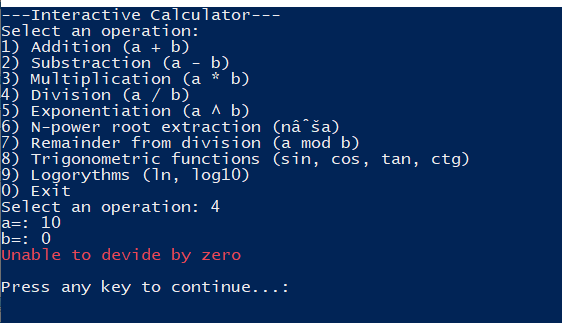
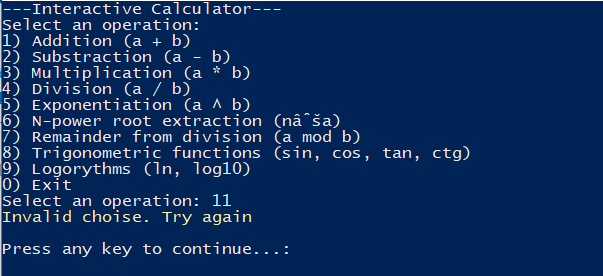
Рисунок 2.2 – Тестирование скрипта с некорректным вводом

Рисунок 2.3 – Тестирование скрипта с делением на ноль

Рисунок 2.4 – Тестирование скрипта с вводом неверного номера операции

Для следующего задания была разработана программа по преобразованию вводимого показателя температуры в иные системы счисления. В приложении Б представлен скрипт, написанный для выполнения требуемых задач.

Основные функции:

* Parse-TempInput– обрабатывает вводимые в консоль данные;
* Convert-Temperature – преобразовывает данные в необходимый вид;

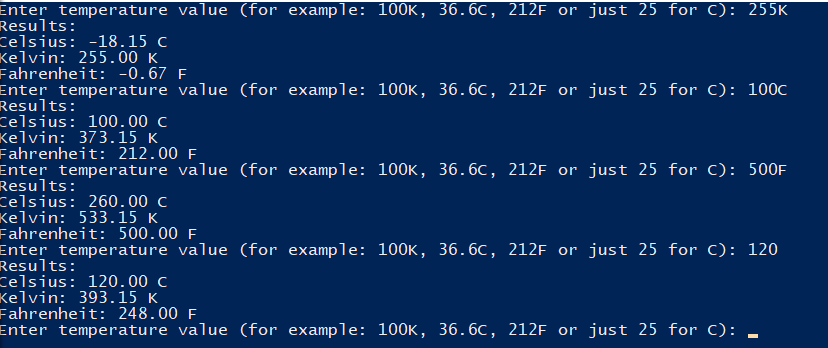
Для обработки ошибок используем структуру try-catch.

Рисунок 2.5 – Результат выполнения скрипта перевода температур

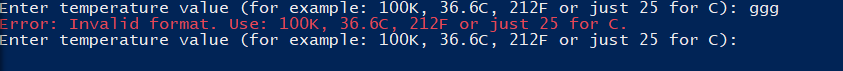


Рисунок 2.6 – Тестирование некорректного ввода

# **3. Изучение основ bash в Linux**

Bash – это командная оболочка для для Unix-подобных операционных систем (таких как Linux, macOS). Она предоставляет интерфейс командной строки (CLI) для выполнения задач, таких как навигация по файлам, управление процессами и настройка системы. Кроме того, Bash поддерживает создание скриптов – файлов с последовательностями команд, которые используются для автоматизации рутинных задач.

Сравнение команд в Windows и Bash

|  |  |
| --- | --- |
| Windows | Bash |
| cd | cd (change directory)  Дополнительные возможности:   * cd или cd ~ (переход в домашнюю директорию); * cd – (возврат в предыдущую рабочую директорию); * cd ~username (переход в домашний каталог указанного пользователя). |
| copy , xcopy | cp (copy) является аналогом Windows copy и xcopy.  Основные параметры:   * i (запрос подтверждения перезаписи); * r, R (рекурсивное копирование директорий); * v (подробный вывод процесса копирования); * p (сохранение атрибутов файлов); * u (копирование только обновленных файлов). |
| copy \*.\* + \*.\* | cat (concatenate) предоставляет многофункциональные возможности работы с файлами файлы, как просмотр их содержимого, создание и объединение файлов.  cat \*.\*+\*.\*>\*.\* |
| copy con  copy con + \*.\* | tee осуществляет одновременную запись в стандартный вывод и файлы. Она названа подобным образом в следствии сходства Т-образной трубы, что отлично отображает сущность ее задачи. Ключ -a добавляет данные в конец файла. |
| move, ren | mv (move) выполняет перемещение и переименование файлов.  Ключи:   * i (подтверждение перезаписи) * v (подробный вывод). |
| fc | diff (difference) сравнивает файлы и директории  Основные ключи:   * u (унифицированный формат вывода); * r (рекурсивное сравнение директорий); * i (игнорирование регистра). |
| del | rm (remove) выполняет удаление файлов и директорий. Ключи:   * i (интерактивный режим с подтверждением); * r, R (рекурсивное удаление); * f (принудительное удаление); |
| md | mkdir используется для создания директорий.  Основные ключи:   * p (создание промежуточных каталогов); * v (вывод создаваемых директориев); * m (установка прав доступа к папке). |
| rd /Q | rmdir используется для удаления пустых директорий. Основные ключи:   * p (удаление родительских каталогов); * v (вывод удляемых директориев). |
| rd | rm используется для удаления каталога с содержимым |
| dir | ls (list) предназначена для вывода содержимого директорий и предоставляет расширенные возможности форматирования вывода:  Ключи:   * -l (подробный формат с правами доступа, владельцем, размером и временем модификации); * -S (сортировка по размер); * -r (обратный порядок сортировки); * -X (сортировка по расширению файлов); * -a (показ всех файлов, включая скрытые); * -R (рекурсивный обход поддиректорий); |

# **3.13 Команда touch**

Данная команда служит для изменения временных меток файлов и создания новых файлов.

**3.14 Команда echo**

Данная команд выводит текст в стандартный вывод или перенаправляет его в файлы. В качестве параметров она принимает строку, которую мы хотим вывести. Давайте рассмотрим ключи.

Ключи управления выводом:

* -n (подавление добавления символа новой строки);
* -e (включение интерпретации escape-последовательностей);
* -E (явное отключение интерпретации escape-последовательностей).

**3.15 Команда grep**

Команда grep (Global Regular Expression Print) предназначена для поиска текста по шаблону в файлах или же в потоке данных c поддержкой регулярных выражений.

# **3.16 Операторы**

В Linux предоставляется богатый набор операторов для различных задач. В рамках данной лабораторной работы мы рассмотрим операторы перенаправления и операторы управления процессами.

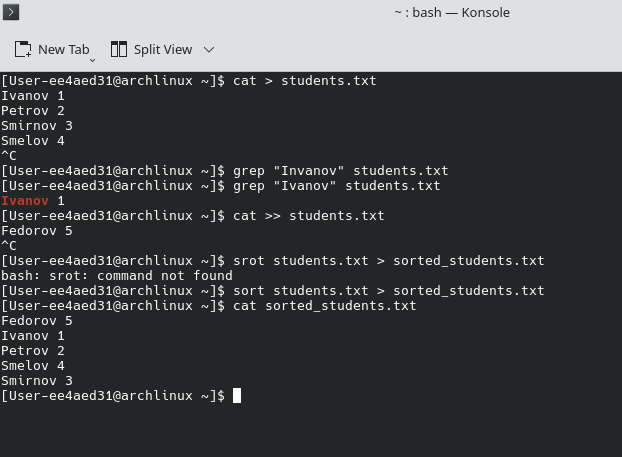
Операторы перенаправления:

* > (перенаправление stdout с перезаписью файла);
* >> (перенаправление stdout с добавлением в файл);
* < (перенаправление stdin из файла).

Операторы управления процессами:

* & (запуск процесса в фоновом режиме);
* ; (последовательное выполнение команд независимо от результата);
* && (логическое И);
* || (логическое ИЛИ);
* ! (инверсия кода возврата).

На рисунке 3.1 представлена последовательность команд для создания файла и записи в него информации с консоли, затем поиск информации, далее добавление новых записей в конец файла и вывод всех записей из файла в отсортированном порядке.

Рисунок 3.1 – Выполнение заданной последовательности команд

# **3.17 Переменные окружения в Linux**

Система переменных окружения в Linux аналогична Windows, но имеет свои особенности:

* глобальные переменные доступны всем процессам;
* локальные переменные ограничены текущей сессией;
* основные утилиты: printenv, env, export;

Самой важной переменной, как и в Windows считается PATH, так как там можно найти все необходимые нам исполняемые файлы.

# **3.18 Правила формирования названий команд и параметров в Linux**

В наименовании команд используются следующие принципы:

* аббревиатуры: cd (change directory), ps (process status);
* усечение: diff (difference), dir (directory);
* удаление гласных: rm (remove), cp (copy), mv (move);
* композитные формы: chmod (change mode), ssh (secure shell);
* прямые обозначения: find, echo, sort.

# **3.19 Sh-файлы и как ими пользоваться**

Shell-скрипты (файлы .sh) являются аналогом BAT-файлов в Windows и служат для автоматизации задач. Для запуска скрипта сначала нам нужно дать права на исполнение скрипта с помощью команды chmod + x file\_name.sh

Первым заданием нам нужно написать скрипт, который используя системные переменные, выведет нам имя текущего пользователя, имя компьютера и дату. В листинге 3.1 представлен данный скрипт. На рисунке 3.2 продемонстрирован результат выполнения.

|  |
| --- |
| #!/bin/bash  echo $USER  echo $NAME  date |

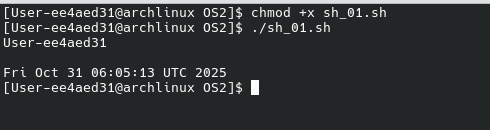
Листинг 3.1 Содержимое sh\_01.sh

Рисунок 3.2 – Результат выполнения файла sh\_01.sh

Во втором задании нам нужно написать скрипт, который выведет нам имя текущего sh-файла, время последней модификации и полный путь к файлу. В листинге 3.2 представлен данный скрипт. На рисунке 3.3 продемонстрирован результат выполнения.

|  |
| --- |
| #!/bin/bash  echo "$(basename $0)"  echo "$(stat -c %y $0)"  echo "$(realpath $0)" |

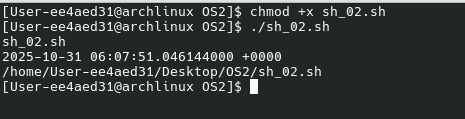
Листинг 3.2 Содержимое sh\_02.sh

Рисунок 3.3 – Результат выполнения файла sh\_02.sh

В третьем задании задачей являлась работа с параметрами. Было необходимо вывести все переданные параметры в строку, а также первые 4 по отдельности. В листинге 3.3 представлен данный скрипт. На рисунке 3.4 продемонстрирован результат выполнения.

|  |
| --- |
| #!/bin/bash  echo $\*  echo $1  echo $2  echo $3  echo $4 |

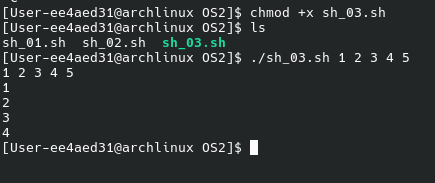
Листинг 3.3 Содержимое sh\_03.sh

Рисунок 3.4 – Результат выполнения файла sh\_03.sh

В четвертом задании нам нужно написать скрипт, который будет вычислять арифметические выражения с переданными в него параметрами. В листинге 3.4 представлен данный скрипт. На рисунке 3.5 продемонстрирован результат выполнения.

|  |
| --- |
| #!/bin/bash  a=$1  b=$2  c=$3  res1=$((a - b))  echo "$a - $b = $res1"  res2=$((a + b))  echo "$a + $b = $res2"  if [ "$b" -eq 0 ]; then  echo "Error: attempt to divide by zero!"  else  res3=$((c / b))  echo "$c / $b = $res3"  fi  res4=$((c \* b))  echo "$c \* $b = $res4"  res5=$(( (a - b) \* (b - a) ))  echo "($a - $b) \* ($b - $a) = $res5" |

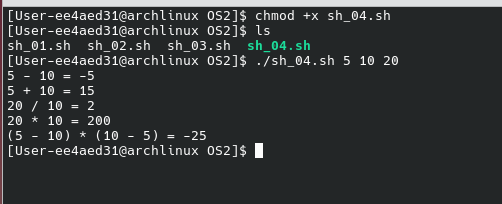
Листинг 3.4 Содержимое sh\_04.sh

Рисунок 3.5 – Результат выполнения файла sh\_04.sh

В пятом задании было необходимо написать скрипт, позволяющий создавать и удалять файлы и который принимает на вход два параметра: режим работы и название файла, а также ловит ошибки и выводит сообщения в консоль. В листинге 3.5 представлен данный скрипт. На рисунке 3.6 продемонстрирован результат выполнения.

|  |
| --- |
| #!/bin/bash  if [ -z "$1" ] && [ -z "$2" ]; then  echo "Help: set first param to work mode (create, delete), second to a file name."  exit 0  fi  if [ -z "$1" ]; then  echo "Error: work mode param is missing"  exit 1  fi  if [ "$1" != "create" ] && [ "$1" != "delete" ]; then  echo "Help: wrong work mode input (try create/delete)"  exit 1  fi  if [ -z "$2" ]; then  echo "Error: file name is missing"  exit 1  fi  if [ "$1" = "create" ]; then  if [ -f "$2" ]; then  echo "Error: file already exists"  exit 1  else  touch "$2"  echo "File created: $2"  exit 0  fi  fi  if [ "$1" = "delete" ]; then  if [ ! -f "$2" ]; then  echo "Error: no file was found"  exit 1  else  rm "$2"  echo "File deleted: $2"  exit 0  fi  fi |

Листинг 3.5 Содержимое sh\_05.sh

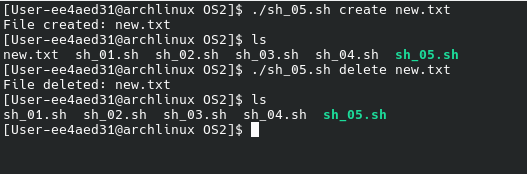


Рисунок 3.6 – Результат выполнения файла sh\_05.sh

# **4. Основы bash-скриптов**

В данном разделе мы немного подробнее разберем некоторые особенности синтаксиса bash-конструкций, а также напишем свои более сложные скрипты на основе полученных знаний.

# **4.1 Написание собственных скриптов**

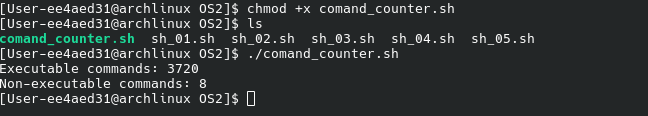
Для первого задания было необходимо написать скрипт, который анализирует директории из переменной PATH и подсчитывает количество исполняемых и неисполняемых файлов. Он представлен в приложении В. На рисунке 4.1 показан результат выполнения.

Рисунок 4.1 – Результат выполнения скрипта 1

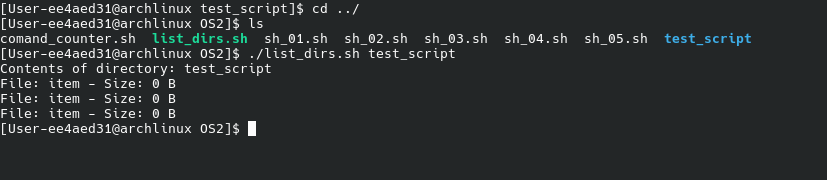
В качестве последнего задания необходимо написать программу, которая выведет список вложенных каталогов и файлов в указанном каталоге. При этом для вложенных каталогов выводится количество файлов внутри них, а для файлов выводится их размер. Скрипт описан в приложение Г. На рисунке 4.2 представлен результат выполнения данного скрипта.

Рисунок 4.2 – Результат выполнения скрипта 2

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы были успешно освоены практические навыки работы с командной оболочкой Linux и файловой системой. В процессе работы выполнены ключевые операции управления файлами и каталогами, включая создание структурированных директорий, копирование с сохранением атрибутов, изменение прав доступа и массовые операции с файлами.

**Приложение А**

Скрипт 1 для Powershell

function Read-Number([string]$prompt) {

while ($true) {

$input = Read-Host $prompt

if ($input -match '^\s\*-?\d+([.,]\d+)?\s\*$') {

$norm = $input -replace ',', '.'

return [double]$norm

}

else {

Write-Host "Incorrect number. Try again" -ForegroundColor Yellow

}

}

}

function Read-Int([string]$prompt) {

while ($true) {

$user\_input = Read-Host $prompt

if ($user\_input -match '^\s\*-?\d+\s\*$') {

return [int]$user\_input

}

else {

Write-Host "Incorrect number. Try again" -ForegroundColor Yellow

}

}

}

function To-Radians([double]$value, [string]$unit) {

if ($unit -eq 'd') {

return $value \* [System.Math]::PI / 180.0

}

return $value

}

function From-Radians([double]$value, [string]$unit) {

if ($unit -eq 'd') {

return $value \* 180.0 / [System.Math]::PI

}

return $value

}

function Show-Menu {

Clear-Host

Write-Host "---Interactive Calculator---"

Write-Host "Select an operation: "

Write-Host "1) Addition (a + b)"

Write-Host "2) Substraction (a - b)"

Write-Host "3) Multiplication (a \* b)"

Write-Host "4) Division (a / b)"

Write-Host "5) Exponentiation (a ^ b)"

Write-Host "6) N-power root extraction (n√a)"

Write-Host "7) Remainder from division (a mod b)"

Write-Host "8) Trigonometric functions (sin, cos, tan, ctg)"

Write-Host "9) Logorythms (ln, log10)"

Write-Host "0) Exit"

}

function Do-Trigonometry {

$unit = ''

while ($unit -notin @('r', 'd')) {

$unit = Read-Host "Enter values for trygonometric functions: (r) radiand (d) degrees"

$unit = $unit.ToLower()

}

$ops = @('sin', 'cos', 'tan', 'ctg')

$op = ''

while ($op -notin $ops) {

$op = Read-Host "Function (${($ops -join ', ')})"

$op.ToLower()

}

$x = Read-Number "Enter the argument"

$rad = To-Radians $x $unit

switch ($op) {

'sin' { $res = [System.Math]::Sin($rad) }

'cos' { $res = [System.Math]::Cos($rad) }

'tan' { $res = [System.Math]::Tan($rad) }

'ctg' { $res = 1 / [System.Math]::Tan($rad) }

}

Write-Host "Result: $res"

}

function Do-Logarythm {

$ops = @('ln', 'log10')

$op = ''

while ($op -notin $ops) {

$op = Read-Host "Choose the logorythmic operation (ln,log10)"

$op.ToLower()

}

$x = Read-Number "Enter the POSITIVE number"

if ($x -le 0) {

Write-Host "Logorythm is defined for positive number only." -ForegroundColor Yellow

return

}

switch ($op) {

'ln' { $res = [System.Math]::Log($x) }

'log10' { $res = [System.Math]::Log10($x) }

}

Write-Host "Result: $x"

}

$exited = $false

while ($true) {

if($exited){

break

}

Show-Menu

$choise = Read-Host "Select an operation"

switch ($choise) {

'1' {

$a = Read-Number "a="

$b = Read-Number "b="

Write-Host "Addition result: $([Math]::Round(($a+$b),12))"

}

'2' {

$a = Read-Number "a="

$b = Read-Number "b="

Write-Host "Substraction result: $($a-$b)"

}

'3' {

$a = Read-Number "a="

$b = Read-Number "b="

Write-Host "Multiplication result: $($a\*$b)"

}

'4' {

$a = Read-Number "a="

$b = Read-Number "b="

if ($b -eq 0) {

Write-Host "Unable to devide by zero" -ForegroundColor Red

}

else {

Write-Host "Division result: $($a/$b)"

}

}

'5' {

$a = Read-Number "Basis="

$b = Read-Number "Power="

try {

$res = [System.Math]::Pow($a, $b)

Write-Host "Exponentiation result: $res"

}

catch {

Write-Host "Error $\_" -ForegroundColor Red

}

}

'6' {

$N = Read-Number "Root power="

if ($N -eq 0) {

Write-Host "Root power must be more then 0" -ForegroundColor Yellow

}

else {

$a = Read-Host "a="

if ($a -lt 0 -and ($N % 2 -eq 0)) {

Write-Host "Unable to extract even root from a negative number" -ForegroundColor Yellow

}

else {

try {

$res = [System.Math]::Pow($a, 1.0 / $n)

Write-Host "N-power root extraction result: $res"

}

catch {

Write-Host "Error: $\_" -ForegroundColor Red

}

}

}

}

'7' {

$a = Read-Number "a="

$b = Read-Number "b="

if ($b -eq 0) {

Write-Host "Error: Unable to devide by zero" -ForegroundColor Red

}

else {

$quot = [System.Math]::Floor($a / $b)

$res = $a - ($quot \* $b)

Write-Host "Remainder from division result: $res"

}

}

'8' {

Do-Trigonometry

}

'9' {

Do-Logarythm

}

'0' {

Write-Host "Exiting the application..."-ForegroundColor Green

$exited = $true

break

}

default {

Write-Host "Invalid choise. Try again" -ForegroundColor Yellow

}

}

Write-Host ""

Read-Host "Press any key to continue..."

}

**Приложение Б**

Скрипт 2 для PowerShell

function Parse-TempInput {

param([string]$temp\_input)

if (-not $temp\_input) {

throw "Empty input"

}

$s = $temp\_input.Trim().Replace(" ", "")

if($s -eq "exit"){

return -1

}

if ($s -match '^(?<num>-?\d+\.?\d\*)(?<unit>[cCfFkK])$') {

$num = $matches['num']

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

}

elseif ($s -match '^(?<num>-?\d+\.?\d\*)$') {

$num = $matches['num']

$unit = 'C'

}

else {

throw "Invalid format. Use: 100K, 36.6C, 212F or just 25 for C."

}

return @{ Value = [double]$num; Unit = $unit }

}

function Convert-Temperature {

param([double]$value, [string]$unit)

switch ($unit) {

'C' {

$kelvin = $value + 273.15

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

return @{ Celsius = $value; Kelvin = $kelvin; Fahrenheit = $fahrenheit }

}

'K' {

$celsius = $value - 273.15

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

return @{ Celsius = $celsius; Kelvin = $value; Fahrenheit = $fahrenheit }

}

'F' {

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

$kelvin = $celsius + 273.15

return @{ Celsius = $celsius; Kelvin = $kelvin; Fahrenheit = $value }

}

}

}

while($true){

try {

$user\_input = Read-Host "Enter temperature value (for example: 100K, 36.6C, 212F or just 25 for C)"

$parsed = Parse-TempInput $user\_input

if($parsed -eq -1){

break

}

$result = Convert-Temperature -value $parsed.Value -unit $parsed.Unit

Write-Host "Results:"

Write-Host ("Celsius: {0:N2} C" -f $result.Celsius)

Write-Host ("Kelvin: {0:N2} K" -f $result.Kelvin)

Write-Host ("Fahrenheit: {0:N2} F" -f $result.Fahrenheit)

}

catch {

Write-Host "Error: $\_" -ForegroundColor Red

}

}

**Приложение В**

Скрипт 1 для bash

#!/bin/bash

executable\_count=0

non\_executable\_count=0

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

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

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

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

if [[ -x "$command" ]]; then

((executable\_count++))

else

((non\_executable\_count++))

fi

done

fi

done

echo "Executable commands: $executable\_count"

echo "Non-executable commands: $non\_executable\_count"

**Приложение Г**

Скрипт 2 для bash

#!/bin/bash

convert\_size() {

local size\_in\_bytes=$1

if (( size\_in\_bytes < 1024 )); then

echo "${size\_in\_bytes} B"

elif (( size\_in\_bytes < 1048576 )); then

echo "$(( size\_in\_bytes / 1024 )) KB"

elif (( size\_in\_bytes < 1073741824 )); then

echo "$(( size\_in\_bytes / 1048576 )) MB"

else

echo "$(( size\_in\_bytes / 1073741824 )) GB"

fi

}

list\_directory() {

local dir=$1

echo "Contents of directory: $dir"

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

if [ -d "$item" ]; then

local file\_count=$(find "$item" -type f | wc -l)

echo "Directory: $(basename "$item") - Files: $file\_count"

list\_directory "$item"

elif [ -f "$item" ]; then

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

local readable\_size=$(convert\_size "$size")

echo "File: $(basename "$item") - Size: $readable\_size"

fi

done

}

if [ -z "$1" ]; then

echo "Please specify a directory to analyze."

exit 1

fi

list\_directory "$1"