

ОП.01 Операционные системы, среды и архитектура аппаратных средств

Тема: Работа со встроенными приложениями операционных систем Windows и Linux (текстовые редакторы, архиваторы, средства резервного копирования, стандартные системные утилиты)

Цель практического занятия

Формирование у обучающихся практических навыков работы со встроенными приложениями операционных систем Windows и Linux и их практического использования при создании сценариев автоматизации на языках BAT, PowerShell и Bash.

Задачи практического занятия

1. Изучить встроенные текстовые редакторы Windows и Linux как инструменты создания и редактирования сценариев автоматизации.
2. Освоить встроенные средства архивирования данных и научиться применять их в сценариях BAT, PowerShell и Bash.
3. Изучить встроенные средства резервного копирования и реализовать процедуры создания резервных копий с использованием командной строки.
4. Освоить базовые команды работы с файловой системой в средах:
 1. PowerShell;
 2. Bash (Linux).
5. Научиться разрабатывать простые автоматизированные сценарии:
 1. создание каталогов;
 2. создание и редактирование файлов;
 3. копирование и перемещение данных;
 4. удаление файлов и каталогов;
 5. архивирование данных;
 6. резервное копирование.
6. Сформировать понимание различий между средами выполнения BAT, PowerShell и Bash.
7. Развить навыки анализа возможностей встроенных средств Windows и Linux с точки зрения автоматизации системных задач.

Практическая часть

Задача 1. PowerShell (Windows): создание рабочей структуры и резервное копирование с архивированием

Цель задачи: создать и выполнить два PowerShell-скрипта:

1. скрипт создаёт структуру папок и формирует системный отчёт (info.txt)

2. скрипт создаёт ZIP-архив резервной копии и ведёт журнал (backup.log)

Этап 1. Создание папки для задания (где будут лежать скрипты)

Вариант А (через Проводник)

1. Win + E → открыть Проводник
2. Перейти в Документы:
C:\Users\areis\Documents
3. Создать папку:
OP01_P16_Task4

Итоговый путь:

C:\Users\areis\Documents\OP01_P16_Task4

Вариант В (через PowerShell командой)

1. Открыть PowerShell (Win + X → Windows Terminal/PowerShell)
2. Выполнить команды:

```
cd C:\Users\areis\Documents
```

```
mkdir OP01_P16_Task4
```

```
cd OP01_P16_Task4
```

```
PS C:\WINDOWS\System32> cd C:\Users\grtgr\Documents
```

```
PS C:\Users\grtgr\Documents> mkdir OP01_P16_Task4
```

Каталог: C:\Users\grtgr\Documents

Mode	LastWriteTime	Length	Name
d-----	28.02.2026 10:55		OP01_P16_Task4

```
PS C:\Users\grtgr\Documents>
```

```
PS C:\Users\grtgr\Documents> cd OP01_P16_Task4
```

```
PS C:\Users\grtgr\Documents\OP01_P16_Task4> |
```

Проверка:

```
pwd
```

```
dir
```

```
PS C:\Users\grtgr\Documents\OP01_P16_Task4> pwd

Path
----
C:\Users\grtgr\Documents\OP01_P16_Task4

PS C:\Users\grtgr\Documents\OP01_P16_Task4> dir
PS C:\Users\grtgr\Documents\OP01_P16_Task4> |
```

Этап 2. Разрешить запуск локальных скриптов (делается один раз)

В PowerShell выполнить:

Этап 2. Разрешить запуск локальных скриптов (делается один раз)

В PowerShell выполнить:

```
Set-ExecutionPolicy -Scope CurrentUser RemoteSigned
```

Если спросит подтверждение — нажать Y и Enter.

Проверка текущей политики:

```
Get-ExecutionPolicy -Scope CurrentUser
```

Ожидаемо: RemoteSigned

```
PS C:\Users\grtgr\Documents\OP01_P16_Task4> Set-ExecutionPolicy -Scope CurrentUser RemoteSigned
PS C:\Users\grtgr\Documents\OP01_P16_Task4> Get-ExecutionPolicy -Scope CurrentUser
RemoteSigned
PS C:\Users\grtgr\Documents\OP01_P16_Task4> |
```

Этап 3. Скрипт №1 Create-Workspace.ps1

3.1 Создать файл скрипта

Находясь в папке:

C:\Users\areis\Documents\OP01_P16_Task4

Открыть Блокнот для создания файла:

notepad .\Create-Workspace.ps1

3.2 Впишите код

В Блокноте впишите код и сохранить файл:

```
Create-Workspace.ps1
Файл  Изменить  Просмотр

$fullName = Read-Host "Enter student's full name"

$documents = [Environment]::GetFolderPath('MyDocuments')
$baseDir = Join-Path $documents "Practice16"

$scriptsDir = Join-Path $baseDir "Scripts"
$backupsDir = Join-Path $baseDir "Backups"
$logsDir = Join-Path $baseDir "Logs"
$infoFile = Join-Path $baseDir "info.txt"

New-Item -ItemType Directory -Force -Path $scriptsDir | Out-Null
New-Item -ItemType Directory -Force -Path $backupsDir | Out-Null
New-Item -ItemType Directory -Force -Path $logsDir | Out-Null

$os = Get-CimInstance Win32_OperatingSystem

$lines = @(
    "Student: $fullName"
    "Date: $(Get-Date)"
    "Computer Name: $env:COMPUTERNAME"
    "OS: $($os.Caption)"
    "Version: $($os.Version)"
    "Build: $($os.BuildNumber)"
)

$lines | Set-Content -Encoding UTF8 -Path $infoFile

Write-Host ""
Write-Host "Workspace successfully created."
Write-Host "Folder: $baseDir"
Write-Host "Info file: $infoFile"
```

Сохранение:

1. File → Save (или Ctrl + S)
2. Закрыть Блокнот

Важно: если используете «Сохранить как», выбирайте кодировку UTF-8.

1. Ввод данных пользователя

```
$fullName = Read-Host "Enter student's full name"
```

1. Read-Host — команда для ввода данных с клавиатуры.
2. В кавычках — текст приглашения.
3. Всё, что введёт пользователь, сохраняется в переменную \$fullName.
4. \$fullName — строковая переменная.

2. Определение системной папки "Документы"

```
$documents = [Environment]::GetFolderPath('MyDocuments')
```

1. [Environment] — .NET-класс.
2. GetFolderPath('MyDocuments') — получает реальный путь к папке «Документы».
3. Это важно, потому что папка может быть:
 1. обычной (C:\Users\Имя\Documents)
 2. или перенаправленной (например, в OneDrive).
4. Путь сохраняется в переменную \$documents.

3. Формирование пути к рабочей папке

```
$baseDir = Join-Path $documents "Practice16"
```

1. Join-Path корректно соединяет два пути.
2. Не нужно вручную писать \.
3. В результате получается путь:

C:\Users\...\Documents\Practice16

4. Сохраняется в \$baseDir.

4. Формирование путей к подкаталогам и файлу

```
$scriptsDir = Join-Path $baseDir "Scripts"  
$backupsDir = Join-Path $baseDir "Backups"  
$logsDir = Join-Path $baseDir "Logs"  
$infoFile = Join-Path $baseDir "info.txt"
```

Создаются переменные с путями:

1. \$scriptsDir → ...Practice16\Scripts
2. \$backupsDir → ...Practice16\Backups
3. \$logsDir → ...Practice16\Logs
4. \$infoFile → ...Practice16\info.txt

На этом этапе файлы и папки ещё не созданы — мы только подготовили пути.

5. Создание каталогов

```
New-Item -ItemType Directory -Force -Path $scriptsDir | Out-Null  
New-Item -ItemType Directory -Force -Path $backupsDir | Out-Null  
New-Item -ItemType Directory -Force -Path $logsDir | Out-Null
```

Разберём одну строку:

1. New-Item — создаёт новый объект (файл, папку и т.д.)
2. -ItemType Directory — указывает, что создаём папку
3. -Path \$scriptsDir — путь, где создаётся папка
4. -Force — если папка уже существует, ошибка не возникнет
5. | Out-Null — скрывает вывод команды в консоль

Итог: создаются три папки (если их не было).

6. Получение информации об ОС

```
$os = Get-CimInstance Win32_OperatingSystem
```

1. Get-CimInstance — получает данные из системы (через WMI/CIM).
2. Win32_OperatingSystem — класс с информацией об ОС.

3. В переменную `$os` сохраняется объект с множеством свойств:

1. `Caption` (название ОС)
2. `Version`
3. `BuildNumber`
4. и другие

7. Создание массива строк

```
$lines = @(
    "Student: $fullName"
    "Date: $(Get-Date)"
    "Computer Name: $env:COMPUTERNAME"
    "OS: $($os.Caption)"
    "Version: $($os.Version)"
    "Build: $($os.BuildNumber)"
)
```

Что происходит:

1. `@()` — создаёт массив.
2. Внутри — набор строк.
3. Переменные подставляются автоматически.

Разберём отдельные конструкции:

`$fullName`

Подставляет введённое имя.

`$(Get-Date)`

1. `Get-Date` возвращает текущую дату и время.
2. `$()` — подвыражение (чтобы выполнить команду внутри строки).

`$env:COMPUTERNAME`

1. Обращение к переменной среды Windows.
2. Возвращает имя компьютера.

`$(os.Caption)`

1. `$os` — объект.
2. `.Caption` — его свойство.
3. `$()` нужно, чтобы корректно вставить значение в строку.

Итог: `$lines` — массив из 6 строк.

8. Запись массива в файл

```
$lines | Set-Content -Encoding UTF8 -Path $infoFile
```

1. | — конвейер (pipeline).
2. \$lines передаётся в Set-Content.
3. Set-Content — записывает данные в файл.
4. -Encoding UTF8 — указывает кодировку (важно для текста).
5. -Path \$infoFile — путь к файлу.

Если файла нет — он создаётся.

Если файл есть — он перезаписывается.

9. Вывод информации в консоль

```
Write-Host "Done."  
Write-Host "Workspace folder: $baseDir"  
Write-Host "Info file:      $infoFile"
```

1. Write-Host выводит текст в консоль.
2. \$baseDir и \$infoFile подставляются в строку.
3. Это информирует пользователя, где созданы папка и файл.

Итог работы скрипта

После выполнения:

1. Определяется системная папка Documents.
2. Создаётся структура:

```
Documents\Practice16  
  Scripts  
  Backups  
  Logs
```

3. Создаётся файл:

```
Documents\Practice16\info.txt
```

4. В файл записываются:
 1. имя студента
 2. дата
 3. имя компьютера
 4. данные об ОС
5. В консоль выводится результат.

3.3 Запустить скрипт №1

В PowerShell выполнить:

```
Set-ExecutionPolicy -Scope CurrentUser RemoteSigned  
.\Create-Workspace.ps1
```

Скрипт попросит ввести ФИО (на английском). Ввести и нажать Enter.

3.4 Проверить результат скрипта №1

1. Узнать реальный путь Documents, который использует скрипт:

```
[Environment]::GetFolderPath('MyDocuments')
```

2. Проверить, что создана структура Practice16:

```
$docs = [Environment]::GetFolderPath('MyDocuments')  
dir (Join-Path $docs "Practice16")  
dir (Join-Path $docs "Practice16\Scripts")  
dir (Join-Path $docs "Practice16\Backups")  
dir (Join-Path $docs "Practice16\Logs")
```

3. Проверить содержимое файла info.txt:

```
Get-Content (Join-Path $docs "Practice16\info.txt")
```

Ожидаемо: строки Student / Date / Computer Name / OS / Version / Build.

```
PS C:\Users\grtgr\Documents\OP01_P16_Task4> $docs = [Environment]::GetFolderPath('MyDocuments')  
PS C:\Users\grtgr\Documents\OP01_P16_Task4> dir (Join-Path $docs "Practice16")  
  
Каталог: C:\Users\grtgr\Documents\Practice16  
  
Mode                LastWriteTime         Length Name  
----                -  
d-----         28.02.2026        11:10      Backups  
d-----         28.02.2026        11:10        Logs  
d-----         28.02.2026        11:10     Scripts  
-a----         28.02.2026        11:10       187 info.txt  
  
PS C:\Users\grtgr\Documents\OP01_P16_Task4> dir (Join-Path $docs "Practice16\Scripts")  
PS C:\Users\grtgr\Documents\OP01_P16_Task4> dir (Join-Path $docs "Practice16\Backups")  
PS C:\Users\grtgr\Documents\OP01_P16_Task4> dir (Join-Path $docs "Practice16\Logs")  
PS C:\Users\grtgr\Documents\OP01_P16_Task4> Get-Content (Join-Path $docs "Practice16\info.txt")  
Student: Иванов Иван Иванович  
Date: 02/28/2026 11:10:51  
Computer Name: DESKTOP-BAC8E6C  
OS: Майкрософт Windows 11 Pro  
Version: 10.0.26200  
Build: 26200
```

Этап 4. Скрипт №2 Backup-Workspace.ps1

4.1 Создать файл скрипта

Вернуться (если нужно) в папку задания:

```
cd C:\Users\grtgr\Documents\OP01_P16_Task4
```

Создать/открыть файл:

```
notepad .\Backup-Workspace.ps1
```

4.2 Вставить код

Вставить и сохранить:

```
$ErrorActionPreference = "Stop"

$documents = [Environment]::GetFolderPath('MyDocuments')
$baseDir = Join-Path $documents "Practice16"
$backupsDir = Join-Path $baseDir "Backups"
$logsDir = Join-Path $baseDir "Logs"
$logFile = Join-Path $logsDir "backup.log"

try {
    if (-not (Test-Path $baseDir)) {
        throw "Workspace folder not found: $baseDir. Run Create-Workspace.ps1 first."
    }

    New-Item -ItemType Directory -Force -Path $backupsDir | Out-Null
    New-Item -ItemType Directory -Force -Path $logsDir | Out-Null

    $dateTag = Get-Date -Format "yyyy-MM-dd_HH-mm-ss"
    $archiveName = "Practice16_backup_$dateTag.zip"
    $archivePath = Join-Path $backupsDir $archiveName

    if (Test-Path $archivePath) {
        Remove-Item $archivePath -Force
    }

    $itemsToZip = Join-Path $baseDir "*"
    $items = Get-ChildItem -Path $itemsToZip -Force -ErrorAction SilentlyContinue

    if ($null -eq $items -or $items.Count -eq 0) {
```

```
throw "Nothing to archive: $baseDir is empty."
}

Compress-Archive -Path $itemsToZip -DestinationPath $archivePath -Force

if (-not (Test-Path $archivePath)) {
    throw "ZIP file was not created (Compress-Archive did not produce output)."
}

$sizeBytes = (Get-Item $archivePath).Length
$sizeMB    = [Math]::Round($sizeBytes / 1MB, 2)

$logLines = @(
    "Date: $(Get-Date)"
    "Source: $baseDir"
    "Archive: $archiveName"
    "Path: $archivePath"
    "Size: $sizeMB MB"
    "-----"
)

$logLines | Add-Content -Encoding UTF8 -Path $logFile

Write-Host "Backup completed."
Write-Host "Archive: $archivePath"
Write-Host "Log:    $logFile"
}
catch {
    $errorMessage = "ERROR: $($_.Exception.Message)"
```

```
New-Item -ItemType Directory -Force -Path $logsDir | Out-Null

Add-Content -Encoding UTF8 -Path $logFile -Value $errorMessage

Write-Host $errorMessage

exit 1

}
```

Сохранить и закрыть Блокнот.

4.3 Запустить скрипт №2

```
.\Backup-Workspace.ps1
```

```
PS C:\Users\grtgr\Documents\OP01_P16_Task4> .\Backup-Workspace.ps1
Backup completed.
Archive: C:\Users\grtgr\Documents\Practice16\Backups\Practice16_backup_2026-02-28_11-18-26.zip
Log: C:\Users\grtgr\Documents\Practice16\Logs\backup.log
```

4.4 Проверить результат скрипта №2

1. Проверить, что появился ZIP-архив:

```
$docs = [Environment]::GetFolderPath('MyDocuments')
dir (Join-Path $docs "Practice16\Backups")
```

Ожидаемо: файл вида Practice16_backup_yyyy-MM-dd_HH-mm-ss.zip

```
PS C:\Users\grtgr\Documents\OP01_P16_Task4> $docs = [Environment]::GetFolderPath('MyDocuments')
PS C:\Users\grtgr\Documents\OP01_P16_Task4> dir (Join-Path $docs "Practice16\Backups")

Каталог: C:\Users\grtgr\Documents\Practice16\Backups

Mode                LastWriteTime         Length Name
----                -
-a-----         28.02.2026   11:18             277 Practice16_backup_2026-02-28_11-18-26.zip
```

2. Проверить лог:

```
Get-Content (Join-Path $docs "Practice16\Logs\backup.log")
```

Ожидаемо: запись Date / Source / Archive / Path / Size.

```
PS C:\Users\grtgr\Documents\OP01_P16_Task4> Get-Content (Join-Path $docs "Practice16\Logs\backup.log")
Date: 02/28/2026 11:18:27
Source: C:\Users\grtgr\Documents\Practice16
Archive: Practice16_backup_2026-02-28_11-18-26.zip
Path: C:\Users\grtgr\Documents\Practice16\Backups\Practice16_backup_2026-02-28_11-18-26.zip
Size: 0 MB
```

3. Проверить, что архив реально открывается (проверка списка файлов):

Каждую команду вписываем по отдельности

```
$zip = (Get-ChildItem (Join-Path $docs "Practice16\Backups") -Filter *.zip | Sort-Object
LastWriteTime -Descending | Select-Object -First 1).FullName
```

```
Add-Type -AssemblyName System.IO.Compression.FileSystem
```

```
[System.IO.Compression.ZipFile]::OpenRead($zip).Entries | Select-Object -First 10 FullName
```

```
PS C:\Users\grtgr\Documents\OP01_P16_Task4> $zip = (Get-Childitem (Join-Path $docs "Practice16\Backups") -Filter *.zip | Sort-Object LastWriteTime -Descending | Select-Object -First 1).FullName
PS C:\Users\grtgr\Documents\OP01_P16_Task4> Add-Type -AssemblyName System.IO.Compression.FileSystem
PS C:\Users\grtgr\Documents\OP01_P16_Task4> [System.IO.Compression.ZipFile]::OpenRead($zip).Entries | Select-Object -First 10 FullName

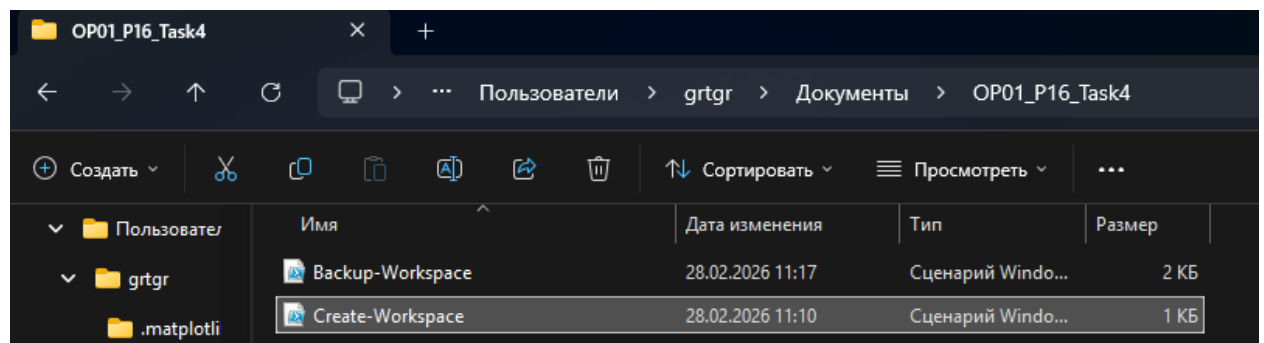
FullName
-----
info.txt
```

Итог выполнения задачи 4 (что должно быть в конце)

Папка со скриптами:

C:\Users\grtgr\Documents\OP01_P16_Task4

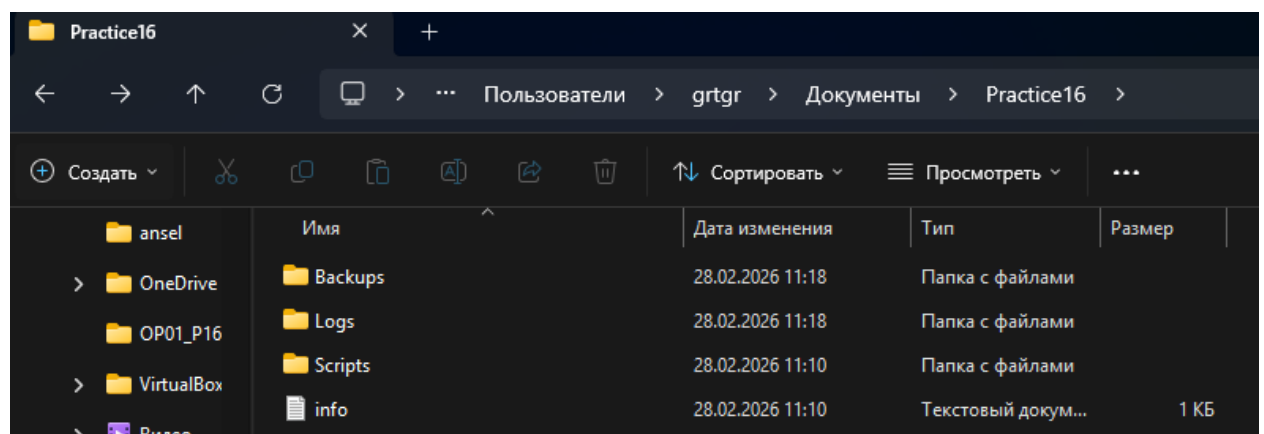
1. Create-Workspace.ps1
2. Backup-Workspace.ps1



Рабочая структура:

(MyDocuments)\Practice16

1. Scripts
2. Backups (там zip-архив)
3. Logs (там backup.log)
4. info.txt



Задача 2. Разработка и выполнение Bash-скриптов в виртуальной машине (Debian 11)

Требование: выполнить все действия в виртуальной машине. Итогом должны быть два работающих скрипта и созданные папки/файлы.

Запускаем виртуальную машину пароль от нее “oper”

1) Подготовка: создать рабочую папку для занятия

1. Открыть терминал в Debian 11.
2. Перейти в домашний каталог (обычно вы уже там, но сделаем явно):

```
cd ~
```

3. Создать папку для занятия:

```
mkdir -p OP01_P16
```

4. Перейти в неё:

```
cd OP01_P16
```

5. Проверить, что папка создана:

```
pwd  
ls -la
```

Ожидаемо: вы видите путь вида /home/<user>/OP01_P16 и пустую директорию.

```
oper1@debian:~$ cd  
oper1@debian:~$ mkdir -p OP01_P16  
oper1@debian:~$ cd OP01_P16  
oper1@debian:~/OP01_P16$ ls -la  
итого 8  
drwxr-xr-x  2 oper1 oper1 4096 фев 28 09:57 .  
drwxr-xr-x 16 oper1 oper1 4096 фев 28 09:57 ..
```

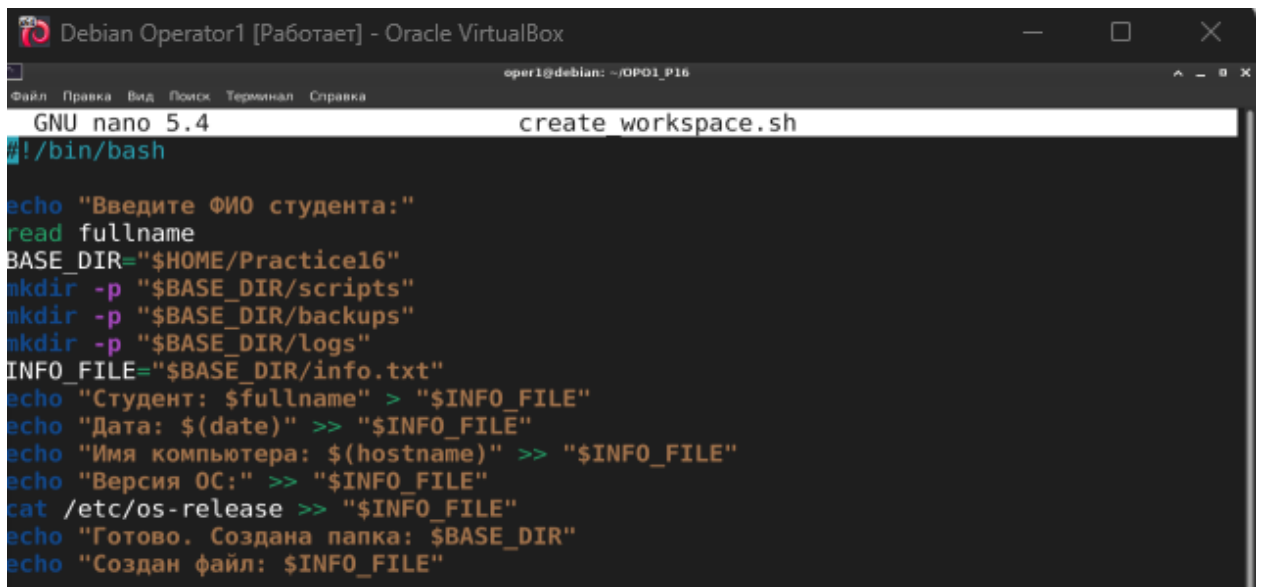
2) Скрипт №1: создание структуры Practice16 и отчёта

2.1 Создание файла скрипта

1. Находясь в папке ~/OP01_P16, создать файл:

```
nano create_workspace.sh
```

2. Впишем туда следующий код:



Сохранить файл нажать Ctrl + O и затем Enter

Выйти из nano для этого жмем Ctrl + X

Разберём построчно скрипт create_workspace.sh.

1.

```
#!/bin/bash
```

Указывает системе, что данный файл является исполняемым скриптом и должен запускаться через интерпретатор bash.

2.

```
echo "Введите ФИО студента:"
```

Выводит в терминал текстовое сообщение с просьбой ввести ФИО.

3.

```
read fullName
```

Считывает введенные пользователем данные и сохраняет их в переменную fullName.

4.

```
BASE_DIR="$HOME/Practice16"
```

Создаёт переменную BASE_DIR, в которой хранится путь к каталогу Practice16 в домашней директории пользователя.

Переменная \$HOME автоматически содержит путь к домашней папке текущего пользователя.

5.

```
mkdir -p "$BASE_DIR/scripts"
```

Создаёт папку scripts внутри каталога Practice16.

Ключ -p означает:

1. создать родительские каталоги при необходимости;
2. не выводить ошибку, если каталог уже существует.

6.

```
mkdir -p "$BASE_DIR/backups"
```

Создаёт папку backups внутри Practice16.

7.

```
mkdir -p "$BASE_DIR/logs"
```

Создаёт папку logs внутри Practice16.

8.

```
INFO_FILE="$BASE_DIR/info.txt"
```

Создаёт переменную INFO_FILE, в которой хранится путь к файлу info.txt.

9.

```
echo "Студент: $fullName" > "$INFO_FILE"
```

Записывает строку с ФИО студента в файл info.txt.

Оператор > создаёт новый файл или перезаписывает существующий.

10.

```
echo "Дата: $(date)" >> "$INFO_FILE"
```

Добавляет в файл текущую дату и время.

Команда \$(date) выполняется и её результат подставляется в строку.

Оператор >> добавляет текст в конец файла.

11.

```
echo "Имя компьютера: $(hostname)" >> "$INFO_FILE"
```

Добавляет в файл имя текущего компьютера.

Команда \$(hostname) возвращает имя системы.

12.

```
echo "Версия ОС:" >> "$INFO_FILE"
```

Добавляет заголовок перед информацией об операционной системе.

13.

```
cat /etc/os-release >> "$INFO_FILE"
```

Выводит содержимое файла /etc/os-release (в котором хранится информация о версии Linux) и добавляет его в info.txt.

14.

```
echo "Готово. Создана папка: $BASE_DIR"
```

Выводит сообщение о создании основной директории.

15.

```
echo "Создан файл: $INFO_FILE"
```

Выводит сообщение о создании файла info.txt с указанием его пути.

2.2 Назначение прав на выполнение

Проверить, что файл существует:

```
ls -la
```

Назначить права на запуск:

```
chmod +x create_workspace.sh
```

Проверить права:

```
ls -la create_workspace.sh
```

Ожидаемо: в начале строки будет -rwx... (есть x).

```
oper1@debian:~/OP01_P16$ ls -la
итого 12
drwxr-xr-x  2 oper1 oper1 4096 фев 28 10:06 .
drwxr-xr-x 17 oper1 oper1 4096 фев 28 10:03 ..
-rw-r--r--  1 oper1 oper1  547 фев 28 10:06 create_workspace.sh
oper1@debian:~/OP01_P16$ chmod +x create_workspace.sh
oper1@debian:~/OP01_P16$ ls -la create_workspace.sh
-rwxr-xr-x  1 oper1 oper1  547 фев 28 10:06 create_workspace.sh
```

2.3 Запуск скрипта №1

Запустить:

```
./create_workspace.sh
```

Скрипт попросит ввести ФИО — ввести и нажать Enter.

```
oper1@debian:~/OP01_P16$ ./create_workspace.sh
Введите ФИО студента:
Арестов Иван Сергеевич
Готово. Создана папка: /home/oper1/Practicel6
Создан файл: /home/oper1/Practicel6/info.txt
```

2.4 Проверка результата скрипта №1

1. Проверить, что появилась папка Practicel6:

```
ls -la ~
```

2. Посмотреть структуру:

```
ls -la ~/Practice16
ls -la ~/Practice16/scripts
ls -la ~/Practice16/backups
ls -la ~/Practice16/logs
```

```
operl@debian:~/OP01_P16$ ls -la ~/Practice16
итого 24
drwxr-xr-x  5 operl operl 4096 фев 28 10:08 .
drwxr-xr-x 18 operl operl 4096 фев 28 10:08 ..
drwxr-xr-x  2 operl operl 4096 фев 28 10:08 backups
-rw-r--r--  1 operl operl  424 фев 28 10:08 info.txt
drwxr-xr-x  2 operl operl 4096 фев 28 10:08 logs
drwxr-xr-x  2 operl operl 4096 фев 28 10:08 scripts
operl@debian:~/OP01_P16$ ls -la ~/Practice16/logs
итого 8
drwxr-xr-x 2 operl operl 4096 фев 28 10:08 .
drwxr-xr-x 5 operl operl 4096 фев 28 10:08 ..
```

3. Открыть и проверить содержимое отчёта:

```
cat ~/Practice16/info.txt
```

Ожидаемо: строки со студентом, датой, hostname и блок /etc/os-release.

```
operl@debian:~/OP01_P16$ cat ~/Practice16/info.txt
Студент: Арестов Иван Сергеевич
Дата: Сб 28 фев 2026 10:08:09 MSK
Имя компьютера: debian
Версия ОС:
PRETTY_NAME="Debian GNU/Linux 11 (bullseye)"
NAME="Debian GNU/Linux"
VERSION_ID="11"
VERSION="11 (bullseye)"
VERSION_CODENAME=bullseye
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
```

Скрипт №2: резервное копирование Practice16 с архивированием и логом

3.1 Создание файла скрипта

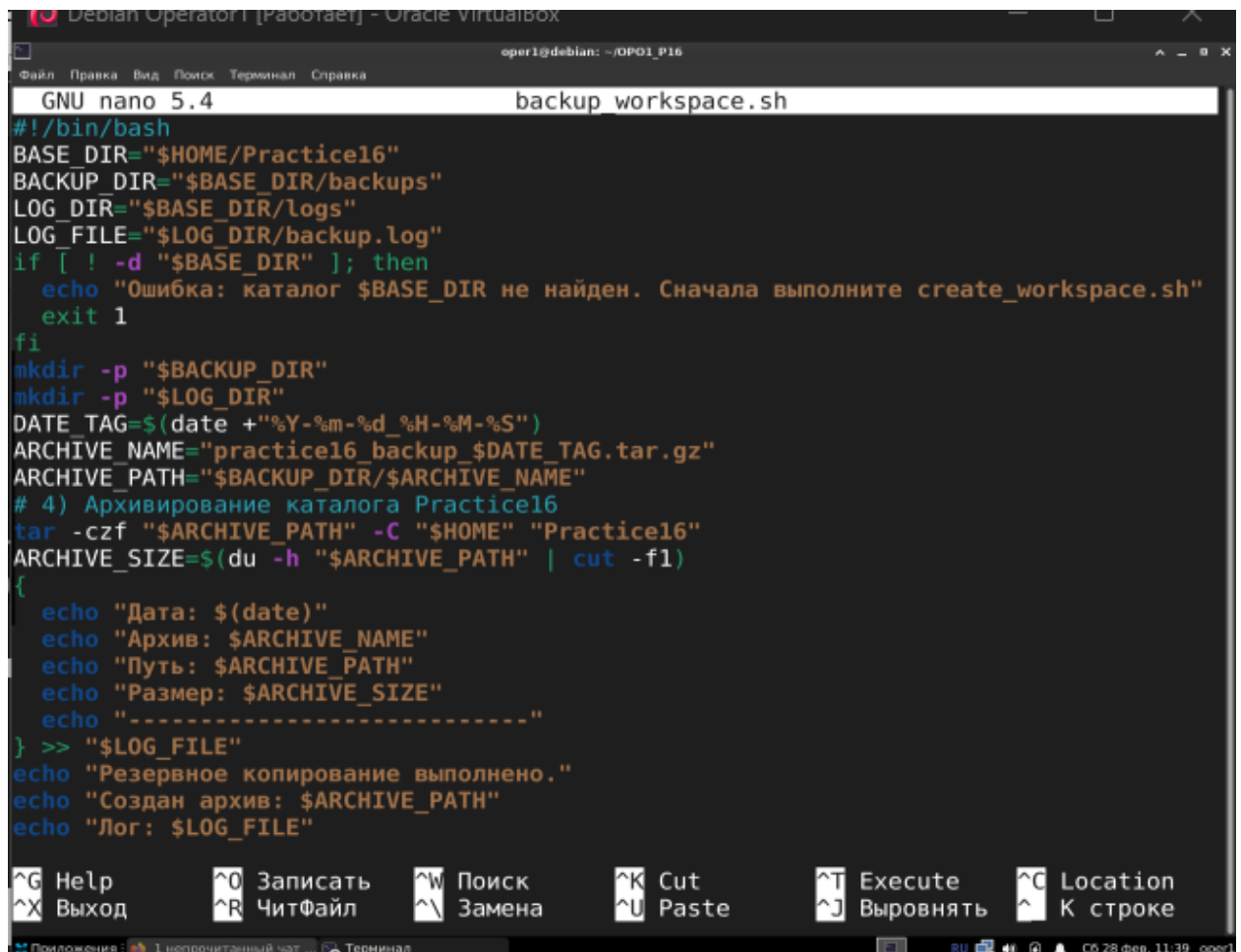
1. Вернуться в папку занятия:

```
cd ~/OP01_P16
```

2. Создать файл:

```
nano backup_workspace.sh
```

3. Впишем туда код:



```
GNU nano 5.4 backup_workspace.sh
#!/bin/bash
BASE_DIR="$HOME/Practice16"
BACKUP_DIR="$BASE_DIR/backups"
LOG_DIR="$BASE_DIR/logs"
LOG_FILE="$LOG_DIR/backup.log"
if [ ! -d "$BASE_DIR" ]; then
    echo "Ошибка: каталог $BASE_DIR не найден. Сначала выполните create_workspace.sh"
    exit 1
fi
mkdir -p "$BACKUP_DIR"
mkdir -p "$LOG_DIR"
DATE_TAG=$(date +"%Y-%m-%d_%H-%M-%S")
ARCHIVE_NAME="practice16_backup_${DATE_TAG}.tar.gz"
ARCHIVE_PATH="$BACKUP_DIR/$ARCHIVE_NAME"
# 4) Архивирование каталога Practice16
tar -czf "$ARCHIVE_PATH" -C "$HOME" "Practice16"
ARCHIVE_SIZE=$(du -h "$ARCHIVE_PATH" | cut -f1)
{
    echo "Дата: $(date)"
    echo "Архив: $ARCHIVE_NAME"
    echo "Путь: $ARCHIVE_PATH"
    echo "Размер: $ARCHIVE_SIZE"
    echo "-----"
} >> "$LOG_FILE"
echo "Резервное копирование выполнено."
echo "Создан архив: $ARCHIVE_PATH"
echo "Лог: $LOG_FILE"
```

Сохранить файл нажать Ctrl + O и затем Enter

Выйти из nano для этого жмем Ctrl + X

Разберём Bash-скрипт подробно и построчно — что делает каждая строка и какие механизмы Linux используются.

1. Указание интерпретатора

```
#!/bin/bash
```

Это **shebang**.

1. Говорит системе, что файл нужно запускать через интерпретатор `/bin/bash`.
2. Без этой строки скрипт может выполняться через другой shell (например `sh`), что иногда вызывает ошибки.

2. Переменные с путями

```
BASE_DIR="$HOME/Practice16"
```

1. `$HOME` — переменная среды (домашняя папка пользователя).
2. Формируется путь:

```
/home/username/Practice16
```

3. Сохраняется в переменную `BASE_DIR`.

```
BACKUP_DIR="$BASE_DIR/backups"
```

1. Создаёт путь к папке резервных копий.
2. Итог:

/home/username/Practice16/backups

```
LOG_DIR="$BASE_DIR/logs"
```

1. Путь к папке логов.

```
LOG_FILE="$LOG_DIR/backup.log"
```

1. Полный путь к файлу журнала.
2. Пример:

/home/username/Practice16/logs/backup.log

3. Проверка существования основной папки

```
if [ ! -d "$BASE_DIR" ]; then
```

Разберём:

1. if — условный оператор.
2. [...] — тестовое выражение.
3. -d — проверяет, существует ли каталог.
4. ! — отрицание.

То есть:

"Если папки Practice16 **не существует** — выполнить блок ниже".

```
echo "Ошибка: каталог $BASE_DIR не найден..."
```

1. echo выводит сообщение в консоль.
2. \$BASE_DIR подставляет значение переменной.

```
exit 1
```

1. Завершает выполнение скрипта.
2. 1 — код ошибки (0 = успех, любое другое число = ошибка).

4. Создание папок (если их нет)

```
mkdir -p "$BACKUP_DIR"
```

1. mkdir — создать каталог.
2. -p — создать родительские каталоги при необходимости и не выдавать ошибку, если уже существует.

То есть:

1. если папка backups отсутствует — создаётся,
2. если есть — ошибки не будет.

То же самое:

```
mkdir -p "$LOG_DIR"
```

5. Формирование имени архива

```
DATE_TAG=$(date +"%Y-%m-%d_%H-%M-%S")
```

Разбор:

1. date — команда получения текущей даты.
2. +"%Y-%m-%d_%H-%M-%S" — форматирование:
 1. %Y — год
 2. %m — месяц
 3. %d — день
 4. %H — часы
 5. %M — минуты
 6. %S — секунды
3. \$(...) — подстановка результата команды в переменную.

Пример результата:

2026-02-28_14-35-22

```
ARCHIVE_NAME="practice16_backup_$DATE_TAG.tar.gz"
```

Создаётся имя файла архива.

Пример:

practice16_backup_2026-02-28_14-35-22.tar.gz

```
ARCHIVE_PATH="$BACKUP_DIR/$ARCHIVE_NAME"
```

Формируется полный путь:

/home/user/Practice16/backups/practice16_backup_2026-02-28_14-35-22.tar.gz

6. Архивирование каталога

```
tar -czf "$ARCHIVE_PATH" -C "$HOME" "Practice16"
```

Разберём параметры tar:

1. -c — create (создать архив)
2. -z — сжать через gzip

3. -f — имя файла архива

`"$ARCHIVE_PATH"` — куда сохранить архив.

Что делает -C "\$HOME"?

1. -C означает "перейти в указанный каталог перед выполнением".
2. Здесь tar сначала "переходит" в домашнюю папку.
3. Затем архивирует папку Practice16.

Это нужно, чтобы внутри архива не было полного пути /home/user/..., а только Practice16.

7. Получение размера архива

```
ARCHIVE_SIZE=$(du -h "$ARCHIVE_PATH" | cut -f1)
```

Разберём:

du -h

1. du — показывает размер файла.
2. -h — в удобном формате (К, М, G).

Пример:

2.4M practice16_backup_...

| cut -f1

1. | — передаёт вывод в другую команду.
2. cut -f1 — берёт только первый столбец.

Итог:

2.4M

Сохраняется в ARCHIVE_SIZE.

8. Запись в лог-файл

```
{
  echo "Дата: $(date)"
  echo "Архив: $ARCHIVE_NAME"
  echo "Путь: $ARCHIVE_PATH"
  echo "Размер: $ARCHIVE_SIZE"
  echo "-----"
} >> "$LOG_FILE"
```

Разберём конструкцию:

`{ ... }`

Группировка команд.

>>

Оператор добавления в файл (append).

1. Если файл существует — запись добавляется в конец.
2. Если файла нет — он создаётся.

В лог записываются:

1. текущая дата
2. имя архива
3. путь
4. размер

9. Сообщения пользователю

```
echo "Резервное копирование выполнено."  
echo "Создан архив: $ARCHIVE_PATH"  
echo "Лог: $LOG_FILE"
```

1. Выводит результат выполнения.
2. Показывает, где лежит архив.
3. Показывает, где находится лог.

Итог работы скрипта

После выполнения:

1. Проверяется существование папки Practice16.
2. Создаются папки backups и logs (если их нет).
3. Формируется имя архива с датой.
4. Создаётся архив tar.gz.
5. Определяется размер архива.
6. Информация записывается в backup.log.
7. В консоль выводится итог.

3.2 Назначение прав на выполнение

```
chmod +x backup_workspace.sh  
ls -la backup_workspace.sh
```

```
oper1@debian:~/OP01_P16$ chmod +x backup_workspace.sh  
oper1@debian:~/OP01_P16$ ls -la backup_workspace.sh  
-rwxr-xr-x 1 oper1 oper1 948 фев 28 10:12 backup_workspace.sh
```

3.3 Запуск скрипта №2

Запустить:

```
./backup_workspace.sh
```

```
oper1@debian:~/OP01_P16$ ./backup_workspace.sh
Резервное копирование выполнено.
Создан архив: /home/oper1/Practice16/backups/practice16_backup_2026-02-28_10-12-42.tar.gz
Лог: /home/oper1/Practice16/logs/backup.log
```

3.4 Проверка результата скрипта №2

1. Проверить, что архив появился:

```
ls -la ~/Practice16/backups
```

2. Проверить лог:

```
cat ~/Practice16/logs/backup.log
```

3. Проверить содержимое архива (не распаковывая):

```
tar -tzf ~/Practice16/backups/*.tar.gz | head
```

Ожидаемо: выводятся пути вида Practice16/info.txt, Practice16/scripts/, Practice16/logs/ и т.д.

```
oper1@debian:~/OP01_P16$ ls -la ~/Practice16/backups/
итого 12
drwxr-xr-x 2 oper1 oper1 4096 фев 28 10:12 .
drwxr-xr-x 5 oper1 oper1 4096 фев 28 10:08 ..
-rw-r--r-- 1 oper1 oper1 502 фев 28 10:12 practice16_backup_2026-02-28_10-12-42.tar.gz
oper1@debian:~/OP01_P16$ cat ~/Practice16/logs/backup.log
Дата: Сб 28 фев 2026 10:12:42 MSK
Архив: practice16_backup_2026-02-28_10-12-42.tar.gz
Путь: /home/oper1/Practice16/backups/practice16_backup_2026-02-28_10-12-42.tar.gz
Размер: 4,0К
-----
oper1@debian:~/OP01_P16$ tar -tzf ~/Practice16/backups/*.tar.gz | head
Practice16/
Practice16/backups/
Practice16/scripts/
Practice16/logs/
Practice16/info.txt
```

4) Итог того, что должно получиться

После выполнения всех пунктов у вас должно быть:

1. Папка занятия:
 1. ~/OP01_P16/create_workspace.sh
 2. ~/OP01_P16/backup_workspace.sh
2. Рабочая структура:
 1. ~/Practice16/scripts
 2. ~/Practice16/backups
 3. ~/Practice16/logs
 4. ~/Practice16/info.txt

3. Резервные копии и журнал:

1. архив ~/Practice16/backups/practice16_backup_*.tar.gz
2. лог ~/Practice16/logs/backup.log

Задача 3. Расширенное резервное копирование и обслуживание архивов

В этой задаче выполняется усложнение сценария резервного копирования: скрипт должен принимать параметры, проверять свободное место, создавать архив, вести лог и удалять старые копии.

1) Подготовка рабочей папки для задачи 3

1. Открыть терминал.
2. Перейти в домашний каталог:

```
cd ~
```

3. Создать папку для задачи 3 и перейти в неё:

```
mkdir -p OP01_P16_Task3  
cd OP01_P16_Task3
```

4. Проверить:

```
pwd  
ls -la
```

2) Скрипт задачи 3: smart_backup.sh

2.1 Что делает скрипт

Скрипт smart_backup.sh должен:

1. Принимать параметры:
 1. -s путь к папке-источнику (что бэкапим)
 2. -d путь к папке назначения (куда сохраняем архивы)
 3. -k сколько дней хранить архивы (после удалить старые)
 4. -m минимально необходимое свободное место (в МБ)
2. Проверять, что источник существует.
3. Создавать папку назначения, если её нет.
4. Проверять свободное место на диске, где лежит папка назначения.
5. Создавать архив .tar.gz с датой/временем в имени.
6. Записывать лог в backup.log в папке назначения:
 1. дата
 2. что архивировали

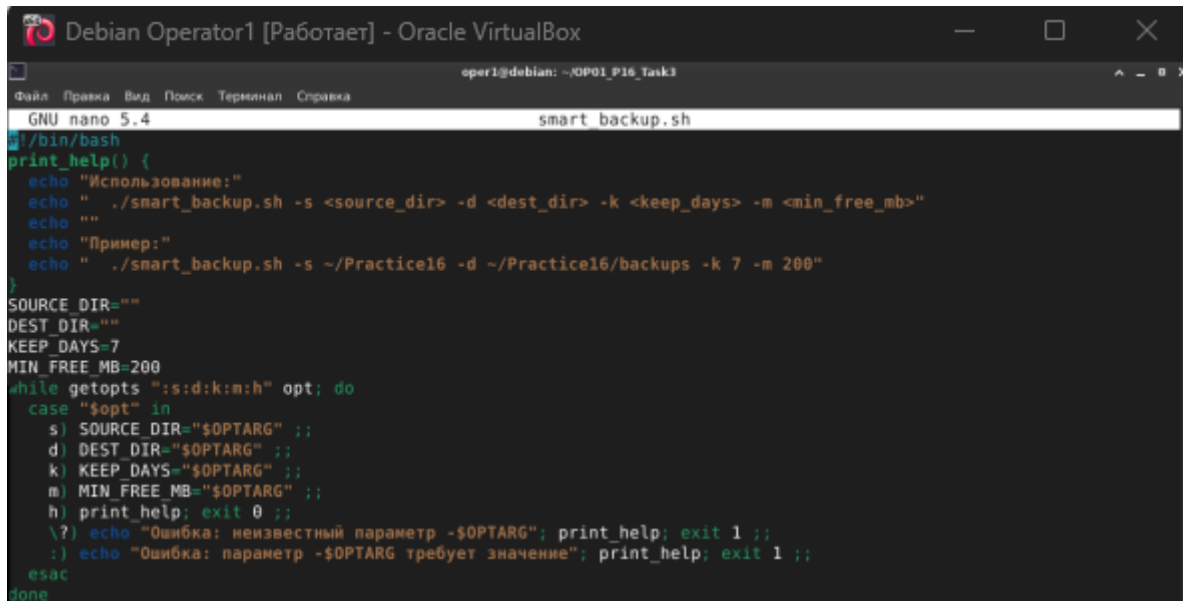
3. куда сохранили
4. размер архива
7. Удалять архивы старше k дней.
8. Выводить итоговое сообщение.

2.2 Создание файла скрипта

1. Создать файл:

```
nano smart_backup.sh
```

Напишем код для выполнения вышеуказанных задач:



```
Debian Operator1 [Работает] - Oracle VirtualBox
oper1@debian: ~/OP01_P16_Task1
GNU nano 5.4 smart_backup.sh
#!/bin/bash
print_help() {
    echo "Использование:"
    echo "  ./smart_backup.sh -s <source_dir> -d <dest_dir> -k <keep_days> -m <min_free_mb>"
    echo ""
    echo "Пример:"
    echo "  ./smart_backup.sh -s ~/Practice16 -d ~/Practice16/backups -k 7 -m 200"
}
SOURCE_DIR=""
DEST_DIR=""
KEEP_DAYS=7
MIN_FREE_MB=200
while getopts "s:d:k:m:h" opt; do
    case "$opt" in
        s) SOURCE_DIR="$OPTARG" ;;
        d) DEST_DIR="$OPTARG" ;;
        k) KEEP_DAYS="$OPTARG" ;;
        m) MIN_FREE_MB="$OPTARG" ;;
        h) print_help; exit 0 ;;
        \?) echo "Ошибка: неизвестный параметр -$OPTARG"; print_help; exit 1 ;;
        :) echo "Ошибка: параметр -$OPTARG требует значение"; print_help; exit 1 ;;
    esac
done
```

```
Debian Operator1 [Работает] - Oracle VirtualBox
oper1@debian: ~/OP01_P16_Task3
GNU nano 5.4 smart_backup.sh
if [ -z "$SOURCE_DIR" ] || [ -z "$DEST_DIR" ]; then
    echo "Ошибка: не заданы обязательные параметры -s и/или -d"
    print_help
    exit 1
fi
SOURCE_DIR="${SOURCE_DIR/#\~/~/}"
DEST_DIR="${DEST_DIR/#\~/~/}"
if [ ! -d "$SOURCE_DIR" ]; then
    echo "Ошибка: каталог-источник не найден: $SOURCE_DIR"
    exit 1
fi
mkdir -p "$DEST_DIR"
if [ [ "$KEEP_DAYS" == ^[0-9]+$ ]; then
    echo "Ошибка: -k должно быть целым числом"
    exit 1
fi
if [ [ "$MIN_FREE_MB" == ^[0-9]+$ ]; then
    echo "Ошибка: -m должно быть целым числом"
    exit 1
fi
FREE_MB=$(df -Pm "$DEST_DIR" | awk 'NR==2 {print $4}')
if [ -z "$FREE_MB" ]; then
    echo "Ошибка: не удалось определить свободное место"
    exit 1
fi
if [ "$FREE_MB" -lt "$MIN_FREE_MB" ]; then
    echo "Ошибка: недостаточно места. Свободно: ${FREE_MB}MB, требуется: ${MIN_FREE_MB}MB"
    exit 1
fi
DATE_TAG=$(date +"%Y-%m-%d_%H-%M-%S")
BASE_NAME=$(basename "$SOURCE_DIR")
ARCHIVE_NAME="${BASE_NAME}_backup_${DATE_TAG}.tar.gz"
ARCHIVE_PATH="$DEST_DIR/$ARCHIVE_NAME"
PARENT_DIR=$(dirname "$SOURCE_DIR")
tar -czf "$ARCHIVE_PATH" -C "$PARENT_DIR" "$BASE_NAME"
TAR_CODE=$?
if [ "$TAR_CODE" -ne 0 ]; then
    echo "Ошибка создания архива"
    exit 1
fi
^G Help      ^O Записать  ^W Поиск     ^K Cut       ^T Execute   ^C Location  M-U Отмена
^X Выход     ^R ЧитФайл  ^N Замена    ^U Paste     ^J Выводить  ^_ К строке  M-E Повтор
Приложения: 1 непрочитанный чат... oper1@debian: ~/OP01...
```

```
fi
ARCHIVE_SIZE=$(du -h "$ARCHIVE_PATH" | awk '{print $1}')
LOG_FILE="$DEST_DIR/backup.log"
{
    echo "Дата: $(date)"
    echo "Источник: $SOURCE_DIR"
    echo "Назначение: $DEST_DIR"
    echo "Архив: $ARCHIVE_NAME"
    echo "Размер: $ARCHIVE_SIZE"
    echo "Свободно места (MB): $FREE_MB"
    echo "Хранить дней: $KEEP_DAYS"
    echo "-----"
} >> "$LOG_FILE"
DELETED_COUNT=$(find "$DEST_DIR" -maxdepth 1 -type f -name "*.tar.gz" -mtime +"$KEEP_DAYS" -print -delete 2>/dev/null)
echo "Резервное копирование выполнено."
echo "Создан архив: $ARCHIVE_PATH"
echo "Лог: $LOG_FILE"
echo "Удалено старых архивов: $DELETED_COUNT"
^G Help      ^O Записать  ^W Поиск     ^K Cut       ^T Execute   ^C Location  M-U Отмена
^X Выход     ^R ЧитФайл  ^N Замена    ^U Paste     ^J Выводить  ^_ К строке  M-E Повтор
Приложения: 1 непрочитанный чат... oper1@debian: ~/OP01...
```

Давайте разберем код smart_backup.sh: что делает каждый блок и ключевые строки.

1) Интерпретатор

```
#!/bin/bash
```

Указывает системе запускать файл через Bash.

2) Функция справки

```
print_help() {
    echo "Использование:"
    echo " ./smart_backup.sh -s <source_dir> -d <dest_dir> -k <keep_days> -m <min_free_mb>"
    echo ""
    echo "Пример:"
    echo " ./smart_backup.sh -s ~/Practice16 -d ~/Practice16/backups -k 7 -m 200"
}
```

1. `print_help()` — функция, которую можно вызвать из скрипта.
2. `echo` печатает подсказку: какие параметры нужны и пример запуска.
3. Используется, когда пользователь ошибся с параметрами или просит `-h`.

3) Переменные по умолчанию

```
SOURCE_DIR=""
DEST_DIR=""
KEEP_DAYS=7
MIN_FREE_MB=200
```

1. `SOURCE_DIR` — папка-источник (что архивируем). Пока пустая, т.к. обязана прийти из `-s`.
2. `DEST_DIR` — папка назначения (куда класть архивы). Пока пустая, обязана прийти из `-d`.
3. `KEEP_DAYS=7` — сколько дней хранить архивы (если пользователь не передал `-k`, будет 7).
4. `MIN_FREE_MB=200` — минимально допустимое свободное место на диске для папки назначения (если не передали `-m`, будет 200MB).

4) Разбор параметров командной строки (getopts)

```
while getopts "s:d:k:m:h" opt; do
    case "$opt" in
        s) SOURCE_DIR="$OPTARG" ;;
        d) DEST_DIR="$OPTARG" ;;
        k) KEEP_DAYS="$OPTARG" ;;
        m) MIN_FREE_MB="$OPTARG" ;;
        h) print_help; exit 0 ;;
        \?) echo "Ошибка: неизвестный параметр -$OPTARG"; print_help; exit 1 ;;
        :) echo "Ошибка: параметр -$OPTARG требует значение"; print_help; exit 1 ;;
    esac
done
```

Что здесь происходит:

1. `getopts` читает ключи `-s`, `-d`, `-k`, `-m`, `-h`.
2. `opt` — переменная, куда попадает текущий найденный ключ.

3. OPTARG — значение ключа (например, у -s ~/Practice16 значение ~/Practice16 попадёт в OPTARG).

Особенности строки ":s:d:k:m:h":

1. Двоеточие в начале : включает «тихий режим», чтобы мы сами обрабатывали ошибки.
2. s: означает, что -s требует аргумент.
3. d: — требует аргумент.
4. k: — требует аргумент.
5. m: — требует аргумент.
6. h — не требует аргумент.

В case:

1. s) — присваиваем SOURCE_DIR.
2. d) — присваиваем DEST_DIR.
3. k) — присваиваем KEEP_DAYS.
4. m) — присваиваем MIN_FREE_MB.
5. h) — печатаем справку и выходим с кодом успеха 0.
6. \?) — неизвестный ключ → ошибка + help + exit 1.
7. :) — ключ без значения (например -s без пути) → ошибка + help + exit 1.

5) Проверка обязательных параметров

```
if [ -z "$SOURCE_DIR" ] || [ -z "$DEST_DIR" ]; then
    echo "Ошибка: не заданы обязательные параметры -s и/или -d"
    print_help
    exit 1
fi
```

1. -z проверяет, что строка пустая.
2. Если не задан источник или назначение — скрипт останавливается и показывает справку.

6) Замена ~ на реальный путь \$HOME

```
SOURCE_DIR="${SOURCE_DIR/#~/ $HOME}"
DEST_DIR="${DEST_DIR/#~/ $HOME}"
```

Это «подстановка в переменной» (parameter expansion):

1. \${VAR/#pattern/replacement} заменяет pattern только **в начале строки**.
2. \~ — символ ~ (экранирован).

3. Если пользователь передал ~/Practice16, получится /home/user/Practice16.

7) Проверка существования источника

```
if [ ! -d "$SOURCE_DIR" ]; then
    echo "Ошибка: каталог-источник не найден: $SOURCE_DIR"
    exit 1
fi
```

1. -d проверяет, что путь существует и это папка.
2. ! — отрицание: если папки нет → ошибка и выход.

8) Создание папки назначения

```
mkdir -p "$DEST_DIR"
```

1. Создаёт папку назначения, если её нет.
2. -p не ругается, если уже существует, и создаёт промежуточные папки.

9) Проверка, что KEEP_DAYS — число

```
if ! [[ "$KEEP_DAYS" =~ ^[0-9]+$ ]]; then
    echo "Ошибка: -k должно быть целым числом"
    exit 1
fi
```

1. [[...]] — расширенная проверка Bash.
2. =~ — проверка по регулярному выражению.
3. ^[0-9]+\$ — только цифры от начала до конца строки.
4. ! — если не число → ошибка.

10) Проверка, что MIN_FREE_MB — число

```
if ! [[ "$MIN_FREE_MB" =~ ^[0-9]+$ ]]; then
    echo "Ошибка: -m должно быть целым числом"
    exit 1
fi
```

Аналогично: защищает от ввода вроде -m abc.

11) Определение свободного места на диске (в MB)

```
FREE_MB=$(df -Pm "$DEST_DIR" | awk 'NR==2 {print $4}')
```

Разбор:

1. df показывает использование диска.
2. -P — «POSIX формат» (стабильный вывод по колонкам).
3. -m — вывод в мегабайтах.

4. "\$DEST_DIR" — смотрим раздел, где находится папка назначения.

```
awk 'NR==2 {print $4}':
```

1. NR==2 — вторая строка (первая — заголовок).
2. \$4 — 4-я колонка, обычно это **Available** (доступно).

Результат (число) сохраняется в FREE_MB.

12) Проверка, что FREE_MB получено

```
if [ -z "$FREE_MB" ]; then
    echo "Ошибка: не удалось определить свободное место"
    exit 1
fi
```

Если df|awk ничего не вернул — прекращаем работу.

13) Проверка достаточного места

```
if [ "$FREE_MB" -lt "$MIN_FREE_MB" ]; then
    echo "Ошибка: недостаточно места. Свободно: ${FREE_MB}MB, требуется:
    ${MIN_FREE_MB}MB"
    exit 1
fi
```

1. -lt — «меньше чем» для чисел.
2. Если свободного места меньше требуемого → ошибка.

14) Генерация имени архива

```
DATE_TAG=$(date +"%Y-%m-%d_%H-%M-%S")
BASE_NAME=$(basename "$SOURCE_DIR")
ARCHIVE_NAME="${BASE_NAME}_backup_${DATE_TAG}.tar.gz"
ARCHIVE_PATH="$DEST_DIR/$ARCHIVE_NAME"
```

1. DATE_TAG — метка даты/времени для уникальности.
2. basename "\$SOURCE_DIR" — берёт имя папки без пути.
 1. /home/user/Practice16 → Practice16
3. ARCHIVE_NAME — имя архива вида Practice16_backup_2026-02-28_12-30-00.tar.gz
4. ARCHIVE_PATH — полный путь, куда сохранить архив.

15) Определение родительской папки источника

```
PARENT_DIR=$(dirname "$SOURCE_DIR")
```

1. dirname возвращает путь без последнего элемента.
2. Нужно, чтобы tar -C работал правильно.

Пример:

1. SOURCE_DIR=/home/user/Practice16
2. PARENT_DIR=/home/user

16) Создание архива

```
tar -czf "$ARCHIVE_PATH" -C "$PARENT_DIR" "$BASE_NAME"
TAR_CODE=$?
```

1. tar -c — создать архив
2. -z — сжать gzip
3. -f "\$ARCHIVE_PATH" — файл архива
4. -C "\$PARENT_DIR" — «перейти» в родительский каталог
5. "\$BASE_NAME" — архивируем папку по имени

Это делает структуру внутри архива красивой: Practice16/..., а не полный путь /home/....

```
TAR_CODE=$?:
```

1. \$? — код завершения последней команды.
2. 0 = успех, не 0 = ошибка.
3. Сохраняем в TAR_CODE для проверки.

17) Проверка, что архив создался без ошибок

```
if [ $TAR_CODE -ne 0 ]; then
    echo "Ошибка создания архива"
    exit 1
fi
```

1. -ne — «не равно» (числовое).
2. Если tar вернул ошибку — прекращаем скрипт.

18) Получение размера архива

```
ARCHIVE_SIZE=$(du -h "$ARCHIVE_PATH" | awk '{print $1}')
```

1. du -h — размер файла в удобном формате (K/M/G).
2. awk '{print \$1}' — берём только размер (первую колонку).
3. Результат сохраняется в ARCHIVE_SIZE.

19) Логирование в файл backup.log

```
LOG_FILE="$DEST_DIR/backup.log"
{
    echo "Дата: $(date)"
    echo "Источник: $SOURCE_DIR"
```

```
echo "Назначение: $DEST_DIR"
echo "Архив: $ARCHIVE_NAME"
echo "Размер: $ARCHIVE_SIZE"
echo "Свободно места (MB): $FREE_MB"
echo "Хранить дней: $KEEP_DAYS"
echo "-----"
} >> "$LOG_FILE"
```

1. LOG_FILE — лог хранится в папке назначения.
2. { ... } >> file — группа команд, вывод всех строк **добавляется** в конец файла.
3. В лог пишется:
 1. дата
 2. источник и назначение
 3. имя архива, размер
 4. свободное место
 5. срок хранения

20) Удаление старых архивов

```
DELETED_COUNT=$(find "$DEST_DIR" -maxdepth 1 -type f -name "*.tar.gz" -mtime +"$KEEP_DAYS" -print -delete 2>/dev/null | wc -l)
```

Это самый «умный» блок.

Разбор по частям:

1. find "\$DEST_DIR" — ищем в папке назначения
2. -maxdepth 1 — только в этой папке, без подпапок
3. -type f — только файлы
4. -name "*.tar.gz" — только архивы tar.gz
5. -mtime +"\$KEEP_DAYS" — файлы старше указанного количества дней
6. -print — вывести найденные файлы (это будет считаться в wc -l)
7. -delete — удалить найденные файлы
8. 2>/dev/null — скрыть сообщения об ошибках (например, нет прав)
9. | wc -l — посчитать количество строк, то есть сколько файлов удалено

Результат сохраняется в DELETED_COUNT.

21) Итоговый вывод пользователю

```
echo "Резервное копирование выполнено."
echo "Создан архив: $ARCHIVE_PATH"
```

```
echo "Лог: $LOG_FILE"
echo "Удалено старых архивов: $DELETED_COUNT"
```

Печатает, что сделано:

1. где архив
2. где лог
3. сколько старых архивов удалено

Итог: что делает smart_backup.sh целиком

1. Принимает параметры: источник, назначение, срок хранения, минимум свободного места.
2. Проверяет правильность входных данных.
3. Проверяет, что источник существует.
4. Проверяет свободное место на диске назначения.
5. Создает архив .tar.gz с датой/временем.
6. Пишет подробный лог.
7. Удаляет старые архивы.
8. Печатает итог.

3) Назначить права на выполнение

Команда 1

```
chmod +x smart_backup.sh
```

Что делает:

1. `chmod` — изменяет права доступа к файлу.
2. `+x` — добавляет право **выполнения** (execute) для файла.
3. После этого файл можно запускать как программу командой `./smart_backup.sh`.

Зачем нужно:

1. В Linux файл со скриптом по умолчанию часто не является исполняемым.
2. Без `chmod +x` при запуске будет ошибка типа:
 1. Permission denied

Команда 2

```
ls -la smart_backup.sh
```

Что делает:

1. `ls` — показывает файлы.

2. `-l` — подробный формат (права, владелец, размер, дата).
3. `-a` — показать все файлы (включая скрытые). Для одного файла не критично, но привычно.

Что смотреть в выводе:

1. В начале строки должны появиться буквы `x` (execute), например:

1. `-rwxr-xr-x ... smart_backup.sh`

Расшифровка:

1. `r` — чтение (read)
2. `w` — запись (write)
3. `x` — выполнение (execute)

```
oper1@debian:~/OP01_P16_Task3$ ls -la smart_backup.sh
-rwxr-xr-x 1 oper1 oper1 2893 фев 28 10:17 smart_backup.sh
```

4) Запуск (пример для структуры Practice16)

```
./smart_backup.sh -s ~/Practice16 -d ~/Practice16/backups -k 7 -m 200
```

```
oper1@debian:~/OP01_P16_Task3$ ./smart_backup.sh -s ~/Practice16 -d ~/Practice16/backups -k 7 -m 200
Резервное копирование выполнено.
Создан архив: /home/oper1/Practice16/backups/Practice16_backup_2026-02-28_10-18-27.tar.gz
Лог: /home/oper1/Practice16/backups/backup.log
Удалено старых архивов: 0
```

Разбор по частям:

```
./smart_backup.sh
```

1. `./` означает: “запусти файл из текущей папки”.
2. В Linux текущая папка **не запускается автоматически**, поэтому нужно `./`.

```
-s ~/Practice16
```

1. `-s` (source) — папка-источник, которую нужно архивировать.
2. `~/Practice16` — путь в домашней папке пользователя:

1. `~` = `/home/username`

```
-d ~/Practice16/backups
```

1. `-d` (destination) — папка, куда сохранить архивы и лог.
2. Если её нет, скрипт создаст (через `mkdir -p`).

```
-k 7
```

1. `-k` (keep days) — сколько дней хранить архивы.
2. Архивы старше 7 дней будут удаляться.

```
-m 200
```

1. -m (min free MB) — минимально допустимое свободное место на диске назначения.
2. Если свободно меньше 200 MB — скрипт завершится с ошибкой и архив не создаст.

Ожидаемый вывод:

1. Сообщение, что бэкап выполнен, где архив, где лог, сколько старых архивов удалено.

```
oper1@debian:~/OP01_P16_Task3$ ./smart_backup.sh -s ~/Practice16 -d ~/Practice16/backups -k 7 -m 200
Резервное копирование выполнено.
Создан архив: /home/oper1/Practice16/backups/Practice16_backup_2026-02-28_10-18-27.tar.gz
Лог: /home/oper1/Practice16/backups/backup.log
Удалено старых архивов: 0
```

5) Проверка результата

5.1 Убедиться, что архив появился

```
ls -la ~/Practice16/backups | tail
```

```
oper1@debian:~/OP01_P16_Task3$ ls -la ~/Practice16/backups | tail
итого 20
drwxr-xr-x 2 oper1 oper1 4096 фев 28 10:18 .
drwxr-xr-x 5 oper1 oper1 4096 фев 28 10:08 ..
-rw-r--r-- 1 oper1 oper1 309 фев 28 10:18 backup.log
-rw-r--r-- 1 oper1 oper1 502 фев 28 10:12 practice16_backup_2026-02-28_10-12-42.tar.gz
-rw-r--r-- 1 oper1 oper1 1212 фев 28 10:18 Practice16_backup_2026-02-28_10-18-27.tar.gz
```

Разбор:

1. ls -la — список файлов в папке backups подробно.
2. | tail — показывает последние строки списка (обычно самые свежие файлы в конце, но зависит от сортировки).

Что должно быть:

1. Файл типа:
 1. Practice16_backup_2026-02-28_14-35-22.tar.gz
 2. или practice16_backup_...tar.gz (зависит от имени в скрипте)
2. Плюс файл backup.log.

Если архива нет:

1. значит скрипт завершился с ошибкой (часто — неверный путь, нет места, нет прав).

```
oper1@debian:~/OP01_P16_Task3$ ls -la ~/Practice16/backups | tail
итого 20
drwxr-xr-x 2 oper1 oper1 4096 фев 28 10:18 .
drwxr-xr-x 5 oper1 oper1 4096 фев 28 10:08 ..
-rw-r--r-- 1 oper1 oper1 309 фев 28 10:18 backup.log
-rw-r--r-- 1 oper1 oper1 502 фев 28 10:12 practice16_backup_2026-02-28_10-12-42.tar.gz
-rw-r--r-- 1 oper1 oper1 1212 фев 28 10:18 Practice16_backup_2026-02-28_10-18-27.tar.gz
```

5.2 Посмотреть лог

```
cat ~/Practice16/backups/backup.log
```

Разбор:

1. cat выводит содержимое файла в терминал.

Что должно быть в логе:

1. Дата выполнения
2. Источник
3. Назначение
4. Имя архива
5. Размер
6. Сколько дней хранить
7. Разделитель -----

И таких блоков может быть несколько (после каждого запуска добавляется запись).

```
oper1@debian:~/OP01_P16_Task3$ cat ~/Practice16/backups/backup.log
Дата: 06 28 фев 2026 10:18:27 MSK
Источник: /home/oper1/Practice16
Назначение: /home/oper1/Practice16/backups
Архив: Practice16_backup_2026-02-28_10-18-27.tar.gz
Размер: 4,0К
Свободно места (МБ): 13016
Хранить дней: 7
.....
```

5.3 Проверить содержимое архива, не распаковывая

```
tar -tzf ~/Practice16/backups/*.tar.gz | head
```

Разбор:

1. tar — работа с tar-архивами.
2. -t — вывести список содержимого (test/list), не извлекая.
3. -z — архив сжат gzip (.tar.gz).
4. -f — указать файл архива.
5. ~/Practice16/backups/*.tar.gz — взять все архивы .tar.gz в папке.
6. | head — показать только первые 10 строк, чтобы не выводить огромный список.

Ожидаемо:

1. Появятся пути файлов и папок внутри архива, например:
 1. Practice16/info.txt
 2. Practice16/Scripts/...
 3. Practice16/Logs/...

```
oper1@debian:~/OP01_P16_Task3$ tar -tzf ~/Practice16/backups/Practice16_backup_2026-02-28_10-18-27.tar.gz | head
Practice16/
Practice16/backups/
Practice16/backups/practicel6_backup_2026-02-28_10-12-42.tar.gz
Practice16/scripts/
Practice16/logs/
Practice16/logs/backup.log
Practice16/info.txt
```

5.4 Контрольный тест ошибки: неверный источник

```
./smart_backup.sh -s ~/NO_SUCH_DIR -d ~/Practice16/backups -k 7 -m 200
```

Что проверяем:

1. Как скрипт реагирует на неправильный ввод.

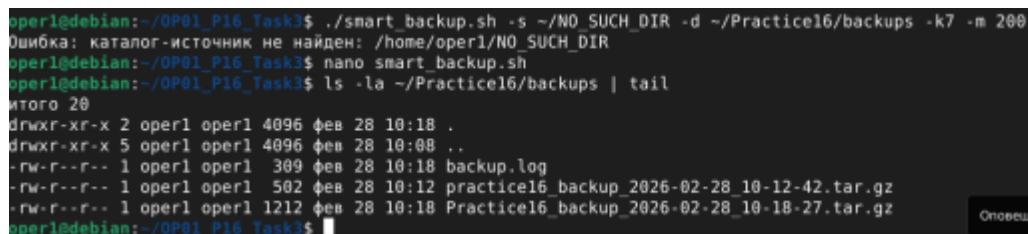
Ожидаемо:

1. Сообщение вида:
 1. Ошибка: каталог-источник не найден: ...
2. Архив **не создаётся**.
3. Лог либо не пополняется, либо (если добавляли логирование ошибок) будет запись об ошибке.

Как убедиться, что архива не появилось:

1. снова выполнить:

```
ls -la ~/Practice16/backups | tail
```



```
operl@debian:~/OP01_P16_Task3$ ./smart_backup.sh -s ~/NO_SUCH_DIR -d ~/Practice16/backups -k7 -m 200
Ошибка: каталог-источник не найден: /home/operl/NO_SUCH_DIR
operl@debian:~/OP01_P16_Task3$ nano smart_backup.sh
operl@debian:~/OP01_P16_Task3$ ls -la ~/Practice16/backups | tail
итого 20
drwxr-xr-x 2 operl operl 4096 фев 28 10:18 .
drwxr-xr-x 5 operl operl 4096 фев 28 10:08 ..
-rw-r--r-- 1 operl operl 309 фев 28 10:18 backup.log
-rw-r--r-- 1 operl operl 502 фев 28 10:12 practice16_backup_2026-02-28_10-12-42.tar.gz
-rw-r--r-- 1 operl operl 1212 фев 28 10:18 Practice16_backup_2026-02-28_10-18-27.tar.gz
operl@debian:~/OP01_P16_Task3$
```

6) Ожидаемый результат задачи 3

В результате мы получили:

1. один или несколько архивов вида Practice16_backup_YYYY-MM-DD_HH-MM-SS.tar.gz
2. лог-файл backup.log с историей операций
3. понимание, как делается автоматизация резервного копирования в Bash:
 1. разбор параметров (getopts)
 2. проверки (if, коды возврата)
 3. работа с файловой системой
 4. tar/df/find/du/awk

В ходе выполнения практической работы были разработаны и протестированы скрипты автоматизации в операционных системах Linux (Bash) и Windows (PowerShell).

Было реализовано:

1. автоматическое создание структуры каталогов;
2. формирование системного отчёта (информация о пользователе и ОС);

3. создание архивов резервных копий;
4. ведение журнала операций (логирование);
5. проверка входных параметров и обработка ошибок;
6. контроль свободного места и удаление старых архивов (в расширенной версии).

В результате студент получил:

1. рабочие скрипты резервного копирования;
2. понимание принципов автоматизации в Bash и PowerShell;
3. практический опыт работы с файловой системой и встроенными средствами ОС;
4. навыки обработки параметров, проверок и логирования.

Практическая работа позволила закрепить навыки использования встроенных инструментов Windows и Linux для автоматизации системных задач.