**Задание 1:**

Напиши bash-скрипт, который принимает на вход путь к каталогу и выводит список всех файлов в этом каталоге (и его подкаталогах), отсортированных по размеру в порядке убывания.

**Детали:**

* Скрипт должен принимать путь к каталогу как аргумент ($1).
* Если путь не указан, использовать текущий каталог (".").
* Для поиска файлов используй команду find.
* Для получения размера файлов и их сортировки используй команды du, sort и ls -lh.
* Вывод должен показывать имя файла и его размер в читаемом формате (например, 1K, 10M, 2G).

**Подсказки:**

* Используй конструкцию if [ -z "$1" ]; then ... fi для проверки, передан ли аргумент.
* Пример команды для поиска файлов: find "$dir" -type f.
* Пример команды для отображения размера: du -h.

**Алгоритм:**

1. В Visual Studio Code создан новый файл list\_files\_by\_size.sh

2. Написан и сохранен bash-скрипт:

#!/bin/bash

# Проверяем, передан ли аргумент (путь к каталогу)

if [ -z "$1" ]; then

    dir="."

else

    dir="$1"

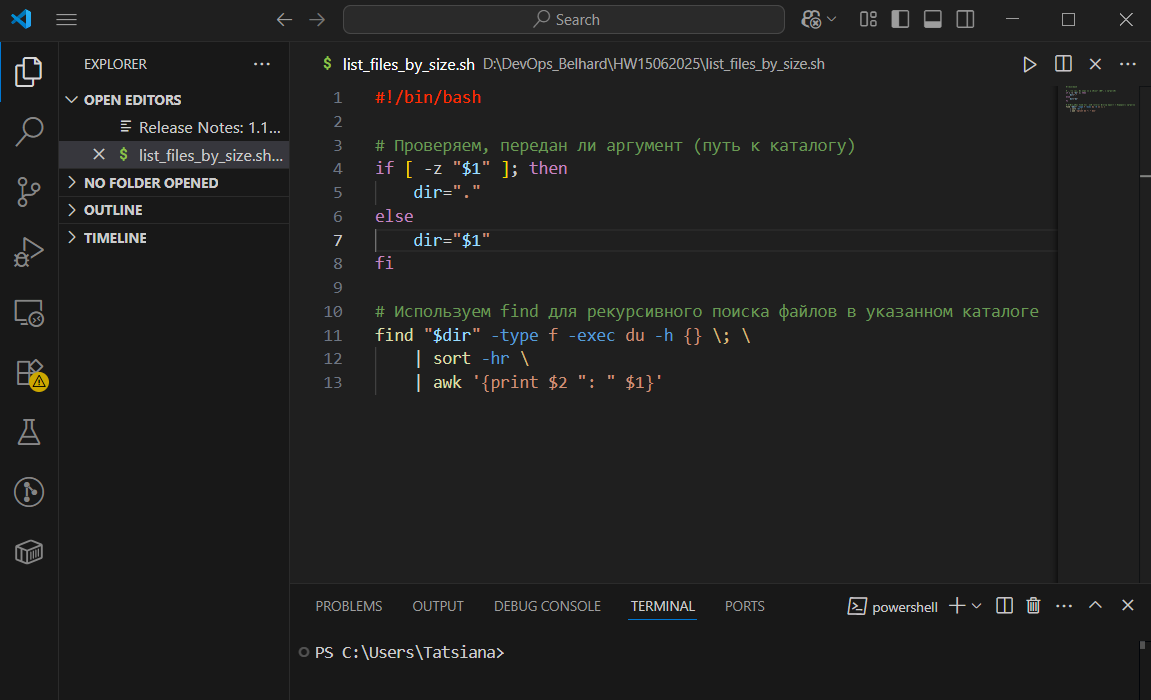
fi

# Используем find для рекурсивного поиска файлов в указанном каталоге

find "$dir" -type f -exec du -h {} \; \

    | sort -hr \

    | awk '{print $2 ": " $1}'



3. В WSL:

Переход в директорию:

cd /mnt/d/DevOps\_Belhard/HW15062025

Назначение файла исполняемым:

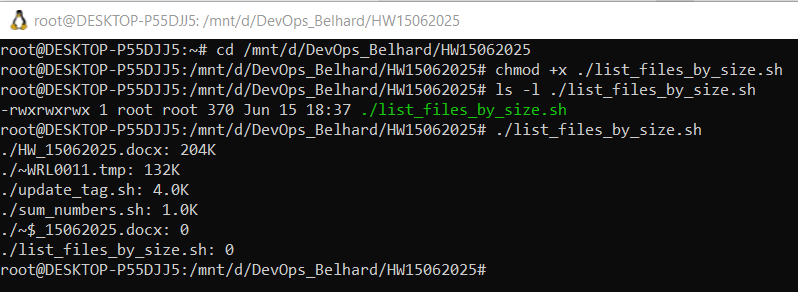
chmod +x ./list\_files\_by\_size.sh

Проверка:

ls -l ./list\_files\_by\_size.sh

Запуск скрипта:

./list\_files\_by\_size.sh



**Задание 2 (с использованием цикла while):**

Напиши bash-скрипт, который:

1. Читает числа из стандартного ввода (построчно).
2. Складывает их в сумму.
3. Выводит сумму после ввода числа 0.

**Детали:**

* Используй цикл while read для чтения чисел.
* Завершай цикл, если введено 0.
* Вводить числа можно через терминал или перенаправление из файла.

**Подсказки:**

* Для сложения чисел используй конструкцию sum=$((sum + number)).
* Для проверки окончания: if [ "$number" -eq 0 ]; then break; fi.
* Для начала суммы: sum=0.

**Пример ввода**:

Введите числа построчно (для завершения введите 0):

5

10

-3

8

0

Сумма всех введённых чисел: 20

**Алгоритм:**

1. В Visual Studio Code создан новый файл sum\_numbers.sh

2. Написан и сохранен bash-скрипт:

#!/bin/bash

# Начинаем с нулевой суммы

sum=0

# Читаем числа построчно из стандартного ввода

while read -p "Введите число (для завершения введите 0): " number; do

    # Если введено 0, заканчиваем цикл

    if [ "$number" -eq 0 ]; then

        break

    fi

    # Суммируем введённое число

    sum=$((sum + number))

done

# Выводим результат

echo "Сумма всех введённых чисел: $sum"

3. В WSL:

Переход в директорию:

cd /mnt/d/DevOps\_Belhard/HW15062025

Установка утилиты dos2unix:

sudo apt update && sudo apt install dos2unix

Преобразование файл в Unix-формат:

dos2unix sum\_numbers.sh

Назначение файла исполняемым:

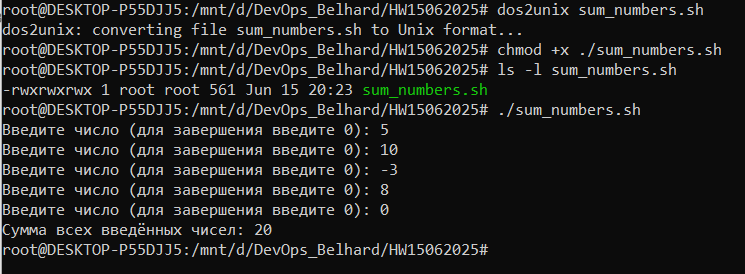
chmod +x ./sum\_numbers.sh

Проверка прав доступа к файлу:

ls -l sum\_numbers.sh

Запуск скрипта:

./sum\_numbers.sh



**Задание 3 (Такой же скрипт, как писали на powershell ранее)**

Напиши bash-скрипт, который:

1. Клонирует указанный Git-репозиторий по URL.
2. Переходит в директорию проекта.
3. Проверяет текущий тег с помощью git describe --tags.
4. Определяет, есть ли новые коммиты после последнего тега.
5. Если тег указывает на самый последний коммит, выводит сообщение “No changes.”
6. Если после тега есть коммиты:
   * Увеличивает патч-версию (например, с 1.0.0 до 1.0.1).
   * Создает новый аннотированный тег.
   * Пушит новый тег в удалённый репозиторий.

После выполнения удаляет локальную копию репозитория (директорию).

**Алгоритм:**

1. В Visual Studio Code создан новый файл update\_tag.sh

2. Написан и сохранен bash-скрипт:

#!/bin/bash

# Получаем URL репозитория из аргумента (обязательно!)

if [ -z "$1" ]; then

    echo "Usage: $0 <repository-url>"

    exit 1

fi

REPO\_URL="$1"

# Клонируем репозиторий

TEMP\_DIR="/tmp/update-tag-tmp"

rm -rf "$TEMP\_DIR"

git clone "$REPO\_URL" "$TEMP\_DIR"

# Переходим в директорию проекта

cd "$TEMP\_DIR" || exit 1

# Получаем текущий тег

CURRENT\_TAG=$(git describe --tags --abbrev=0)

# Получаем последний коммит

LAST\_COMMIT=$(git rev-list HEAD --max-count=1)

# Получаем коммит, на который указывает тег

TAG\_COMMIT=$(git rev-list "$CURRENT\_TAG" --max-count=1)

# Проверяем, есть ли новые коммиты после тега

if [[ "$LAST\_COMMIT" == "$TAG\_COMMIT" ]]; then

    echo "No changes."

else

    # Увеличиваем patch-версию (увеличиваем третью часть номера версии)

    VERSION\_PARTS=(${CURRENT\_TAG//./ })

    PATCH\_VERSION=${VERSION\_PARTS[2]}

    NEW\_PATCH\_VERSION=$(($PATCH\_VERSION + 1))

    NEW\_TAG="${VERSION\_PARTS[0]}.${VERSION\_PARTS[1]}.${NEW\_PATCH\_VERSION}"

    # Создаем новый аннотированный тег

    git tag -a "$NEW\_TAG" -m "Updated patch version"

    # Пушим новый тег в удалённый репозиторий

    git push origin "$NEW\_TAG"

    echo "Created new tag: $NEW\_TAG"

fi

# Удаляем временную директорию

cd ..

rm -rf "$TEMP\_DIR"

3. В WSL:

Переход в директорию:

cd /mnt/d/DevOps\_Belhard/HW15062025

Преобразование файл в Unix-формат:

dos2unix update\_tag.sh

Назначение файла исполняемым:

chmod +x update\_tag.sh

Запуск скрипта:

~~./update\_tag.sh~~

./update\_tag.sh https://github.com/TatsianaVladykina/DevOps\_Belhard

